

Fundamentos de la programación estadística y Data Mining en R

Unidad 3. Regresión Logística y Árboles de Decisión en R

Dr. Germán Rosati (Digital House - UNTREF - UNSAM)

19 julio, 2017

Implementación de un árbol de decisión mediante CART en R

Usaremos la librería `tree` para construir árboles de decisión y clasificación. Vamos a trabajar con el dataset `carseats`. En este dataset `Sales` (la variable dependiente) es una variable continua. Entonces, vamos a usar `ifelse()` para recodificar en dos valores. Y la agregamos a `Carseats`

```
library(tree)
library(ISLR)
attach(Carseats)
High <- ifelse(Sales <= 8, "No", "Yes")
Carseats <- data.frame(Carseats, High)
```

Ahora usamos la función `tree` para generar un árbol de clasificación que prediga la variable nueva (dicotómica que acabamos de crear). Como vamos a ver la sintaxis de `tree()` es muy similar a la de `lm()` y `glm()`.

```
tree.carseats <- tree(High ~ . - Sales, data = Carseats)
summary(tree.carseats)
```

```
##
## Classification tree:
## tree(formula = High ~ . - Sales, data = Carseats)
## Variables actually used in tree construction:
## [1] "ShelveLoc" "Price" "Income" "CompPrice" "Population"
## [6] "Advertising" "Age" "US"
## Number of terminal nodes: 27
## Residual mean deviance: 0.4575 = 170.7 / 373
## Misclassification error rate: 0.09 = 36 / 400
```

Vemos que el Training Error es de 9%. En árboles de clasificación lo que reporta la función como “mean deviance” en el output de `summary()` es: $-2 \sum_m \sum_k n_{mk} * \log * \hat{p}_{mk}$ donde n_{mk} es el número de observaciones en la m -ésima terminal que corresponde a la k -ésima clase. Una desviación pequeña indica un árbol que provee un buen ajuste para los datos (de entrenamiento). La residual mean deviance es simplemente ese valor dividido por $n - |T_0|$.

Implementación de un árbol de decisión mediante CART en R: gráficos

Una cosa interesante y útil que proveen los árboles de decisión es que permiten tener una salida gráfica bastante útil e intuitiva del modelo. Usamos la función `plot()` para graficar la estructura y `text()` para agregarle las “etiquetas”. El argumento `pretty=0` le dice a R que muestre las categorías completas de cada cualquier predictor cualitativo.

```
plot(tree.carseats)
text(tree.carseats, pretty = 0)
```



```

##          43) Price > 122.5 77  55.540 No ( 0.88312 0.11688 )
##          86) CompPrice < 147.5 58  17.400 No ( 0.96552 0.03448 ) *
##          87) CompPrice > 147.5 19  25.010 No ( 0.63158 0.36842 )
##          174) Price < 147 12  16.300 Yes ( 0.41667 0.58333 )
##          348) CompPrice < 152.5 7  5.742 Yes ( 0.14286 0.85714 ) *
##          349) CompPrice > 152.5 5  5.004 No ( 0.80000 0.20000 ) *
##          175) Price > 147 7  0.000 No ( 1.00000 0.00000 ) *
##      11) Advertising > 13.5 45  61.830 Yes ( 0.44444 0.55556 )
##      22) Age < 54.5 25  25.020 Yes ( 0.20000 0.80000 )
##      44) CompPrice < 130.5 14  18.250 Yes ( 0.35714 0.64286 )
##      88) Income < 100 9  12.370 No ( 0.55556 0.44444 ) *
##      89) Income > 100 5  0.000 Yes ( 0.00000 1.00000 ) *
##      45) CompPrice > 130.5 11  0.000 Yes ( 0.00000 1.00000 ) *
##      23) Age > 54.5 20  22.490 No ( 0.75000 0.25000 )
##      46) CompPrice < 122.5 10  0.000 No ( 1.00000 0.00000 ) *
##      47) CompPrice > 122.5 10  13.860 No ( 0.50000 0.50000 )
##      94) Price < 125 5  0.000 Yes ( 0.00000 1.00000 ) *
##      95) Price > 125 5  0.000 No ( 1.00000 0.00000 ) *
##      3) ShelveLoc: Good 85  90.330 Yes ( 0.22353 0.77647 )
##      6) Price < 135 68  49.260 Yes ( 0.11765 0.88235 )
##      12) US: No 17  22.070 Yes ( 0.35294 0.64706 )
##      24) Price < 109 8  0.000 Yes ( 0.00000 1.00000 ) *
##      25) Price > 109 9  11.460 No ( 0.66667 0.33333 ) *
##      13) US: Yes 51  16.880 Yes ( 0.03922 0.96078 ) *
##      7) Price > 135 17  22.070 No ( 0.64706 0.35294 )
##      14) Income < 46 6  0.000 No ( 1.00000 0.00000 ) *
##      15) Income > 46 11  15.160 Yes ( 0.45455 0.54545 ) *

