Árvores de Regressão

Guia Prático

Luís Torgo
FEP, Universidade do Porto
ltorgo@liacc.up.pt

27 de Novembro de 2004

Este guia tem como objectivo ajudar os(as) alunos(as) na utilização do R para obter árvores de regressão.

1. Construír Árvores de Regressão

Nesta primeira secção vamos aprender como obter árvores de regressão no R. O R tem duas packages principais associadas a este tipo de modelos, *tree* e *rpart*. No entanto, a última destas é considerada uma package mais "sofisticada", e será essa que vamos usar neste guia.

Vejamos então um exemplo de utilização. Começemos por carregar o package:

```
> library(rpart)
```

A função principal deste package, chama-se `rpart` e serve para obter árvores de regressão. De facto, a mesma função pode ser usada para obter árvores de classificação, sem qualquer alteração, sendo a própria função que, dependendodo tipo da variável objectivo especificada, decide qual o tipo de árvores que é construída.

Na nossa exemplificação deste tipo de modelos, vamos usar os conhecidos dados *Boston*. Começemos por os carregar,

```
> data(Boston, package = "MASS")
```

Se quiser saber mais alguma informação sobre este conjunto de dados poderá obter ajuda sobre eles com o seguinte comando,

```
> help("Boston", package = "MASS")
```

Para obter uma árvore de regressão para prever a variável *medv* fazemos,

```
> arv <- rpart(medv ~ ., Boston)
```

Se quisermos ter uma ideia da árvore obtida podemos fazer,

```
> arv

n= 506

node), split, n, deviance, yval
      * denotes terminal node

 1) root 506 42716.3000 22.53281
   2) rm< 6.941 430 17317.3200 19.93372
     4) lstat>=14.4 175  3373.2510 14.95600
       8) crim>=6.99237 74  1085.9050 11.97838 *
       9) crim< 6.99237 101  1150.5370 17.13762 *
     5) lstat< 14.4 255  6632.2170 23.34980
      10) dis>=1.5511 248  3658.3930 22.93629
        20) rm< 6.543 193  1589.8140 21.65648 *
        21) rm>=6.543 55   643.1691 27.42727 *
      11) dis< 1.5511 7  1429.0200 38.00000 *
   3) rm>=6.941 76  6059.4190 37.23816
     6) rm< 7.437 46  1899.6120 32.11304
      12) lstat>=9.65 7   432.9971 23.05714 *
      13) lstat< 9.65 39   789.5123 33.73846 *
     7) rm>=7.437 30  1098.8500 45.09667 *
```

2. Visualizar Árvores de Regressão

Na secção anterior vimos já uma forma de obter uma primeira ideia de quais os nós que formam a árvore obtida. Vamos agora ver como obter uma representação gráfica deste tipo de modelos. Tal representação pode ser obtida fazendo,

```
> plot(arv)
> text(arv)
```

O resultado destes comandos é mostrado na Figura 1,

A representação obtida não é muito apelativa. Poderemos melhorá-la (ver Figura 2) usando alguns parâmetros das funções usadas, como por exemplo fazendo,

```
> plot(arv, uniform = T, branch = 0.2, margin = 0.1, compress = T,
+     nspace = 2)
> text(arv, fancy = T, cex = 0.7, font = 10, fwidth = 0.5, fheight = 0.7,
+     use.n = T, all = T, digits = 2, pretty = 0)
```

Se quiser explorar mais alguns detalhes das funções poderá usar o help das mesmas. Note, no entanto, que as funções que são de facto usadas (devido à orientação aos objectos das funções do R) são a função `plot.rpart` e `text.rpart`.

