Universidade de Brasília

DEPARTAMENTO DE ESTATÍSTICA

13 junho 2024

Entrega 5

Prof. Dr. George von Borries

Aluno: Bruno Gondim Toledo

Matrícula: 15/0167636

Aluno: Stefan Zurman Gonçalves

Matrícula: 19/0116994

Tópicos 2

1º/2024

12. Capítulo 9 de James et al.

1) Stefan

2) Stefan

3) Stefan

4) Stefan

5) Stefan

6) Stefan

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 Auto data set

Hint: In the lab, we used the plot() function for svm objects only in cases with p = 2. When p > 2, you can use the plot() function to create plots displaying pairs of variables at a time. Essentially, instead of typing

plot(svmfit, dat)

where svmfit contains your ftted model and dat is a data frame containing your data, you can type

plot(svmfit, dat, x1 ~ x4)

in order to plot just the frst and fourth variables. However, you must replace x1 and x4 with the correct variable names. To fnd out more, type ?plot.svm.

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 = Auto
median(Auto$mpg)
```

```
## [1] 22.75
```

```
Auto$mpg01 = factor(ifelse(Auto$mpg > median(Auto$mpg), 1, 0))

set.seed(150167636)
sample = sample.split(Auto, SplitRatio = .75)
train = subset(Auto, sample == TRUE)
test  = subset(Auto, sample == FALSE)

x <- subset(test, select = -mpg01)
y <- test$mpg01
```

b) Fit a support vector classifer 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 ft the classifer without the gas mileage variable to produce sensible results.

Para custo $c = 1$:

```
fit1 = svm(mpg01 ~ origin + year + acceleration + weight + horsepower + displacement + cylinders,
    data = train,
    kernel = "linear",
    cost = 1,
    cross = 5,
    )

summary(fit1)
```

```
##
## Call:
## svm(formula = mpg01 ~ origin + year + acceleration + weight + horsepower +
##     displacement + cylinders, data = train, kernel = "linear", cost = 1,
##     cross = 5, )
##
```

```
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  61
##
##  ( 31 30 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
##
## 5-fold cross-validation on training data:
##
## Total Accuracy: 90.18182
## Single Accuracies:
##  90.90909 81.81818 92.72727 87.27273 98.18182
```

```
pred1 <- predict(fit1, x)
table(pred1, y)
```

```
##       y
## pred1  0  1
##     0 50  6
##     1  8 53
```

Para custo $c = 2$:

```
fit2 = svm(mpg01 ~ origin + year + acceleration + weight + horsepower + displacement + cylinders,
    data = train,
    kernel = "linear",
    cost = 2,
    cross = 5,
    )

summary(fit2)
```

```
##
## Call:
## svm(formula = mpg01 ~ origin + year + acceleration + weight + horsepower +
##     displacement + cylinders, data = train, kernel = "linear", cost = 2,
##     cross = 5, )
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  2
##
## Number of Support Vectors:  58
##
##  ( 29 29 )
##
```

```
##
## Number of Classes:  2
##
## Levels:
##  0 1
##
## 5-fold cross-validation on training data:
##
## Total Accuracy: 90.54545
## Single Accuracies:
##  92.72727 92.72727 92.72727 96.36364 78.18182
```

```
pred2 <- predict(fit2, x)
table(pred2, y)
```

```
##       y
## pred2  0  1
##     0 50  6
##     1  8 53
```

Para cursto $c = 10$:

```
fit3 = svm(mpg01 ~ origin + year + acceleration + weight + horsepower + displacement + cylinders,
    data = train,
    kernel = "linear",
    cost = 10,
    cross = 5,
    )

summary(fit3)
```

```
##
## Call:
## svm(formula = mpg01 ~ origin + year + acceleration + weight + horsepower +
##     displacement + cylinders, data = train, kernel = "linear", cost = 10,
##     cross = 5, )
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  57
##
##  ( 28 29 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
##
## 5-fold cross-validation on training data:
##
## Total Accuracy: 89.81818
## Single Accuracies:
##  89.09091 92.72727 80 94.54545 92.72727
```

```r
pred3 <- predict(fit3, x)
table(pred3, y)
```

```
##       y
## pred3  0  1
##     0 53  5
##     1  5 54
```

Vemos que para o default da função, $c = 1$, o modelo ajustado à 75% dos dados para treinamento com k-fold, $k = 10$ obteve um erro de classificação de $\frac{6+8}{117} \approx 0,12$, mesmo resultado pro modelo com $c = 2$. Já para $c = 10$, o erro foi de $\frac{5+5}{117} \approx 0,085$. Portanto, o modelo com $c = 10$ foi o que obteve o menor erro de classificação, neste caso.

Diferentemente do procedimento adotado quando trabalhamos com modelos regressivos, não foi feita verificação se as variáveis explicativas eram significativas para o modelo. Apenas foi feita a análise do erro de classificação.

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

Obtendo valores ótimos para custo e gama iterativamente:

```r
ft = tune.svm(x=Auto[,2:8],y=Auto$mpg01,cost=1:10,gamma=seq(0,5,0.1))
ft$best.parameters
```

```
##    gamma cost
## 11     1    1
```

Para kernel polinomial, com custo $c = 1$ e $gamma = 1/7$(default):

```r
fit4 = svm(mpg01 ~ origin + year + acceleration + weight + horsepower + displacement + cylinders,
    data = train,
    kernel = "polynomial",
    cost = 1,
    cross = 5,
    )

summary(fit4)
```

```
##
## Call:
## svm(formula = mpg01 ~ origin + year + acceleration + weight + horsepower +
##     displacement + cylinders, data = train, kernel = "polynomial",
##     cost = 1, cross = 5, )
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  113
##
```

```
##  ( 55 58 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
##
## 5-fold cross-validation on training data:
##
## Total Accuracy: 92
## Single Accuracies:
##  94.54545 92.72727 90.90909 85.45455 96.36364
```

```
pred4 <- predict(fit4, x)
table(pred4, y)
```

```
##       y
## pred4  0  1
##     0 54 10
##     1  4 49
```

Para kernel polinomial, com custo $c = 2$ e $gamma = 0,7$:

