05.Árvores de Regressão e Regressão Logística

Felipe Rocha

2022-12-17

Aula 05: Árvore de Regressão

Antes de tudo, carregando os pacotes

```
#Instalar Pacote
#install.packages("tree")
#install.packages("randomForest")

library(tree) #Fitting Regression Trees
library(MASS) #Carregar o banco de dados
library(randomForest) #Bagging and Random Forests
library(nnet)
```

8.3.2 Fitting Regression Trees

Ajustando a Árvore

```
#Utilizando dados do R (Boston - Valores de casas no suburbio de Boston)
data(Boston)
head(Boston)
```

```
##      crim zn indus chas   nox    rm  age    dis rad tax ptratio  black lstat
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90  4.98
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90  9.14
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83  4.03
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63  2.94
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90  5.33
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12  5.21
##   medv
## 1 24.0
## 2 21.6
## 3 34.7
## 4 33.4
## 5 36.2
## 6 28.7
```

```r
#Informacoes sobre o arquivo de dados podem ser acessadas
#?Boston

#Particionar o conjunto de dados (Treinamento e validacao)
set.seed(1)
train = sample (1:nrow(Boston), nrow(Boston)/2)

#Ajuste de uma arvore
tree.boston <- tree::tree(medv ~ ., data = Boston, subset=train)

#Resumo (Apenas uma variavel foi utilizada na construcao da arvore
summary(tree.boston)
```

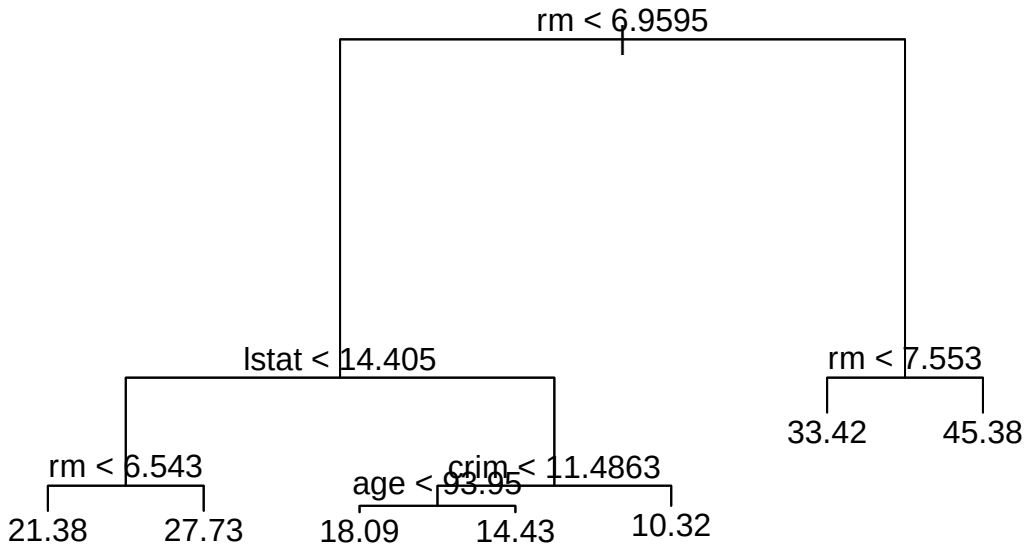
```
##
## Regression tree:
## tree::tree(formula = medv ~ ., data = Boston, subset = train)
## Variables actually used in tree construction:
## [1] "rm"    "lstat" "crim"  "age"
## Number of terminal nodes:  7
## Residual mean deviance:  10.38 = 2555 / 246
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -10.1800  -1.7770  -0.1775   0.0000   1.9230  16.5800
```

Observe que a saída de *summary()* indica que apenas quatro das variáveis foram usadas na construção da árvore para prever **medv** (valor médio de casas ocupadas pelos proprietários em$ 1000s).

- **rm**: número médio de quartos por habitação;
- **lstat**: percentual de indivíduos com nível socioeconômico mais baixo;
- **crim**: taxa de criminalidade per capita por cidade;
- **age**: proporção de unidades ocupadas pelo proprietário construídas antes de 1940.

No contexto de uma árvore de regressão, o *deviance* é simplesmente a soma dos erros quadrados da árvore. Agora plotamos a árvore.

