Seleccion de modelos Pt 4

Fernando Anorve

4/1/2020

Más ejemplos de Ridge y Lasso

Recordando lo que vimos en la ayudantía pasada, existe otra familia de técnicas para selección de modelos, llamados de regulación, que restringen las magnitudes de los coeficientes estimados $\hat{\beta}_j$, con el fin de que resulte en un modelo más simple y su varianza se reduzca. Ambos dependen de un término de penalización asociado a un valor $0 < \lambda < \infty$ por determinar.

$$\hat{\beta}_{ridge} = \text{argmin}_\beta \left\{ \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}$$

$$\hat{\beta}_{lasso} = \text{argmin}_\beta \left\{ \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}$$

Nuestro conjunto de datos

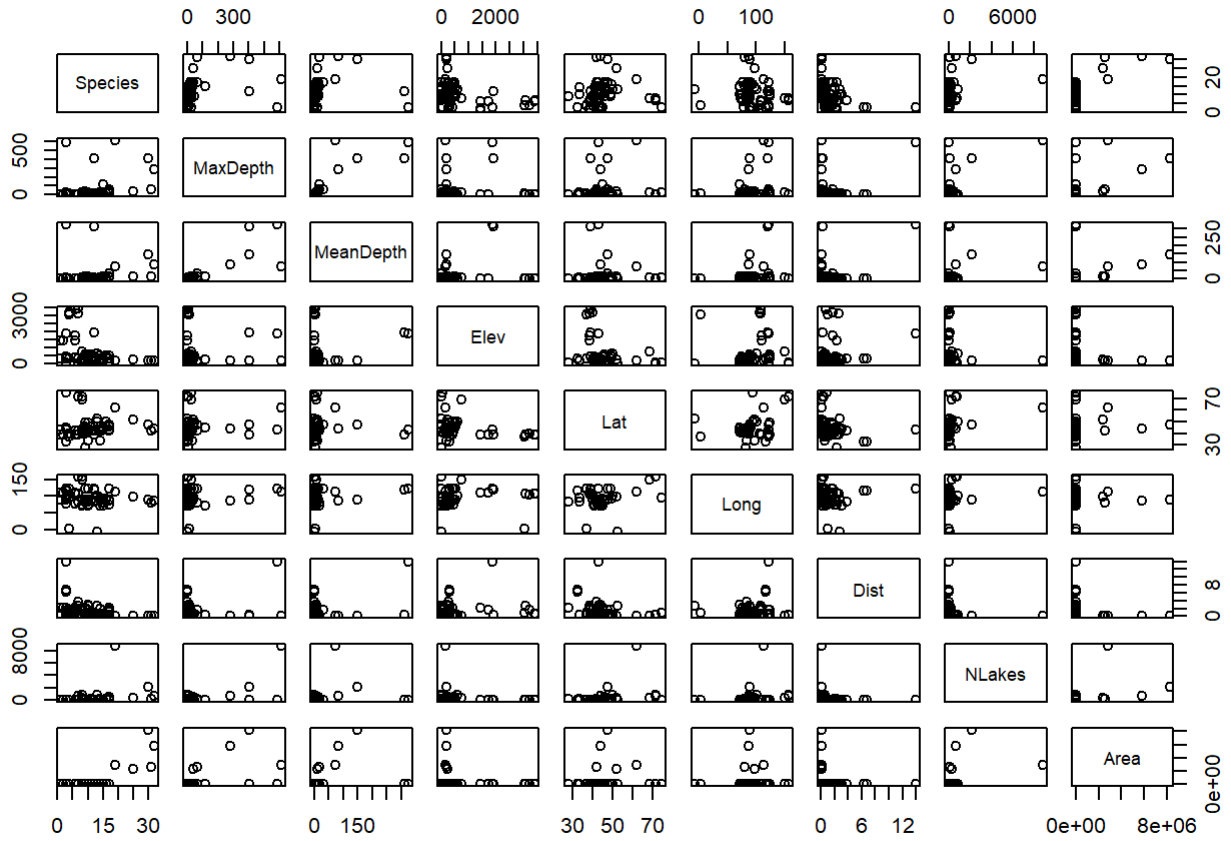
Para este nuevo ejemplo, veremos el conjunto *lakes*, en la librería **alr4**. Estos datos reflejan el número conocido de especies de zooplancton crustáceo en 69 lagos en el mundo. Además, se incluyen algunas características de cada lago.

```
data("lakes")

summary(lakes)
```

```
##     Species          MaxDepth        MeanDepth           Cond       
##  Min.   : 1.00   Min.   :  0.10   Min.   :  0.04   Min.   :   8.00  
##  1st Qu.: 6.00   1st Qu.:  1.50   1st Qu.:  1.00   1st Qu.:  39.75  
##  Median : 9.00   Median :  7.50   Median :  3.40   Median : 167.50  
##  Mean   :10.55   Mean   : 45.45   Mean   : 18.39   Mean   : 258.09  
##  3rd Qu.:14.00   3rd Qu.: 20.40   3rd Qu.: 10.00   3rd Qu.: 314.50  
##  Max.   :32.00   Max.   :613.00   Max.   :325.00   Max.   :1600.00  
##                                                    NA's   :19       
##       Elev             Lat             Long             Dist        
##  Min.   :  -1.0   Min.   :28.00   Min.   : -5.70   Min.   : 0.120  
##  1st Qu.: 187.0   1st Qu.:40.00   1st Qu.: 83.60   1st Qu.: 0.250  
##  Median : 295.0   Median :43.00   Median : 89.60   Median : 0.750  
##  Mean   : 642.2   Mean   :45.17   Mean   : 96.15   Mean   : 1.341  
##  3rd Qu.: 506.0   3rd Qu.:46.20   3rd Qu.:109.60   3rd Qu.: 1.750  
##  Max.   :3433.0   Max.   :74.70   Max.   :156.70   Max.   :14.000  
##                                                                    
##      NLakes           Photo             Area        
##  Min.   :   2.0   Min.   :   1.0   Min.   :      0  
##  1st Qu.:  19.0   1st Qu.:  22.5   1st Qu.:      0  
##  Median :  44.0   Median : 263.0   Median :      8  
##  Mean   : 287.6   Mean   : 370.7   Mean   : 318925  
##  3rd Qu.: 169.0   3rd Qu.: 617.5   3rd Qu.:    149  
##  Max.   :8805.0   Max.   :1500.0   Max.   :8240000  
##                   NA's   :22
```