```
fit5 = svm(mpg01 ~ origin + year + acceleration + weight + horsepower + displacement + cylinders,
    data = train,
    kernel = "polynomial",
    gamma = 0.7,
    cost = 2,
    cross = 5,
    )

summary(fit5)
```

```
##
## Call:
## svm(formula = mpg01 ~ origin + year + acceleration + weight + horsepower +
##     displacement + cylinders, data = train, kernel = "polynomial",
##     gamma = 0.7, cost = 2, cross = 5, )
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  2
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  55
##
##  ( 29 26 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
##
## 5-fold cross-validation on training data:
##
## Total Accuracy: 90.54545
## Single Accuracies:
##  90.90909 94.54545 90.90909 89.09091 87.27273
```

```r
pred5 <- predict(fit5, x)
table(pred5, y)
```

```
##       y
## pred5  0  1
##     0 55 11
##     1  3 48
```

Observamos resultados ligeiramente piores que para os modelos de kernel linear. Aplicando os parâmetros retornados pelo fine-tuning, o erro de classificação se manteve constante.

Para kernel radial, com custo $c = 1$ e $gamma = 1/7$(default):

```r
fit6 = svm(mpg01 ~ origin + year + acceleration + weight + horsepower + displacement + cylinders,
    data = train,
    kernel = "radial",
    cost = 1,
    cross = 5,
    )

summary(fit6)
```

```
##
## Call:
## svm(formula = mpg01 ~ origin + year + acceleration + weight + horsepower +
##     displacement + cylinders, data = train, kernel = "radial", cost = 1,
##     cross = 5, )
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  77
##
##  ( 39 38 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
##
## 5-fold cross-validation on training data:
##
## Total Accuracy: 90.54545
## Single Accuracies:
##  90.90909 94.54545 89.09091 87.27273 90.90909
```

```
pred6 <- predict(fit6, x)
table(pred6, y)
```

```
##      y
## pred6  0  1
##     0 50  4
##     1  8 55
```

Para kernel radial, com custo $c = 2$ e $gamma = 0,7$:

```
fit7 = svm(mpg01 ~ origin + year + acceleration + weight + horsepower + displacement + cylinders,
    data = train,
    kernel = "radial",
    gamma = 0.7,
    cost = 2,
    cross = 5,
    )

summary(fit7)
```

```
##
## Call:
## svm(formula = mpg01 ~ origin + year + acceleration + weight + horsepower +
##     displacement + cylinders, data = train, kernel = "radial", gamma = 0.7,
##     cost = 2, cross = 5, )
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  2
##
## Number of Support Vectors:  118
##
##  ( 60 58 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
##
## 5-fold cross-validation on training data:
##
## Total Accuracy: 93.81818
## Single Accuracies:
##  98.18182 96.36364 90.90909 94.54545 89.09091
```