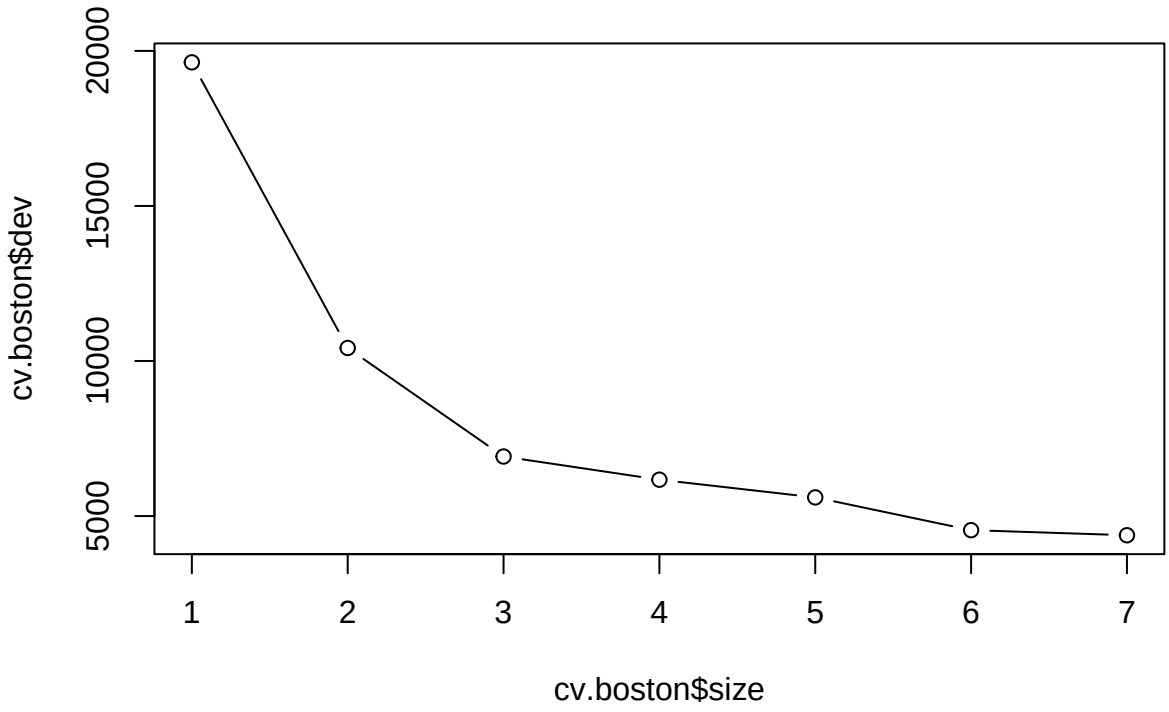
```r
#Apresentacao da arvore ajustada
plot(tree.boston)
text(tree.boston)
```



A árvore indica que valores mais altos de **rm** (quartos) correspondem a casas mais caras (valore médio de US 48.380). A árvore prevê que para poucos quartos ($rm < 6.9595$), baixa renda ($lstat > 14.405$) e altas taxas de crimes ($crim > 11.4863$) possuem as casas com menor valor médio, em torno de US 10.320.

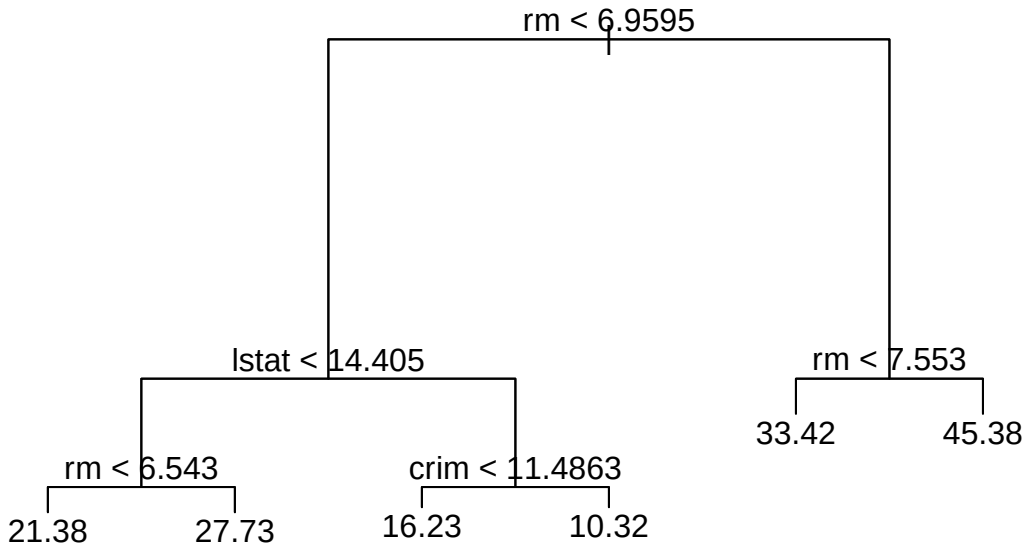
Em algumas situações, essa árvore pode ser super-ajustada, ou seja, causar o problema do over-fitting, que significa se ajustar bem no treinamento mas não trabalhar bem na validação. Agora usamos a função *cv.tree()* para ver se a poda da árvore melhora o desempenho.

```
cv.boston=tree::cv.tree(tree.boston)
plot(cv.boston$size ,cv.boston$dev ,type='b')
```



Verificando que de 7 para 6 o deviance se reduz de forma insignificante mas de 6 para 5 já é mais significante, podemos fazer a polda em 5.

```
#Se desejar podar
mod_poda=tree::prune.tree(tree.boston,best=6)
plot(mod_poda)
text(mod_poda, pretty =0)
```



E agora temos apenas 6 nós terminais (após a poda).