Vemos que hay una cantidad considerable de datos faltantes en “Cond” y “Photo”, por lo que podemos prescindir de ellos por ahora y ver cómo se comporta el resto de las variables.

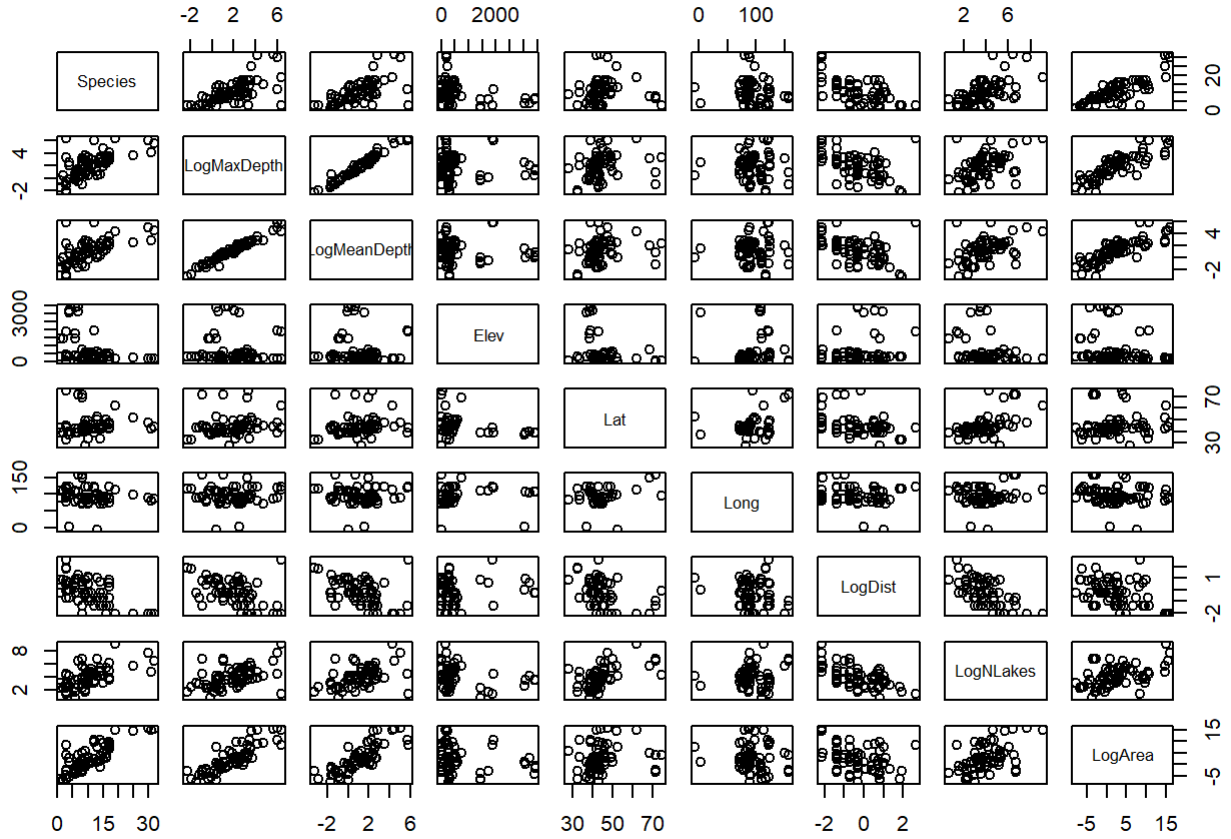
```
lakes$Cond = NULL
lakes$Photo = NULL

lakes = na.omit(lakes)

pairs(lakes)
```