```
pred7 <- predict(fit7, x)
table(pred7, y)
```

```
##      y
## pred7  0  1
##     0 53  5
##     1  5 54
```

Os resultados são todos diferentes, porém próximos se analisarmos somente o poder de classificação. Em todos os casos testados, o erro de classificação ficou em torno de 10%.

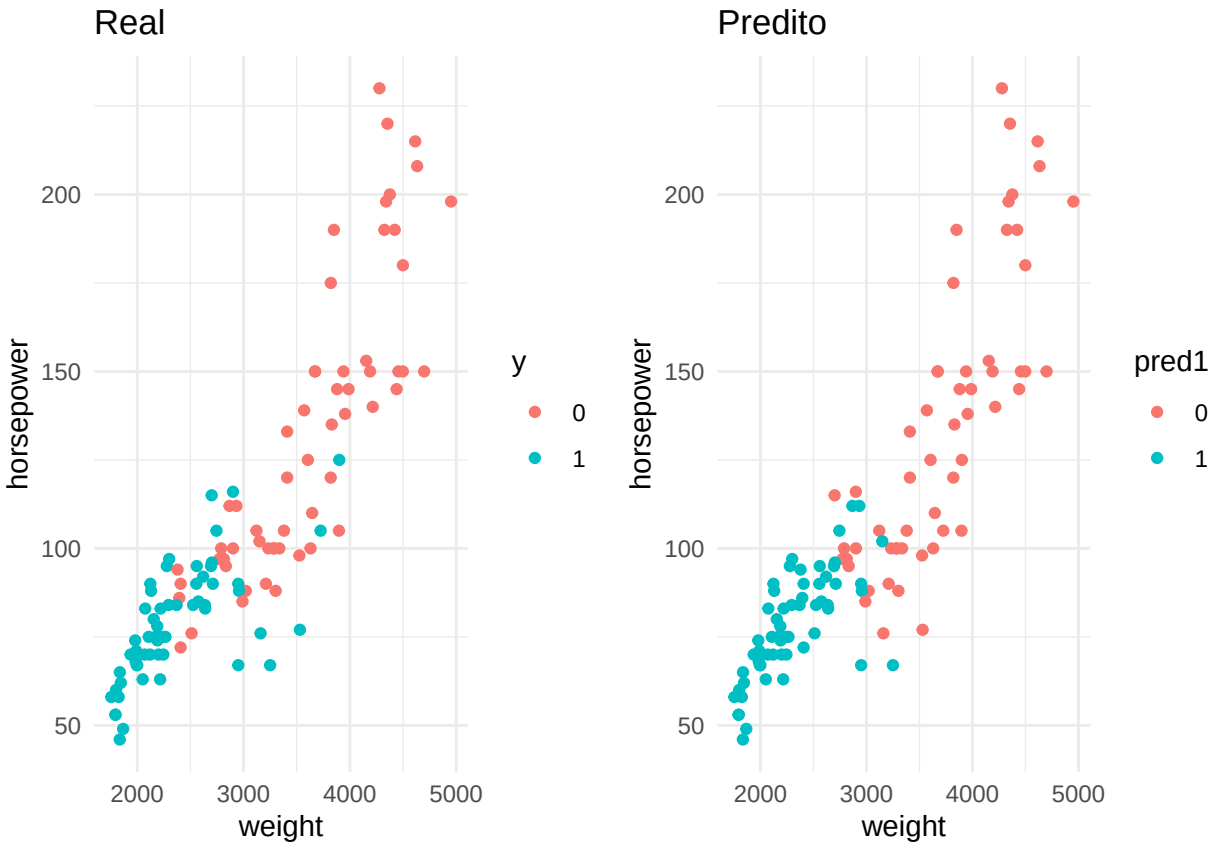
Note que o fine-tuning foi feito de forma ingênua, sem considerar intervalos diferentes além dos fornecido para a função, com o objetivo de rodar num tempo razoável. Talvez um fine-tuning mais refinado poderia ter obtido melhores resultados.

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