```

Implementación de un árbol de decisión mediante CART en R: training y test sets

Para evaluar la performance de clasificación en estos datos, tenemos que estimar el test error. Dividimos las observaciones en test y training sets, creamos un árbol en el training set y evaluamos en el test-set. La función `predict()` se puede usar (de la misma forma que el `glm()` o `lm()`). En el caso de un árbol de clasificación el argumento `type="class"` le dice a R que devuelva la categoría predicha. En este caso, vemos que la tasa de predicciones correctas cambia a 71.5 % sobre el test set.

```

set.seed(2)
train <- sample(1:nrow(Carseats), 200)
Carseats.test <- Carseats[-train, ]
High.test <- High[-train]
tree.carseats <- tree(High ~ . - Sales, Carseats, subset = train)
tree.pred <- predict(tree.carseats, Carseats.test, type = "class")
table(tree.pred, High.test)

```

```

##          High.test
## tree.pred No Yes
##          No  86  27
##          Yes 30  57

```

Luego, podemos pensar en “podar” el árbol para obtener mejores resultados. la función `cv.tree()` realiza una validación cruzada para determinar el nivel óptimo de complejidad del árbol. “Cost complexity pruning” (podado basado en costo-complejidad) se usa para seleccionar una secuencia de árboles. Usamos el argumento `FUN = prune` para indicar que queremos que el error de clasificación sea el valor que se use para el proceso

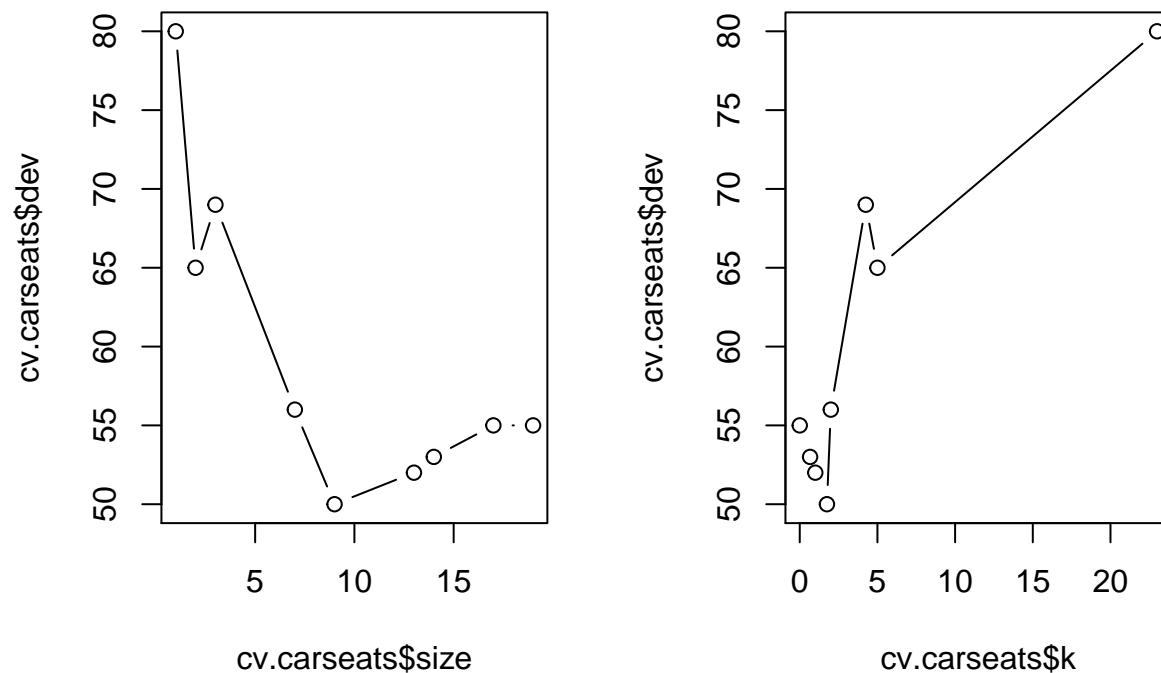
de podado. La función `cv.tree()` usa por defecto el desvío. A su vez, `cv.tree()` reporta la cantidad de nodos terminales en cada árbol considerado (`size`), así como el parámetro de costo-complejidad usado (`k` que se corresponde con lo que llamamos antes α).

```
set.seed(3)
cv.carseats <- cv.tree(tree.carseats, FUN = prune.misclass)
names(cv.carseats)
```

```
## [1] "size" "dev" "k" "method"
```