Fazendo Predições

```
yhat=predict(mod_poda ,newdata=Boston[- train ,])
boston.test=Boston[-train ,"medv"]
REQM <- sqrt(mean((yhat -boston.test)^2))
REQM
```

```
## [1] 5.929957
```

Em outras palavras, o conjunto de teste MSE associado à árvore de regressão é 35,28688. A raiz quadrada do MSE é, portanto, em torno de 5,940276, indicando que esse modelo leva a previsões de teste que estão dentro de cerca de US$ 5.940 do verdadeiro valor médio de residências no subúrbio.

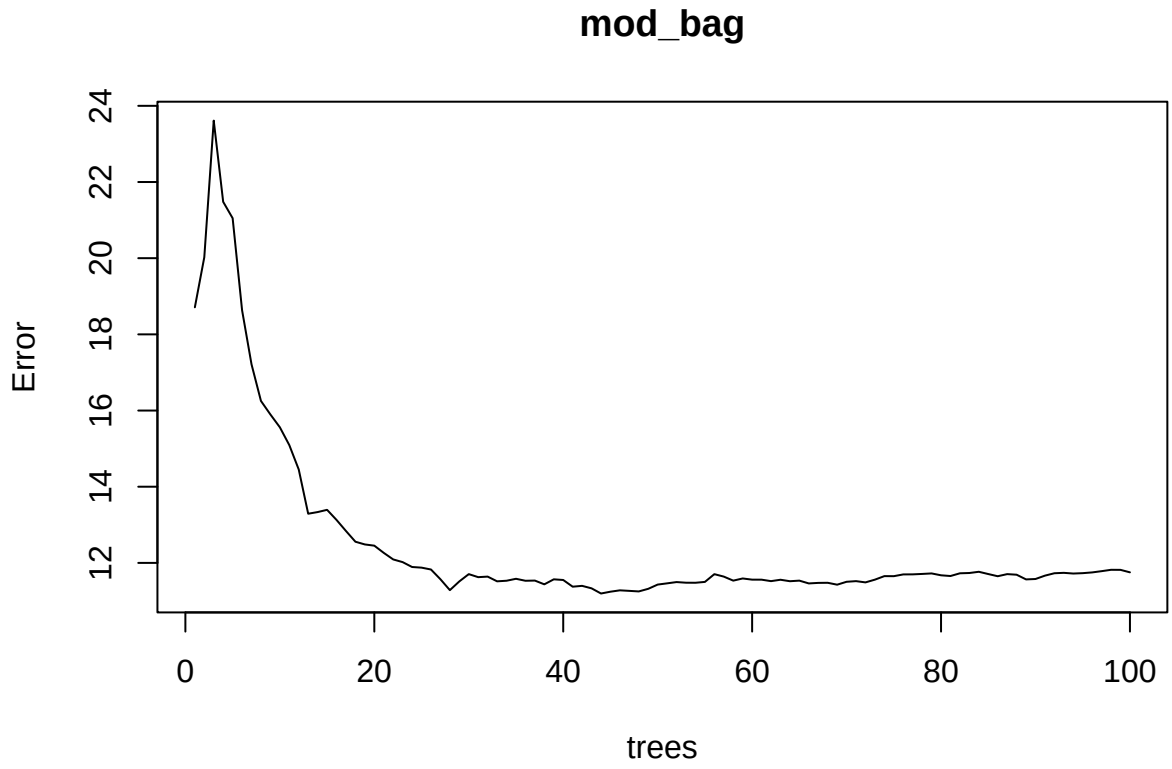
8.3.3 Bagging and Random Forests

Um dos problemas das árvores de regressão é o baixo poder preditivo. A literatura sugere algumas alternativas, dentre eles o **Bagging** e o **Random Forests**.

```
mod_bag = randomForest::randomForest( medv ~ .,data=Boston , subset=train ,
mtry=ncol(Boston)-1,importance =TRUE, ntree = 100)
mod_bag
```

```
##
## Call:
##  randomForest(formula = medv ~ ., data = Boston, mtry = ncol(Boston) -      1, importance = TRUE, nt
##                Type of random forest: regression
##                      Number of trees: 100
## No. of variables tried at each split: 13
##
##           Mean of squared residuals: 11.75203
##                     % Var explained: 84.71
```

```
plot(mod_bag)
```



O argumento

$$mtry = 13$$

indica que todos os 13 preditores devem ser considerados para cada divisão da árvore — em outras palavras, o **Bagging** deve ser feito e não o **Random Forests**.

```
#Fazendo predição
yhat.bag = predict (mod_bag , newdata=Boston[-train ,])
#Avaliando o modelo
REQM_bag <- sqrt(mean((yhat.bag - boston.test)^2))
REQM_bag
```

```
## [1] 4.942389
```

O conjunto de teste MSE associado à árvore de regressão bagged é 23,23893 (e RQME 4.820678), menro que a árvore ajustada anteriormente. Poderíamos alterar o número de árvores cultivadas por *randomForest()* usando o argumento **ntree** que, ao invés de utilziar todas as variáveis, utiliza apenas uma certa quantidade.

A literatura sugere utilizar $\sqrt{p}$ quando se constroe uma random forest de classificação e $\frac{p}{3}$ quando se trata de uma random forest de regressão. No nosso exemplo, usaremos $\frac{13}{3} = 4$ aproximadamente.

```
mod_rf = randomForest::randomForest(medv ~ .,data=Boston , subset=train ,
mtry=4, importance =TRUE)
yhat.rf = predict(mod_rf,newdata=Boston[- train ,])
REQM_rf <- sqrt(mean((yhat.rf-boston.test)^2))
REQM_rf
```

```
## [1] 4.297153
```

Que resultou em uma melhoria em relação ao **Bagging**.