Vemos que hay algunas variables bastante sesgadas hacia cero (MaxDepth, MeanDepth, Area, NLakes, Dist).

Algunas veces, en estos casos puede funcionar convertir estas variables a escala log. **Ojo:** No siempre conviene ni siempre es posible. No hay que malacostumbrarse…

```
lakes$MaxDepth = log(lakes$MaxDepth)
lakes$MeanDepth = log(lakes$MeanDepth)
lakes$Area = log(lakes$Area)
lakes$NLakes = log(lakes$NLakes)
lakes$Dist = log(lakes$Dist)

names(lakes)[c(2,3,7,8,9)] = c("LogMaxDepth","LogMeanDepth","LogDist","LogNLakes","LogArea")

pairs(lakes)
```



¡En esta ocasión parece funcionar bastante bien para estas variables! Lo que parece una buena noticia.

- Para las otras variables, ¿R sugiere alguna transformación?

```
summary(powerTransform(cbind(LogMaxDepth, LogMeanDepth, Elev, Lat,Long, LogDist, LogNLakes,LogAr
ea)~1,lakes,family="yjPower"))
```

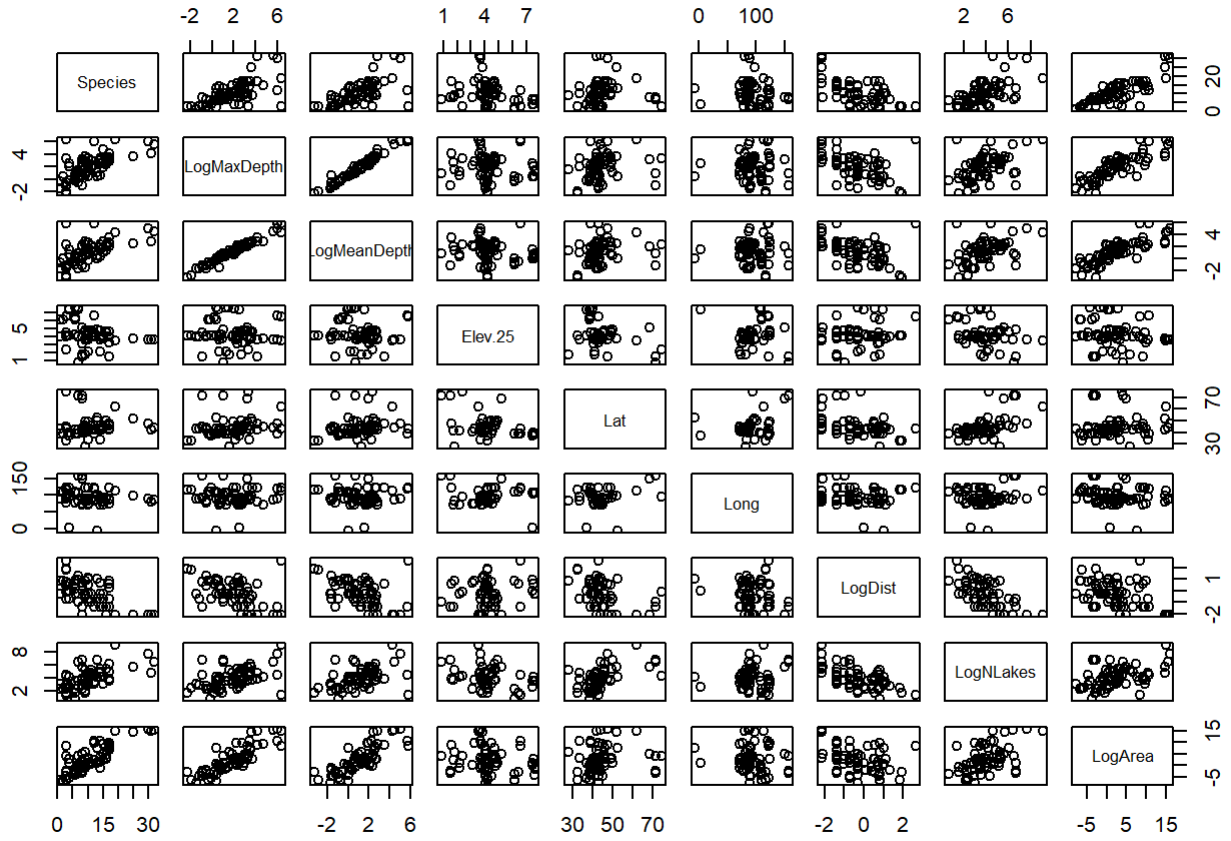
```
## yjPower Transformations to Multinormality
##              Est Power Rounded Pwr Wald Lwr Bnd Wald Upr Bnd
## LogMaxDepth     0.9130        1.00       0.7782       1.0479
## LogMeanDepth    0.9836        1.00       0.8548       1.1123
## Elev            0.1486        0.15       0.0466       0.2506
## Lat            -0.3712        0.00      -1.2207       0.4782
## Long            1.3436        1.34       1.0061       1.6811
## LogDist         0.7612        1.00       0.4543       1.0682
## LogNLakes       0.6228        1.00       0.1256       1.1201
## LogArea         0.8447        0.84       