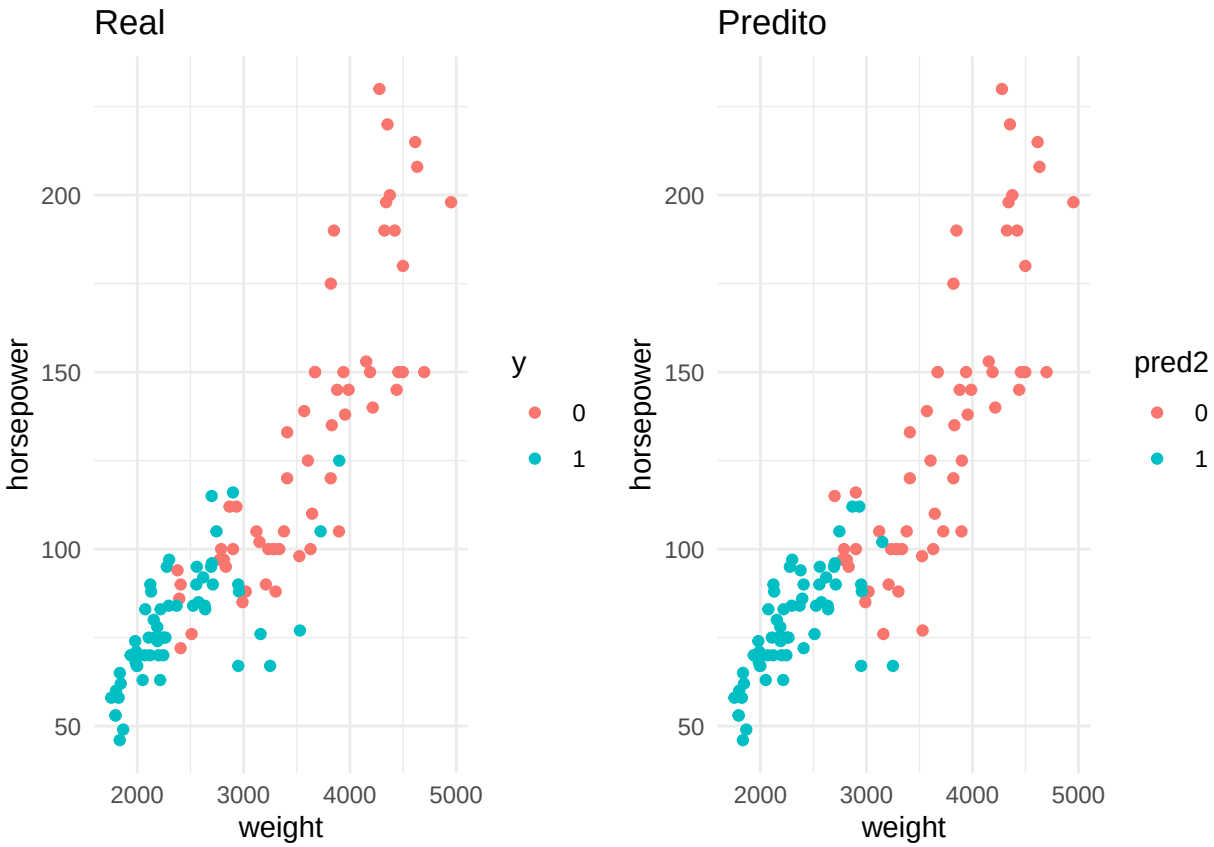
Não é possível uma visualização dos resultados, pois o número de variáveis explicativas é maior que 2. Entretanto, podemos ver duas a duas como as variáveis se comportam ante ao valor real x valor predito.

Escolherei as variáveis *weight* e *horsepower* para visualização, visto que intuitivamente poderíamos pensar que quanto mais pesado e/ou mais potente o carro, maior a probabilidade de ter um consumo de combustível mais alto. Veremos se os modelos capturam essa relação.

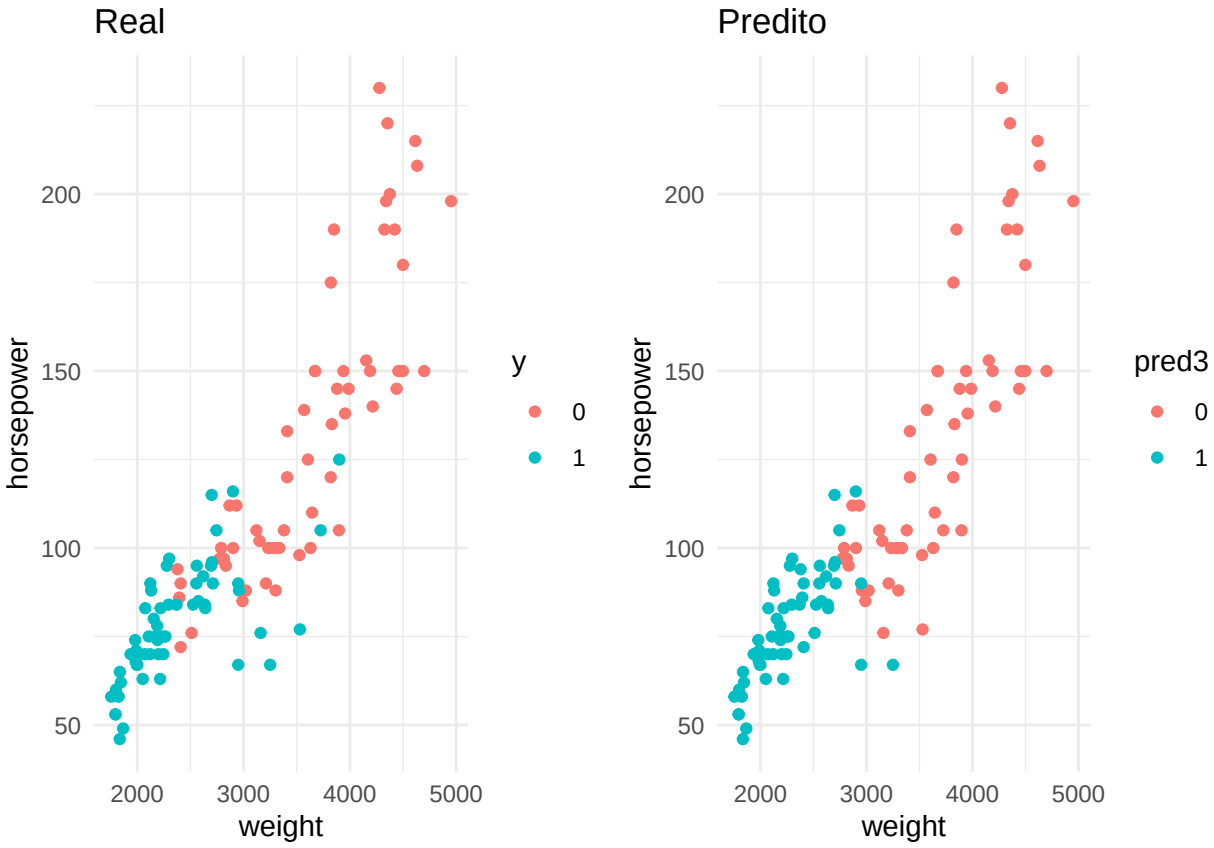
Modelo 1: Kernel linear, custo = 1