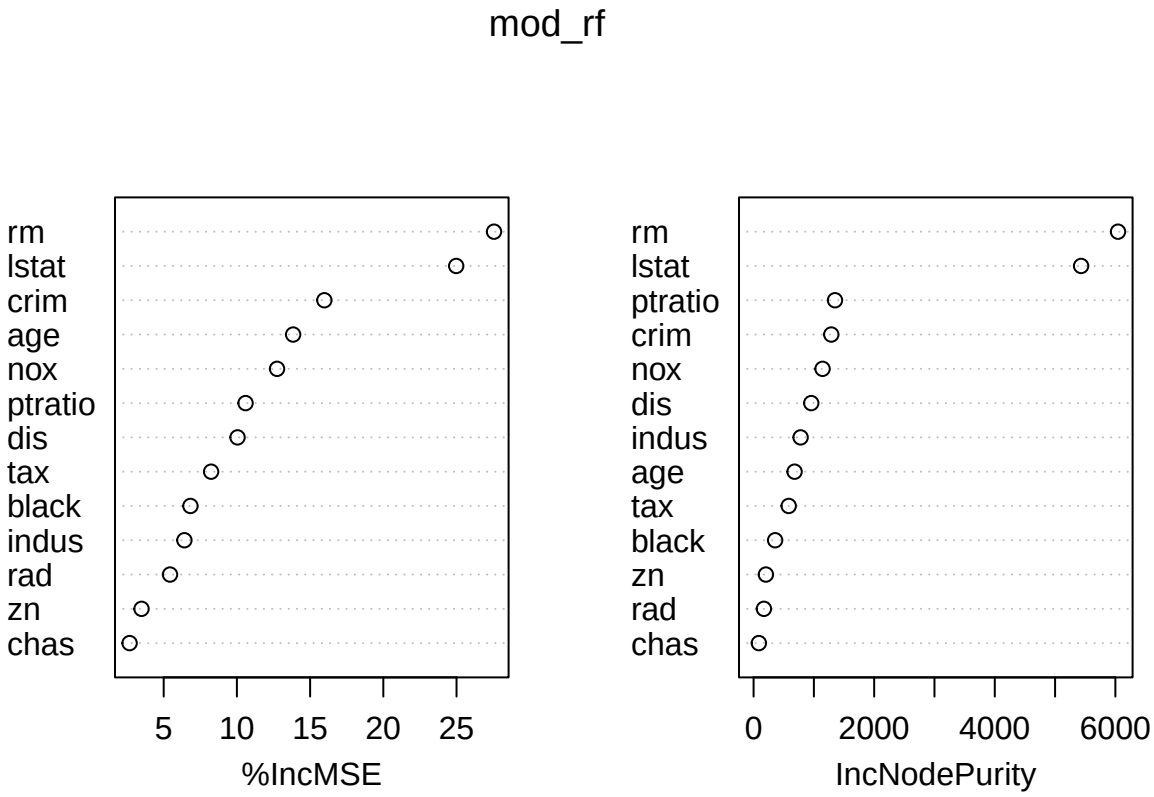
Usando a função **important()**, podemos ver a importância de cada variável.

```
#Importância (Baseado nas amostras out-of_bag randomForest )
i_mod_rf <-randomForest::importance(mod_rf)
i_mod_rf
```

```
##           %IncMSE IncNodePurity
## crim    15.972407    1288.76533
## zn       3.486962     204.35374
## indus    6.412381     779.69802
## chas     2.670205      87.76351
## nox     12.742027    1143.90586
## rm      27.559356    6043.88317
## age     13.840752     679.90177
## dis     10.042468     956.35075
## rad      5.432088     171.14024
## tax      8.238144     582.68790
## ptratio 10.591407    1350.27872
## black    6.825984     357.26501
## lstat   24.976651    5431.82100
```

Duas medidas de importância variável são relatadas. O primeiro é baseado na diminuição média da precisão nas previsões das amostras fora da sacola quando uma determinada variável é excluída do modelo. O último é uma medida da diminuição total na impureza do nó que resulta das divisões sobre essa variável, calculada a média de todas as árvores. No caso de **Random Forests regressão**, a impureza do nó é medida pelo RSS de treinamento, e para **Random Forests de classificação** pelo deviance. Gráficos dessas medidas de importância podem ser produzidos usando a função *varImpPlot()*.

```
randomForest::varImpPlot(mod_rf)
```



Os resultados indicam que em todas as árvores consideradas na **Random Forests**, o nível de riqueza da comunidade(**lstat**) e o tamanho da casa (**rm**) são de longe as duas variáveis mais importantes.