0.7579       0.9315
##
##  Likelihood ratio test that all transformation parameters are equal to 0
##                                          LRT df       pval
## LR test, lambda = (0 0 0 0 0 0 0 0) 725.5169  8 < 2.22e-16
```

Más buenas noticias: La función powerTransform no nos sugiere transformaciones para la mayoría de las variables que transformamos. Para LogArea sugiere una transformación de 0.84. Esto no es muy intuitivo, y el intervalo (0.7579, 0.9315) tampoco contiene potencias muy intuitivas. ¿Cómo explicar una transformación a la

0.84?

- Para Lat se sugiere una transformación logarítmica, pero yo no haría tanto caso en tanto que regresando a la matriz de gráficas, no noto la necesidad de transformar esta variable. **Recuerden:** sólo son sugerencias
- En la gráfica sí percibo cierto sesgo de “Elev”. Se nos sugiere una transformación de 0.15, que no es muy intuitiva. Entre ambos límites del intervalo se encuentra también 0.25, por lo que preferiría llevar a cabo esta transformación

```
lakes$Elev = lakes$Elev^0.25

names(lakes)[4] = c("Elev.25")

pairs(lakes)
```



Podemos concluir en este punto el análisis exploratorio, utilizando las transformaciones antes expuestas.

Antes de continuar, echémosle un vistazo al modelo lineal completo. Del resumen de los parámetros, no aparecen ni LogMaxDepth ni LogMeanDepth como significativas, aunque la gráfica anterior sí muestra cierta correlación((positiva) entre especies y ambas variables.

¿Alguna idea de por qué ocurre esto?