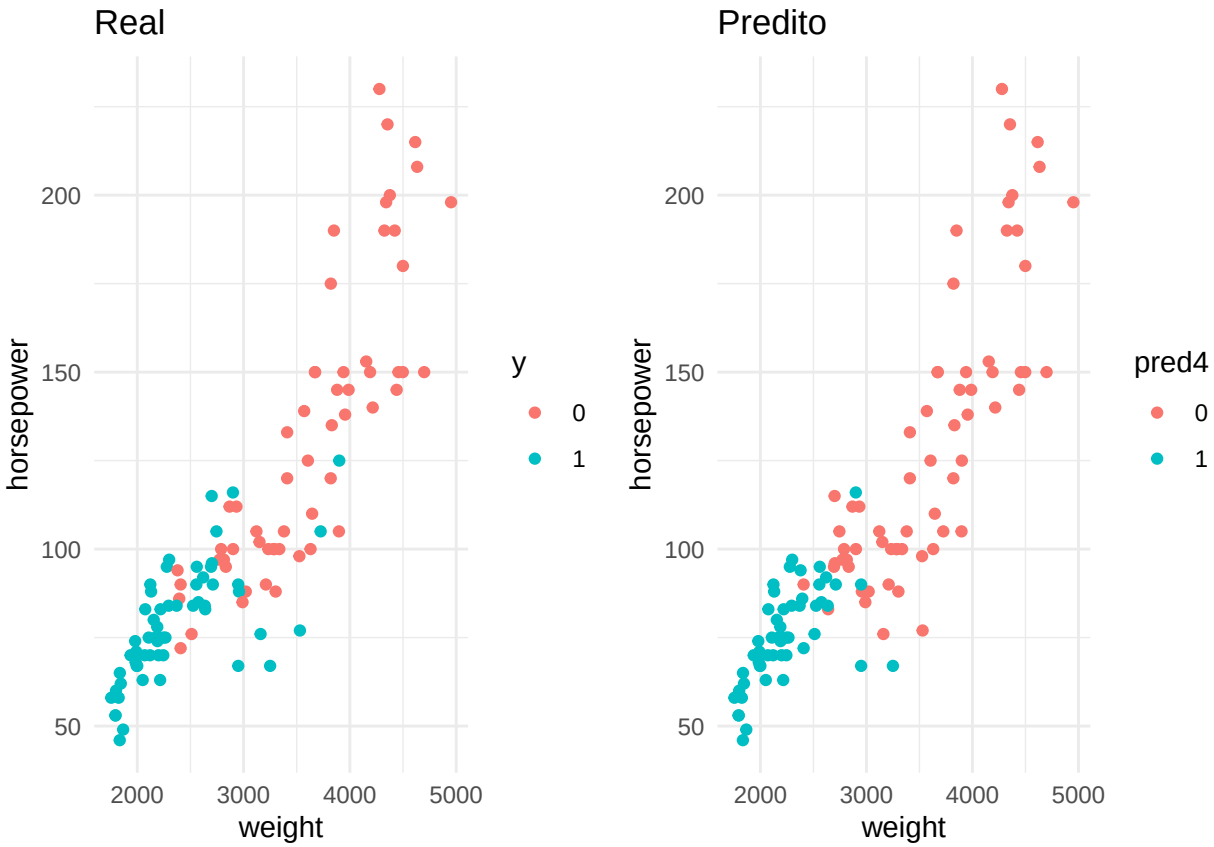
Modelo 2: Kernel linear, custo = 2



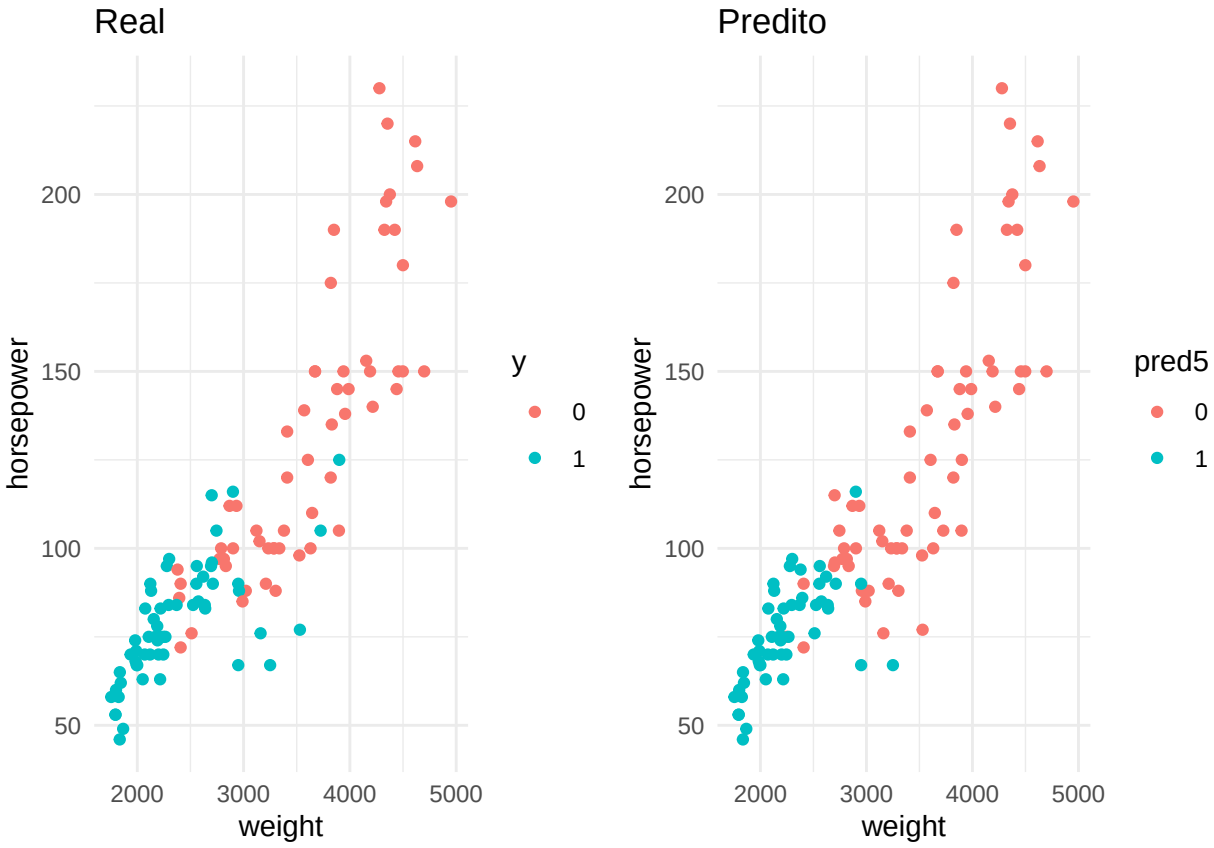
Modelo 3: Kernel linear, custo = 10



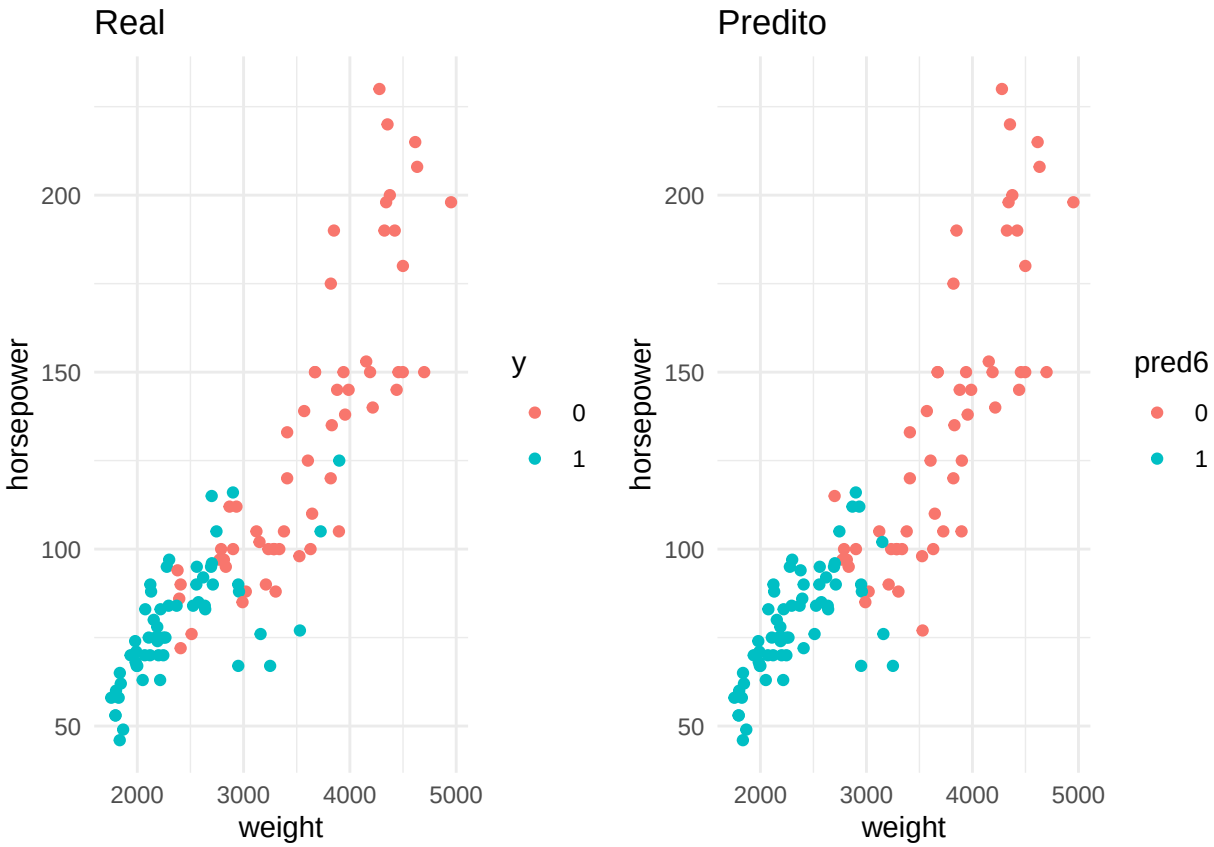
Modelo 4: kernel polinomial, com custo $c = 1$ e $gamma = 1/7$(default):



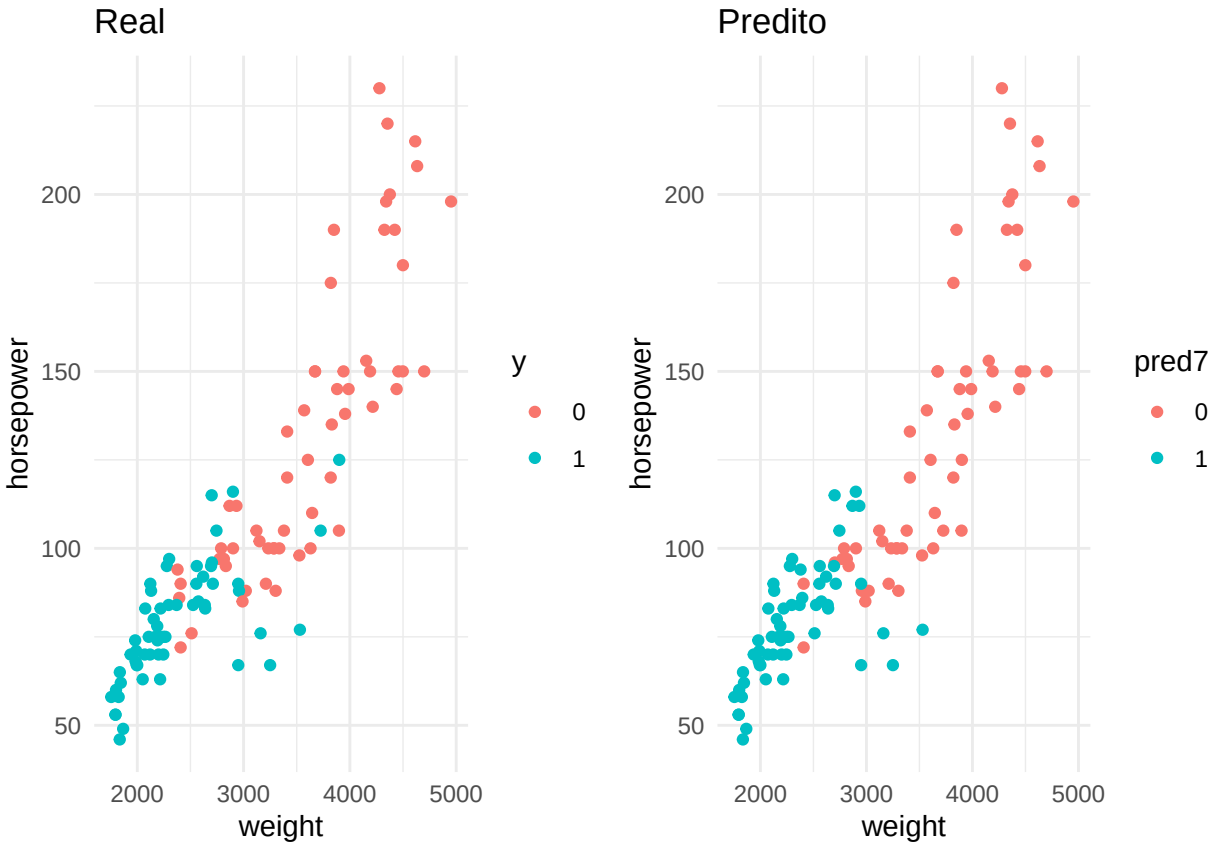
Modelo 5: kernel polinomial, com custo $c = 2$ e $gamma = 0,7$:



Modelo 6: kernel radial, com custo $c = 1$ e $gamma = 1/7$(default):



Modelo 7: kernel radial, com custo $c = 2$ e $gamma = 0,7$:



8) This problem involves the OJ data set which is part of the ISLR2 package.