Aula 06: Regressão Logística

```
##Leitura de dados

#Modelo de Probabilidade Linear
dados<-read.table("F:/Estudo/Estudo---UFV/UFV---GitHub/01.Datasets/dados_doenca.txt", h=T)
head(dados)
```

```
##    ano doenca    tm    ur
## 1 1987      1 30.14 82.86
## 2 1988      0 30.66 79.57
## 3 1989      0 26.10 89.14
## 4 1990      1 28.43 91.00
## 5 1991      0 29.57 80.57
## 6 1992      1 31.25 67.82
```

Modelo Linear

```r
#Ajuste do modelo
reg<-lm(doenca ~ tm + ur, data=dados)
summary(reg)
```

```
##
## Call:
## lm(formula = doenca ~ tm + ur, data = dados)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.8359 -0.1656  0.1292  0.2596  0.4109
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -10.74306    4.80208  -2.237   0.0557 .
## tm            0.28588    0.12031   2.376   0.0448 *
## ur            0.03536    0.01897   1.864   0.0993 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4469 on 8 degrees of freedom
## Multiple R-squared:  0.4143, Adjusted R-squared:  0.2678
## F-statistic: 2.829 on 2 and 8 DF,  p-value: 0.1177
```

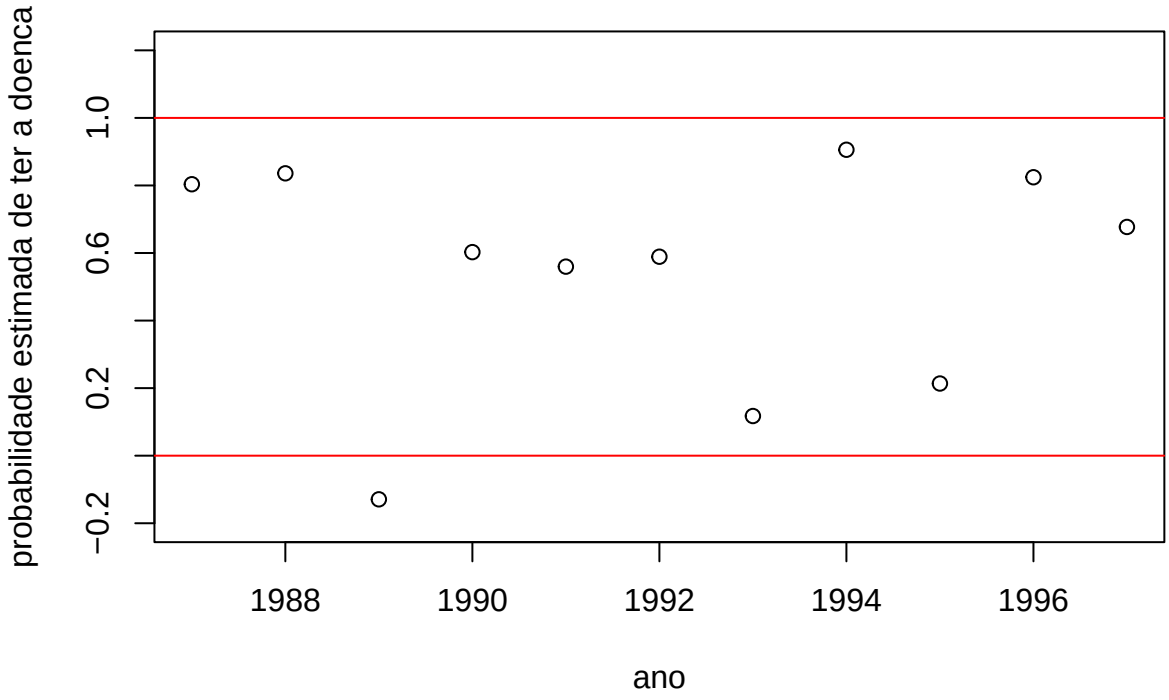
Como não estamos interessados em inferência e sim em calssificaçã, podemos acessar as classificações por meio do *fitted*.

```r
#Probabilidades do ano ter a doenca
data.frame(ano=dados$ano,probabilidade=reg$fitted)
```

```
##     ano probabilidade
## 1  1987     0.8036191
## 2  1988     0.8359253
## 3  1989    -0.1292359
## 4  1990     0.6026376
## 5  1991     0.5596835
## 6  1992     0.5890570
## 7  1993     0.1174565
## 8  1994     0.9057419
## 9  1995     0.2136531
## 10 1996     0.8243142
## 11 1997     0.6771478
```

E assim temos a probabilidade de ter ocorrido a doença naquele determinado ano.

```r
#Figura (anos em funcaodas probabilidades)
plot(dados$ano,reg$fitted, xlab="ano",
ylab="probabilidade estimada de ter a doenca", ylim=c(-0.2,1.2))
#Adicionando espaco parametrico de probabilidade
abline(h=0, col="red")
abline(h=1, col="red")
```



Observe que temos um valor fora do intervalo de probabilida $[0, 1]$, o que é um problema do modelo de probabilidade linear.