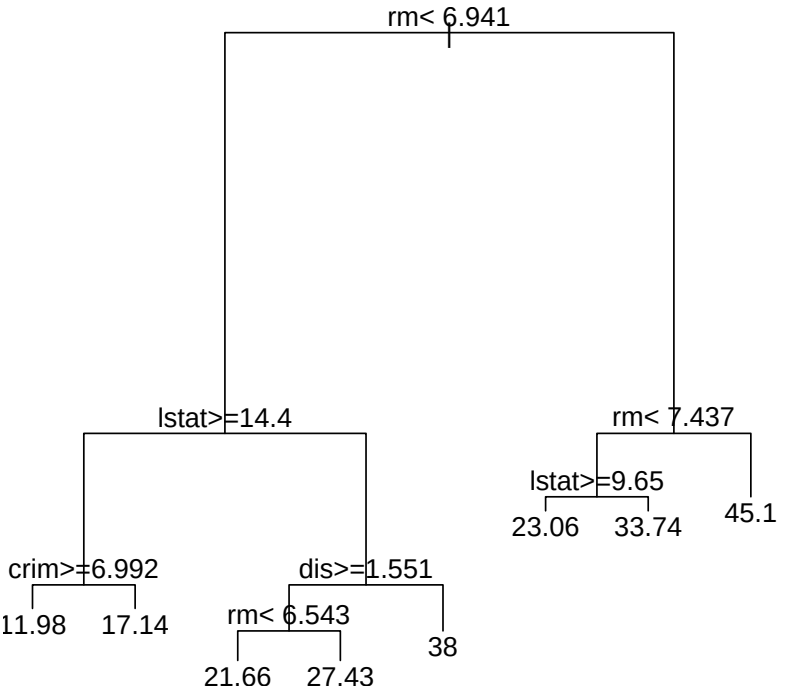

Figura 1: Uma representação gráfica da árvore.

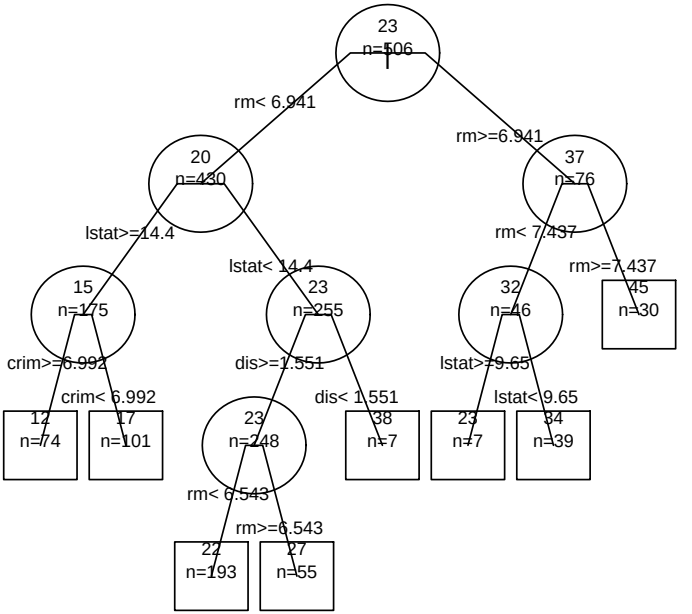

Figura 2: Uma representação mais sofisticada da árvore.

3. Simplificação das Árvores (Pruning)

As árvores obtidas pela função rpart, contêm informação que pode ser usada para fazer o *pruning* das árvores (ver aulas teóricas). Concretamente a função `rpart` retorna informação, obtida por um processo interno de validação cruzada, do erro estimado para a árvore obtida bem como para uma série de sub-árvores deste modelo. Essa informação pode ser obtida por,

```
> printcp(arv)

Regression tree:
rpart(formula = medv ~ ., data = Boston)

Variables actually used in tree construction:
[1] crim  dis   lstat rm

Root node error: 42716/506 = 84.42

n= 506

        CP nsplit rel error  xerror     xstd
1 0.452744      0   1.00000 1.00675 0.083394
2 0.171172      1   0.54726 0.62516 0.054477
3 0.071658      2   0.37608 0.43272 0.044907
4 0.036164      3   0.30443 0.34978 0.040968
5 0.033369      4   0.26826 0.35441 0.043525
6 0.026613      5   0.23489 0.34711 0.043465
7 0.015851      6   0.20828 0.31367 0.042361
8 0.010000      7   0.19243 0.29776 0.041767
```

Esta tabela indica, para cada valor do parâmetro de *cost complexity pruning* (CP), qual o valor do erro relativo na amostra de treino, bem como o erro estimado por validação cruzada e o respectivo erro padrão desta estimativa. A cada valor de CP corresponde uma sub-árvore, sendo a última linha da tabela a árvore retornada pela função `rpart`, e a primeira linha uma sub-árvore formada unicamente por uma folha (ou seja sem qualquer *split*).

Na Figura 3 vemos o mesmo tipo de informação de forma gráfica.