```
oj = OJ
```

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

```
oj$id = 1:nrow(oj)
set.seed(150167636)
amostra = sample(1:nrow(oj),800)
train = oj |> filter(id %in% amostra) |> select(-id)
test = oj |> filter(!id %in% amostra) |> select(-id)

x <- subset(test, select = -Purchase)
y <- test$Purchase
```

b) Fit a support vector classifer to the training data using cost = 0.01, with Purchase as the response and the other variables as predictors. Use the summary() function to produce summary statistics, and describe the results obtained.

```
fit8 = svm(Purchase ~ ., data = train, kernel="linear", cost = 0.01)
summary(fit8)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, kernel = "linear", cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  441
##
##  ( 220 221 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

c) What are the training and test error rates?

```
pred8_1 <- predict(fit8, subset(train, select = -Purchase))
table(pred8_1, train$Purchase)
```

```
##
## pred8_1  CH  MM
##      CH 435  77
##      MM  60 228
```

```
pred8_2 <- predict(fit8, x)
table(pred8_2, y)
```

```
##         y
## pred8_2  CH  MM
##       CH 143  26
##       MM  15  86
```

Vemos que a taxa de erro de classificação do modelo ajustado sob os dados de treino são da ordem de $\frac{77+60}{800} \approx 0,17$, enquanto que para os dados de teste, a taxa de erro foi de $\frac{26+15}{270} \approx 0,15$.