Fazendo a classificação, temos

```
#Acessando medidas de qualidade
prob<-reg$fitted.values
prob
```

```
##          1          2          3          4          5          6          7
##  0.8036191  0.8359253 -0.1292359  0.6026376  0.5596835  0.5890570  0.1174565
##          8          9         10         11
##  0.9057419  0.2136531  0.8243142  0.6771478
```

```
#Classificando em classes
reg.classe<- ifelse(prob > 0.5, 1, 0)
reg.classe
```

```
##  1  2  3  4  5  6  7  8  9 10 11
##  1  1  0  1  1  1  0  1  0  1  1
```

Por meio dessa classificação, podemos contruir nossa tabela de confusão, que é muito importante para que possamos avaliar a capacidade preditiva do modelo.

```r
#Tabela de Confusão
tabela <- table(dados$doenca,reg.classe)
tabela
```

```
##    reg.classe
##     0 1
##   0 3 2
##   1 0 6
```

Na diagonal pricipalz temos as classificações corretas e na outra, as erradas. A partir dessa tabela, é possível calcular outras medidas.

```r
#Taxa de Erro Aparente (Percentual de erros)
TEA <- 1 - sum(diag(tabela))/sum(tabela)
TEA
```

```
## [1] 0.1818182
```

```r
#Acuracia (Percentual de acertos)
AC <- sum(diag(tabela))/sum(tabela)
AC
```

```
## [1] 0.8181818
```

```r
#Sensibilidade (taxa de Verdadeiros Positivos)
S <- tabela[2,2]/ sum(tabela[2,])
S
```

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

```r
#Especificidade (taxa de Verdadeiros Negativos)
E <- tabela[1,1]/ sum(tabela[1,])
E
```

```
## [1] 0.6
```

Além do problema de termos probabilidades fora do intervalo [0,1], o modelo linear considera que a relação entre as varáveis independentes e a variável resposta como linear, o que na prática nem sempre é real. Para solucionar esse problema, temos outros modelos.

Modelo Logit

```r
#Modelo Logit (Nao lineariadade entre Y e variaveis explicativas)
logit <- glm(doenca ~ tm + ur , data = dados, family=binomial(link="logit"))

#Valores de probabilidade estimados
prob<-logit$fitted.values
```

```r
#Classificao
prob<-logit.classe<- ifelse(prob > 0.5, 1, 0)

#Acessando medidas de qualidade
#Tabela de confusao
tabela <- table(dados$doenca,logit.classe)
tabela
```

```
##    logit.classe
##     0 1
##   0 4 1
##   1 1 5
```

```r
#Taxa de Erro Aparente
TEA <- 1 - sum(diag(tabela))/sum(tabela)
TEA
```

```
## [1] 0.1818182
```

```r
#Acuracia
AC <- sum(diag(tabela))/sum(tabela)
AC
```

```
## [1] 0.8181818
```

```r
#Sensibilidade (Verdadeiros Positivos)
S <- tabela[2,2]/ sum(tabela[2,])
S
```

```
## [1] 0.8333333
```

```r
#Especificidade (Verdadeiros Negativos)
E <- tabela[1,1]/ sum(tabela[1,])
E
```

```
## [1] 0.8
```