Usando esta informação poderemos decidir fazer o pruning da árvore inicialmente obtida pela função `rpart`. Isto pode ser conseguido pela função `prune` que permite explicitar um valor para CP sobre o qual queremos fazer pruning. Por exemplo olhando para a tabela anterior poderiamos querer fazer pruning de modo a ficarmos com uma árvore contendo unicamente 4 testes (*splits*), em vez dos 7 da árvore original obtida pela função `rpart`. Para conseguirmos tal árvore poderiamos usar a função `prune` do seguinte modo,

```
> prune(arv, cp = 0.034)

n= 506

node), split, n, deviance, yval
      * denotes terminal node

 1) root 506 42716.300 22.53281
   2) rm< 6.941 430 17317.320 19.93372
     4) lstat>=14.4 175  3373.251 14.95600 *
     5) lstat< 14.4 255  6632.217 23.34980
      10) dis>=1.5511 248  3658.393 22.93629 *
      11) dis< 1.5511 7  1429.020 38.00000 *
   3) rm>=6.941 76  6059.419 37.23816
     6) rm< 7.437 46  1899.612 32.11304 *
     7) rm>=7.437 30  1098.850 45.09667 *
```

O parâmetro cp da função `prune` permite establecer um limite em função dos valores da tabela, para realizar o corte. Em concreto a árvore é cortada no nível com o número de testes correspondentes à linha com o maior valor de cp menor do que o número indicado no parâmetro.

```
> plotcp(arv)
```

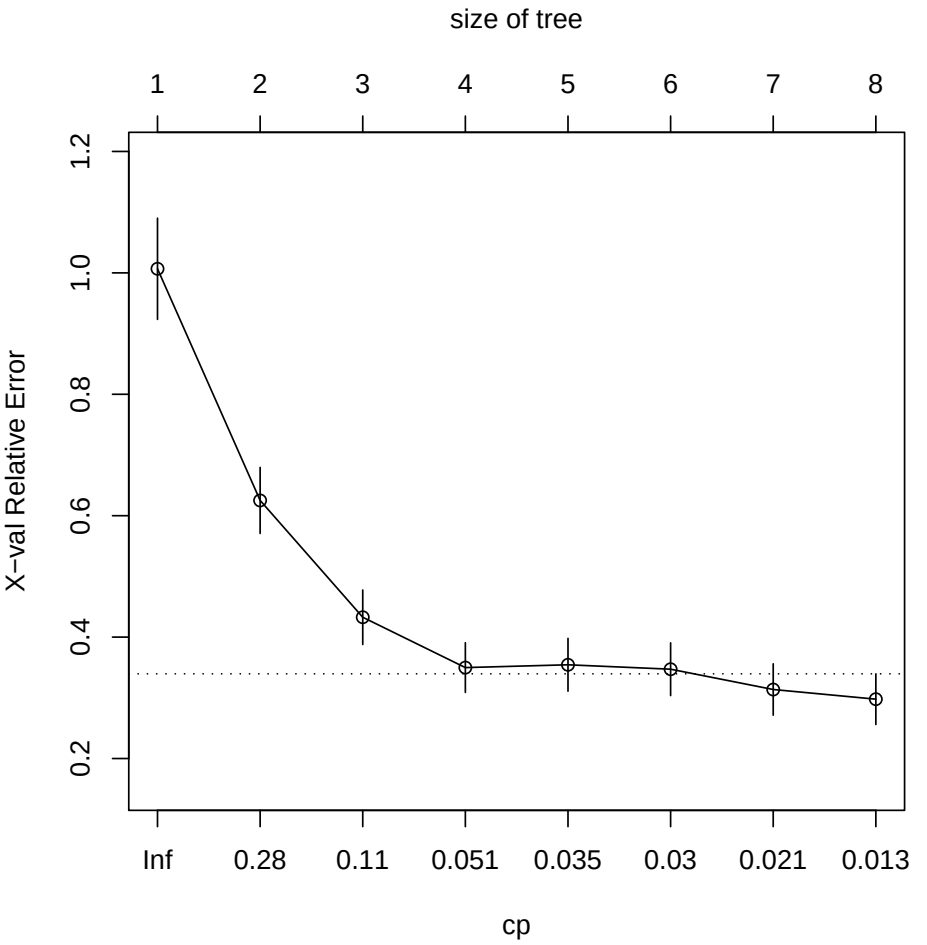

Figura 3: A tabela de *cost complexity prunning*.