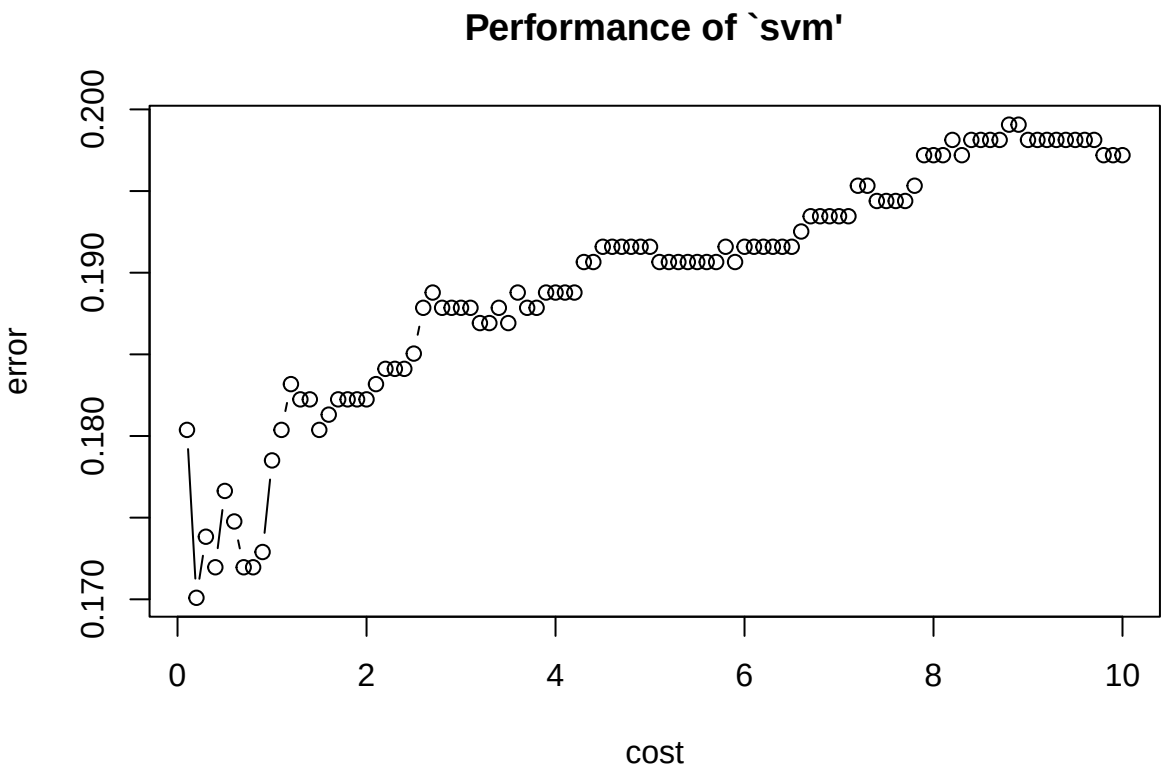
d) Use the tune() function to select an optimal cost. Consider values in the range 0.01 to 10.

```
obj <- tune.svm(Purchase~., data = oj, cost = seq(0.1,10,0.1))
```

```
knitr::kable(obj$best.parameters,row.names = FALSE)
```

cost
0.2

```
plot(obj)
```



A função portanto constata que o valor otimizado para custo seria $c = 0,7$.

e) Compute the training and test error rates using this new value for cost.

```
fit9 = svm(Purchase ~ ., data = train, kernel="linear", cost = 0.7)

pred9_1 <- predict(fit9, subset(train, select = -Purchase))
table(pred9_1, train$Purchase)
```

```
##
## pred9_1  CH  MM
##      CH 435  75
##      MM  60 230
```

```
pred9_2 <- predict(fit9, x)
table(pred9_2, y)
```

```
##        y
## pred9_2  CH  MM
##      CH 142  28
##      MM  16  84
```

Vemos que a taxa de erro de classificação do modelo reajustado com custo $= 0,7$ sob os dados de treino são da ordem de $\frac{75+60}{800} \approx 0,168$, enquanto que para os dados de teste, a taxa de erro foi de $\frac{26+15}{270} \approx 0,163$. Ou seja, diminuiu ligeiramente quando aplicado aos dados de treino, e aumentou ligeiramente quando aplicado aos dados de teste.

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

Para custo = 0,01:

```
fit10 = svm(Purchase ~ ., data = train, kernel="radial", cost = 0.01)

pred10_1 <- predict(fit10, subset(train, select = -Purchase))
table(pred10_1, train$Purchase)
```

```
##
## pred10_1  CH  MM
##       CH 495 305
##       MM   0   0
```

```
pred10_2 <- predict(fit10, x)
table(pred10_2, y)
```

```
##         y
## pred10_2  CH  MM
##       CH 158 112
##       MM   0   0
```

Vemos que para este custo, o erro de classificação foi de $\frac{305+0}{800} \approx 0,38$ para os dados de treino, e de $\frac{112+0}{270} \approx 0,42$ para os dados de teste, o que beira o inaceitável. Com um custo tão baixo, o modelo acaba sempre errando para uma das classes.

Para custo = 0,7:

```
fit11 = svm(Purchase ~ ., data = train, kernel="radial", cost = 0.7)

pred11_1 <- predict(fit11, subset(train, select = -Purchase))
table(pred11_1, train$Purchase)
```

```
##
## pred11_1  CH  MM
##       CH 455  78
##       MM  40 227
```

```
pred11_2 <- predict(fit11, x)
table(pred11_2, y)
```

```
##          y
## pred11_2  CH  MM
##       CH 143  32
##       MM  15  80
```

Vemos que para este custo, o erro de classificação foi de $\frac{78+48}{800} \approx 0,157$ para os dados de treino, e de $\frac{0+270}{270} \approx 0,174$ para os dados de teste, o que aproxima os resultados do kernel linear.

Vemos então que o kernel linear foi melhor em classificar com o custo baixo que o kernel radial. Para o custo ótimo, os valores convergem para ambos os modelos.

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

Para custo = 0,01:

```
fit12 = svm(Purchase ~ ., data = train, kernel="polynomial",degree = 2, cost = 0.01)

pred12_1 <- predict(fit12, subset(train, select = -Purchase))
table(pred12_1, train$Purchase)
```

```
##
## pred12_1  CH  MM
##       CH 495 300
##       MM   0   5
```

```
pred12_2 <- predict(fit12, x)
table(pred12_2, y)
```

```
##          y
## pred12_2  CH  MM
##       CH 158 112
##       MM   0   0
```

Vemos que para este custo, o erro de classificação foi de $\frac{300+0}{800} = 0,375$ para os dados de treino, e de $\frac{112+0}{270} \approx 0,42$ para os dados de teste, o que beira o inaceitável. Com um custo tão baixo, o modelo acaba sempre errando para uma das classes. Este resultado é similar ao kernel radial com o mesmo custo.