Modelo Logit Multinomial

Para quando tenho mais de duas classes

```r
#Logit multinomial (Mais de duas classes
#Instalar pacote
#install.packages("nnet")
#library(nnet)

#leitura dos dados
dados<-iris
head(dados)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
#Ajuste
logitm <- nnet::multinom(Species ~ Sepal.Length + Sepal.Width + Petal.Length +
                           Petal.Width, data = dados)
```

```
## # weights:  18 (10 variable)
## initial  value 164.791843
## iter  10 value 16.177348
## iter  20 value 7.111438
## iter  30 value 6.182999
## iter  40 value 5.984028
## iter  50 value 5.961278
## iter  60 value 5.954900
## iter  70 value 5.951851
## iter  80 value 5.950343
## iter  90 value 5.949904
## iter 100 value 5.949867
## final  value 5.949867
## stopped after 100 iterations
```

```
#Probabilidades estimadas
prob_logitm <- predict(logitm, dados[,1:4], "probs")
prob_logitm
```

```
##           setosa   versicolor    virginica
## 1   1.000000e+00 1.526406e-09 2.716417e-36
## 2   9.999996e-01 3.536476e-07 2.883729e-32
## 3   1.000000e+00 4.443506e-08 6.103424e-34
## 4   9.999968e-01 3.163905e-06 7.117010e-31
## 5   1.000000e+00 1.102983e-09 1.289946e-36
## 6   1.000000e+00 3.521573e-10 1.344907e-35
## 7   1.000000e+00 4.098064e-08 3.016154e-33
## 8   1.000000e+00 2.615330e-08 2.972971e-34
## 9   9.999871e-01 1.294210e-05 7.048364e-30
## 10  9.999992e-01 8.386603e-07 1.454198e-32
## 11  1.000000e+00 2.161864e-10 1.241888e-37
## 12  9.999997e-01 3.238036e-07 1.545112e-32
## 13  9.999992e-01 8.320656e-07 1.402024e-32
## 14  9.999998e-01 1.776283e-07 6.091969e-34
## 15  1.000000e+00 2.490019e-14 4.289244e-44
## 16  1.000000e+00 5.099113e-14 5.053040e-42
## 17  1.000000e+00 1.180774e-12 1.043681e-39
## 18  1.000000e+00 1.119797e-09 1.233997e-35
## 19  1.000000e+00 2.229749e-10 1.278090e-36
## 20  1.000000e+00 3.414358e-10 1.306813e-36
## 21  9.999999e-01 5.088458e-08 1.418328e-33
```

```
## 22  1.000000e+00 5.983234e-10 2.761055e-35
## 23  1.000000e+00 3.282647e-11 2.381898e-39
## 24  9.999998e-01 2.467861e-07 6.662407e-30
## 25  9.999768e-01 2.323802e-05 1.868716e-29
## 26  9.999965e-01 3.538327e-06 1.482164e-30
## 27  9.999999e-01 5.849351e-08 6.536682e-32
## 28  1.000000e+00 3.674991e-09 1.310414e-35
## 29  1.000000e+00 2.112377e-09 5.720335e-36
## 30  9.999968e-01 3.188981e-06 7.381858e-31
## 31  9.999956e-01 4.413191e-06 1.554498e-30
## 32  1.000000e+00 1.585769e-09 2.578398e-34
## 33  1.000000e+00 2.696754e-11 2.849881e-40
## 34  1.000000e+00 3.875622e-13 2.425003e-42
## 35  9.999994e-01 6.152555e-07 6.606045e-32
## 36  1.000000e+00 2.079286e-09 5.317228e-36
## 37  1.000000e+00 4.138112e-11 1.071492e-38
## 38  1.000000e+00 2.595111e-09 6.271520e-37
## 39  9.999987e-01 1.303796e-06 1.422388e-31
## 40  1.000000e+00 1.515201e-08 1.346082e-34
## 41  1.000000e+00 4.651074e-10 2.558009e-36
## 42  9.997542e-01 2.458213e-04 1.376952e-26
## 43  9.999998e-01 2.285045e-07 6.575528e-33
## 44  1.000000e+00 1.317919e-08 2.900340e-31
## 45  9.999999e-01 7.470478e-08 7.649899e-32
## 46  9.999996e-01 4.478126e-07 2.893285e-31
## 47  1.000000e+00 1.934115e-09 3.064974e-36
## 48  9.999997e-01 3.187312e-07 1.436229e-32
## 49  1.000000e+00 3.731511e-10 2.742847e-37
## 50  1.000000e+00 1.503286e-08 1.297787e-34
## 51  2.427101e-07 9.999877e-01 1.201699e-05
## 52  2.160475e-07 9.999501e-01 4.968516e-05
## 53  4.640834e-09 9.987828e-01 1.217158e-03
## 54  4.185792e-10 9.999567e-01 4.326447e-05
## 55  2.752538e-09 9.985711e-01 1.428890e-03
## 56  7.824187e-11 9.998954e-01 1.045901e-04
## 57  2.356899e-08 9.986727e-01 1.327314e-03
## 58  3.195371e-07 9.999997e-01 5.641233e-10
## 59  6.116463e-09 9.999850e-01 1.497847e-05
## 60  1.501151e-08 9.999848e-01 1.523161e-05
## 61  9.809848e-10 1.000000e+00 4.165185e-08
## 62  1.773719e-07 9.999615e-01 3.834000e-05
## 63  1.060055e-09 9.999999e-01 1.034374e-07
## 64  1.308456e-10 9.991850e-01 8.150241e-04
## 65  4.002682e-05 9.999600e-01 1.436141e-08
## 66  1.418052e-06 9.999957e-01 2.908759e-06
## 67  4.799737e-10 9.986481e-01 1.351871e-03
## 68  6.658268e-09 1.000000e+00 1.551529e-08
## 69  1.127345e-11 9.401019e-01 5.989806e-02
## 70  9.220385e-09 9.999999e-01 9.072544e-08
## 71  2.958914e-10 5.945365e-01 4.054635e-01
## 72  8.608392e-07 9.999988e-01 3.522422e-07
## 73  7.324234e-13 7.743208e-01 2.256792e-01
## 74  2.950369e-11 9.999586e-01 4.141866e-05
## 75  1.473401e-07 9.999984e-01 1.455234e-06
```