Se quisermos podemos guardar a árvore resultante num objecto que para todos os efeitos é um objecto idêntico (em termos do que podemos fazer com ele) ao objecto inicial (antes do *pruning*),

```
> arv.prun <- prune(arv, cp = 0.034)
```

Uma regra bastante usada para fazer pruning com base em estimativas por validação cruzada (por exemplo usada no sistema CART [1]), é escolher a menor árvore na sequência apresentada pela função `printcp`, que tenha um erro dentro do intervalo $err_{best} \pm \sigma_{err_{best}}$, em que err_{best} é o erro menor dos erros estimados por validação cruzada, e $\sigma_{err_{best}}$ o respectivo erro padrão. Esta regra é muitas vezes conhecida como a regra 1-SE. Para isso o que normalmente se faz é crescer um árvore muito grande, e depois usar essa estratégia para escolher a "melhor" árvore de entre a sequência de sub-árvores dessa árvore enorme fornecidas pela função `rpart`. Vejamos um exemplo deste tipo de utilização,

```
> arv.grande <- rpart(medv ~ ., Boston, cp = 0)
> xval.prune.rpart <- function(tree, se = 1) {
+     lin.min.err <- which.min(tree$cptable[, 4])
+     tol.err <- tree$cptable[lin.min.err, 4] + se * tree$cptable[lin.min.err,
+         5]
+     se.lin <- which(tree$cptable[, 4] <= tol.err)[1]
+     prune.rpart(tree, cp = tree$cptable[se.lin, 1] + 1e-07)
+ }
> arv.final <- xval.prune.rpart(arv.grande)
```

Para obter uma árvore grande usamos um dos parâmetro da função `rpart`, que determinam quando o crescimento da árvore pára. Consulte o help da função para saber mais detalhes.

A árvore resultante é,

```
> arv.final

n= 506

node), split, n, deviance, yval
      * denotes terminal node

 1) root 506 42716.30000 22.532810
   2) rm< 6.941 430 17317.32000 19.933720
     4) lstat>=14.4 175  3373.25100 14.956000
       8) crim>=6.99237 74  1085.90500 11.978380
        16) nox>=0.6055 62   552.28840 11.077420
          32) lstat>=19.645 44   271.79180  9.913636 *
          33) lstat< 19.645 18    75.23111 13.922220 *
        17) nox< 0.6055 12   223.26670 16.633330 *
       9) crim< 6.99237 101  1150.53700 17.137620
        18) nox>=0.531 77   672.46310 16.238960 *
        19) nox< 0.531 24   216.37960 20.020830 *
     5) lstat< 14.4 255  6632.21700 23.349800
      10) dis>=1.5511 248  3658.39300 22.936290
        20) rm< 6.543 193  1589.81400 21.656480
          40) lstat>=7.57 150  1164.11300 20.993330 *
          41) lstat< 7.57 43   129.63070 23.969770 *
        21) rm>=6.543 55   643.16910 27.427270 *
      11) dis< 1.5511 7  1429.02000 38.000000 *
   3) rm>=6.941 76  6059.41900 37.238160
     6) rm< 7.437 46  1899.61200 32.113040
      12) lstat>=9.65 7   432.99710 23.057140 *
      13) lstat< 9.65 39   789.51230 33.738460 *
     7) rm>=7.437 30  1098.85000 45.096670
      14) ptratio>=17.6 7   465.96860 38.885710 *
      15) ptratio< 17.6 23   280.66610 46.986960 *
```

4. Utilização das Árvores para Previsão

A utilização das árvores de regressão para obter previsões segue o modelo standard da maioria das funções do R, o uso da função `predict`. Vejamos um exemplo concreto,

```
> treino.idxs <- sample(nrow(Boston), as.integer(0.7 * nrow(Boston)))
> treino <- Boston[treino.idxs, ]
> teste <- Boston[-treino.idxs, ]
> arv.grande <- rpart(medv ~ ., treino, cp = 0)
> arv.final <- xval.prune.rpart(arv.grande)
> arv.prevs <- predict(arv.final, teste)
> (mae.arv <- mean(abs(arv.prevs - teste[, "medv"])))

[1] 3.550345

> (mse.arv <- mean((arv.prevs - teste[, "medv"])^2))

[1] 27.78631
```

5. Exercícios

1. Verifique qual o erro numa amostra de teste separada, de 3 árvores escolhidas por si da sequência de sub-árvores de uma árvore obtida nos dados Boston.

2. Compare de igual forma o erro das árvores 1-SE, 0.5-SE e 1.5-SE

Referências

[1] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Statistics/Probability Series. Wadsworth & Brooks/Cole Advanced Books & Software, 1984.