Notar que, a pesar del nombre `dev` corresponde a la tasa de error “cros-validada”. El árbol con 9 nodos terminales es el que tiene el menor error de clasificación “cros-validado” (50). Graficamos la tasa de error como una función del tamaño del árbol y de `k`.

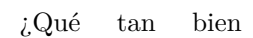
```
par(mfrow = c(1, 2))
plot(cv.carseats$size, cv.carseats$dev, type = "b")
plot(cv.carseats$k, cv.carseats$dev, type = "b")
```



Ahora,

podemos aplicar `prune.misclass()` para podar el árbol y quedarnos con el de 9 nodos.

```
prune.carseats <- prune.misclass(tree.carseats, best = 9)
plot(prune.carseats)
text(prune.carseats, pretty = 0)
```



```
tree.pred <- predict(prune.carseats, Carseats.test, type = "class")
table(tree.pred, High.test)
```

Ahora, el 77% de las observaciones están bien clasificadas. Es decir, que este proceso de cost-complexity pruning generó no solamente un árbol más interpretable sino que además mejoró la performance predictiva. Si incrementamos el valor de `best`, obtenemos un árbol más grande y con peor performance de clasificación.

```

graph TD
    Root[ShelveLoc: Bad, Medium] -->|Price < 142| Left[ShelveLoc: Bad]
    Root -->|Price > 142.5| Right[Population < 278]
    
    Left -->|Income < 100| LeftNo[No]
    Left -->|Income > 100| LeftYes[Yes]
    
    LeftYes -->|Price < 86.5| LeftYesNo[No]
    LeftYes -->|Price > 86.5| LeftYesYes[Yes]
    
    LeftYesYes -->|Advertising < 6.5| LeftYesYesNo[No]
    LeftYesYes -->|Advertising > 6.5| LeftYesYesYes[Yes]
    
    LeftYesYesYes -->|Advertising < 1.5| LeftYesYesYesNo[No]
    LeftYesYesYes -->|Advertising > 1.5| LeftYesYesYesYes[Yes]
    
    LeftYesYesYesYes -->|Age < 33.5| LeftYesYesYesYesNo[No]
    LeftYesYesYesYes -->|Age > 33.5| LeftYesYesYesYesYes[Yes]
    
    LeftYesYesYesYesYes -->|Price < 108.5| LeftYesYesYesYesYesNo[No]
    LeftYesYesYesYesYes -->|Price > 108.5| LeftYesYesYesYesYesYes[Yes]
    
    LeftYesYesYesYesYesYes -->|Price < 115.5| LeftYesYesYesYesYesYesNo[No]
    LeftYesYesYesYesYesYes -->|Price > 115.5| LeftYesYesYesYesYesYesYes[Yes]
    
    LeftYesYesYesYesYesYesYes -->|CompPrice < 118.5| LeftYesYesYesYesYesYesYesNo[No]
    LeftYesYesYesYesYesYesYes -->|CompPrice > 118.5| LeftYesYesYesYesYesYesYesYes[Yes]
    
    Right -->|Population < 278| RightYes[Yes]
    Right -->|Population > 278| RightNo[No]
    
    RightYes -->|Advertising < 10.5| RightYesYes[Yes]
    RightYes -->|Advertising > 10.5| RightYesNo[No]
    
    RightYesYes -->|Price < 99.5| RightYesYesYes[Yes]
    RightYesYes -->|Price > 99.5| RightYesYesNo[No]
  
```

5

```
##           High.test
## tree.pred No Yes
##       No  86  22
##       Yes  30  62
```