Para custo = 0,7:

```r
fit13 = svm(Purchase ~ ., data = train, kernel="radial",degree = 2, cost = 0.7)

pred13_1 <- predict(fit13, subset(train, select = -Purchase))
table(pred13_1, train$Purchase)
```

```
##
## pred13_1  CH  MM
##       CH 455  78
##       MM  40 227
```

```r
pred13_2 <- predict(fit13, x)
table(pred13_2, y)
```

```
##          y
## pred13_2  CH  MM
##       CH 143  32
##       MM  15  80
```

Vemos que para este custo, o erro de classificação foi de $\frac{78+40}{800} = 0,1475$ para os dados de treino, e de $\frac{32+15}{270} \approx 0,174$ para os dados de teste. Agora este modelo obteve resultados comparáveis ao kernel radial para custo = 0,7 e linear para ambos os custos.

h) Overall, which approach seems to give the best results on this data?

Neste caso, o melhor modelo parece ser o linear, por dois motivos:

1) É o modelo mais simples, então devemos segui-lo, pelo princípio da parcimônia.
2) Alcançou resultados comparáveis aos demais modelos, praticamente idêntico.

Como não houve uma substantiva melhora na classificação quando utilizamos os modelos com kernel radial e polinomial, ficar o modelo linear é o mais indicado.

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.

Kernel Linear

No R, primeiramente faremos um *fine-tuning* para descobrir o melhor parâmetro de custo para o modelo:

```r
obj <- tune.svm(Species~., data = iris, cost = seq(0.1,10,0.1))
cost = obj$best.parameters |> pull()
```

Em seguida, iremos dividir os dados em treino e teste, com proporção 70% e 30%, respectivamente. Depois, iremos ajustar o modelo linear utilizando o parâmetro de custo sugerido pelo *fine-tuning*, além de fazer partições do tipo *k-fold*, fixando $k = 5$ (arbitrário). Então, iremos utilizar o modelo para fazer previsão sobre os dados de teste, e compararemos com os valores reais para ter ideia da acurácia do modelo.

```r
set.seed(150167636)
sample = sample.split(iris, SplitRatio = .7)
train = subset(iris, sample == TRUE)
test  = subset(iris, sample == FALSE)

x <- subset(test, select = -Species)
y <- test$Species

fit_linear = svm(Species ~ .,
    data = train,
    kernel = "linear",
    cost = cost,
    cross = 5,
    )

summary(fit_linear)
```

```
##
## Call:
## svm(formula = Species ~ ., data = train, kernel = "linear", cost = cost,
##     cross = 5, )
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  5.8
##
## Number of Support Vectors:  17
##
##  ( 2 8 7 )
##
##
## Number of Classes:  3
##
## Levels:
##  setosa versicolor virginica
##
## 5-fold cross-validation on training data:
##
## Total Accuracy: 94.44444
## Single Accuracies:
##  88.88889 94.44444 94.44444 100 94.44444
```

```r
pred <- predict(fit_linear, x)
table(pred, y)
```

```
##             y
## pred         setosa versicolor virginica
##   setosa         20          0         0
##   versicolor      0         19         3
##   virginica       0          1        17
```

Vemos que o modelo linear obteve resultados excelentes, com acurácia de 100% para a espécie Setosa, 19 em 20 para Versicolor e 17 em 20 para Virginica.

Testaremos agora um *kernel* não-linear para ver se conseguimos melhorar o ajuste do modelo.

Em Julia, poderíamos fazer analogamente:

```
using LIBSVM
using RDatasets
using Printf
using Statistics

iris = dataset("datasets","iris")

X = Matrix(iris[:, 1:4])'

y = iris.Species

Xtrain = X[:, 1:2:end]
Xtest = X[:, 2:2:end]
ytrain = y[1:2:end]
ytest = y[2:2:end]

model = svmtrain(Xtrain, ytrain, kernel = Kernel.Linear, cost = 1.0)

yh , decision_values = svmpredict(model, Xtest);
```

```
@printf "Accuracy: %.2f%%\n" mean(yh .== ytest) * 100
```

```
## Accuracy: 97.33%
```

Kernel Não Linear

Os dados já estão separados em treino e teste, portanto, só precisamos realizar o *fine-tuning* para descobrir os parâmetros do modelo de *kernel* radial, que será o custo e o gama.

```
obj = tune.svm(Species~., data = iris, cost = seq(0.1,10,0.1),gamma=seq(0,5,0.1))
cost = obj$best.parameters
```

Com os valores obtidos do *fine-tuning*, plugamos os parâmetros no modelo e realizamos o ajuste do modelo, também com *k-fold* $k = 5$. Após, iremos utilizar para fazer predições sobre os mesmos dados de teste, e verificar o ajuste do modelo ante os dados reais.

```
fit_nl = svm(Species ~ .,
    data = train,
    kernel = "radial",
    cost = cost |> select(cost) |> pull(),
    gamma = cost |> select(gamma) |> pull(),
    cross = 5,
    )

summary(fit_nl)
```

```
##
## Call:
## svm(formula = Species ~ ., data = train, kernel = "radial", cost = pull(select(cost,
##     cost)), gamma = pull(select(cost, gamma)), cross = 5, )
##
##
## Parameters:
```

```
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  2.3
##
## Number of Support Vectors:  36
##
##  ( 3 16 17 )
##
##
## Number of Classes:  3
##
## Levels:
##  setosa versicolor virginica
##
## 5-fold cross-validation on training data:
##
## Total Accuracy: 96.66667
## Single Accuracies:
##  88.88889 100 100 100 94.44444
```

```
pred <- predict(fit_nl, x)
table(pred, y)
```

```
##             y
## pred         setosa versicolor virginica
##   setosa         20          0         0
##   versicolor      0         19         2
##   virginica       0          1        18
```

Percebemos que o modelo SVM com *kernel* não-linear foi mais acurado que o modelo linear. Os mesmos 100% de acerto de classificação para Setosa foram obtidos, assim como 19 em 20 Versicolor, porém conseguiu 19 em 20 Virginica. Neste caso, o modelo não é tão mais sofisticado, no sentido de complexidade de programação e tempo de execução, portanto, acredito que podemos abrir mão do princípio de parsimônia neste caso e optar pelo modelo mais preciso, que é o de *kernel* radial.

É claro que se a explicação do modelo for mais importante que a sua acurácia, o modelo linear é mais simples de ser explicado, e tem uma taxa de acerto quase tão boa quanto, então também poderia ser utilizado, no lugar do modelo com *kernel* radial.