```
## 76  3.439354e-07 9.999924e-01 7.246952e-06
## 77  6.017178e-10 9.992755e-01 7.245125e-04
## 78  2.112470e-10 7.236305e-01 2.763695e-01
## 79  1.784210e-09 9.990177e-01 9.822717e-04
## 80  8.317614e-06 9.999917e-01 1.361048e-10
## 81  9.293464e-09 9.999999e-01 8.816365e-08
## 82  2.833280e-08 1.000000e+00 5.553317e-09
## 83  2.136523e-07 9.999997e-01 9.050639e-08
## 84  1.096390e-14 1.323524e-01 8.676476e-01
## 85  1.609647e-10 9.977885e-01 2.211499e-03
## 86  1.892766e-07 9.997823e-01 2.175106e-04
## 87  2.692561e-08 9.996965e-01 3.034535e-04
## 88  1.105514e-10 9.997399e-01 2.600700e-04
## 89  7.714596e-08 9.999991e-01 8.170920e-07
## 90  2.388398e-09 9.999886e-01 1.141228e-05
## 91  1.403301e-11 9.999591e-01 4.089587e-05
## 92  1.299698e-09 9.998366e-01 1.633724e-04
## 93  2.152323e-08 9.999995e-01 4.518083e-07
## 94  2.308979e-07 9.999998e-01 8.584159e-10
## 95  1.362045e-09 9.999845e-01 1.546367e-05
## 96  2.350697e-08 9.999997e-01 2.643923e-07
## 97  1.341431e-08 9.999968e-01 3.187736e-06
## 98  4.945474e-08 9.999976e-01 2.382636e-06
## 99  2.224095e-04 9.997776e-01 6.500522e-11
## 100 2.333746e-08 9.999976e-01 2.420920e-06
## 101 9.453717e-25 2.718072e-10 1.000000e+00
## 102 2.762230e-17 3.922358e-04 9.996078e-01
## 103 2.413930e-20 9.974371e-07 9.999990e-01
## 104 1.039086e-18 2.851578e-04 9.997148e-01
## 105 4.877802e-22 9.409138e-08 9.999999e-01
## 106 8.139586e-26 4.698713e-09 1.000000e+00
## 107 2.747116e-14 1.091926e-01 8.908074e-01
## 108 1.841814e-22 4.609074e-06 9.999954e-01
## 109 4.655966e-22 8.093448e-06 9.999919e-01
## 110 1.116285e-20 7.196079e-09 1.000000e+00
## 111 3.360175e-12 9.861345e-03 9.901387e-01
## 112 2.824675e-17 2.619406e-04 9.997381e-01
## 113 2.887245e-17 2.057044e-05 9.999794e-01
## 114 1.356507e-18 3.348943e-05 9.999665e-01
## 115 6.643324e-20 8.391928e-08 9.999999e-01
## 116 1.443873e-16 4.987152e-06 9.999950e-01
## 117 2.506556e-16 2.325939e-03 9.976741e-01
## 118 8.132508e-22 7.823403e-08 9.999999e-01
## 119 1.539275e-32 6.473411e-13 1.000000e+00
## 120 2.586465e-16 7.964338e-02 9.203566e-01
## 121 5.888460e-19 3.959256e-07 9.999996e-01
## 122 6.580602e-16 4.950994e-04 9.995049e-01
## 123 3.543398e-27 3.830263e-09 1.000000e+00
## 124 7.099730e-13 5.193896e-02 9.480610e-01
## 125 1.158605e-17 1.805360e-05 9.999819e-01
## 126 1.014284e-17 4.479026e-04 9.995521e-01
## 127 1.384328e-11 1.760948e-01 8.239052e-01
## 128 1.238609e-11 1.980731e-01 8.019269e-01
## 129 5.264982e-21 7.894776e-07 9.999992e-01
```

```
## 130 1.067125e-15 2.892881e-02 9.710712e-01
## 131 2.185577e-21 3.215285e-06 9.999968e-01
## 132 9.900467e-17 8.276525e-05 9.999172e-01
## 133 1.158989e-21 1.274946e-07 9.999999e-01
## 134 5.926801e-13 7.939466e-01 2.060534e-01
## 135 8.716903e-19 3.353546e-02 9.664645e-01
## 136 1.196029e-21 1.736953e-08 1.000000e+00
## 137 2.573884e-19 1.415958e-07 9.999999e-01
## 138 5.272004e-16 3.535048e-03 9.964650e-01
## 139 4.984248e-11 3.310585e-01 6.689415e-01
## 140 3.159583e-15 1.313812e-04 9.998686e-01
## 141 6.087418e-20 5.142118e-08 9.999999e-01
## 142 1.851909e-13 5.774763e-05 9.999423e-01
## 143 2.762230e-17 3.922358e-04 9.996078e-01
## 144 2.348662e-21 4.707320e-08 1.000000e+00
## 145 2.720648e-20 1.227942e-08 1.000000e+00
## 146 7.661759e-16 7.065708e-06 9.999929e-01
## 147 7.146172e-16 9.093936e-04 9.990906e-01
## 148 1.470964e-14 1.023609e-03 9.989764e-01
## 149 6.009635e-17 4.504137e-06 9.999955e-01
## 150 2.726745e-14 2.243538e-02 9.775646e-01
```

```
#Classes
classe <- apply(prob_logitm, 1, which.max)
head(classe)
```

```
## 1 2 3 4 5 6
## 1 1 1 1 1 1
```

```
#Codificacao original
classe[which(classe=="1")] <- levels(dados$Species)[1]
classe[which(classe=="2")] <- levels(dados$Species)[2]
classe[which(classe=="3")] <- levels(dados$Species)[3]
head(classe)
```

```
##        1        2        3        4        5        6
## "setosa" "setosa" "setosa" "setosa" "setosa" "setosa"
```

```
#Acessando medidas de qualidade
#Tabela de confusao
tabela1<-table(dados$Species, classe)
tabela1
```

```
##             classe
##              setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         49         1
##   virginica       0          1        49
```

```
##Taxa de Erro Aparente
TEA <- 1 - sum(diag(tabela1))/sum(tabela1)
TEA
```

```
## [1] 0.01333333
```

```
#
AC < - sum(diag(tabela1))/sum(tabela1)
```

```
## [1] FALSE
```

```
AC
```

```
## [1] 0.8181818
```