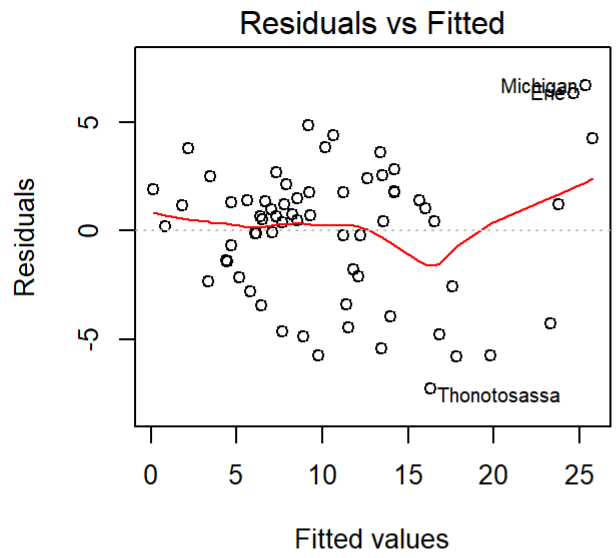
```
lm.usual = lm(Species ~ . , data = lakes)

summary(lm.usual)
```

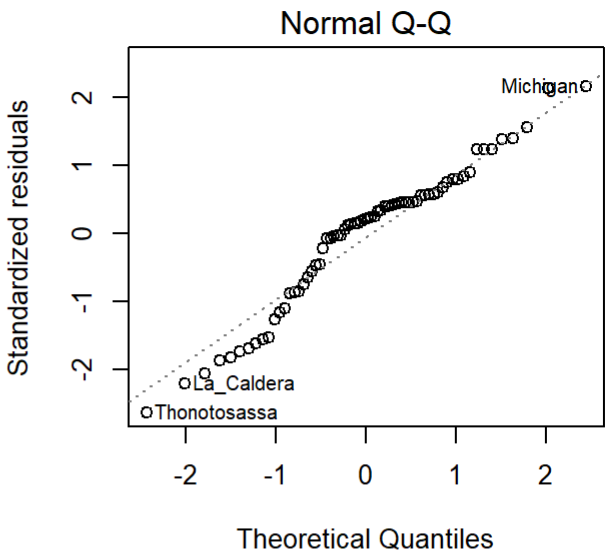
```
##
## Call:
## lm(formula = Species ~ ., data = lakes)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.2685 -2.1044  0.6483  1.7587  6.6770
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  16.16622    3.09720   5.220 2.44e-06 ***
## LogMaxDepth  -0.37132    1.19821  -0.310  0.75773
## LogMeanDepth -0.23652    1.09415  -0.216  0.82961
## Elev.25      -0.35264    0.32060  -1.100  0.27583
## Lat          -0.13717    0.06671  -2.056  0.04421 *
## Long         -0.02564    0.02054  -1.248  0.21692
## LogDist      -1.66018    0.50825  -3.266  0.00182 **
## LogNLakes     0.73091    0.39489   1.851  0.06919 .
## LogArea       0.87463    0.16784   5.211 2.51e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.266 on 59 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.7875, Adjusted R-squared:  0.7587
## F-statistic: 27.34 on 8 and 59 DF,  p-value: < 2.2e-16
```

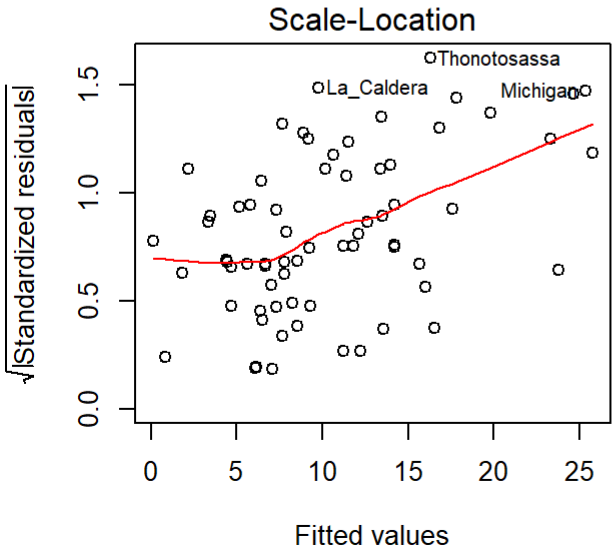
Recordemos que el resumen de coeficientes lo que nos dice es lo que pasa si quitamos una variable y dejamos todas las demás. Como LogMaxDepth y LogMeanDepth están fuertemente correlacionadas, cuando quitamos una, la otra aporta casi la misma información.

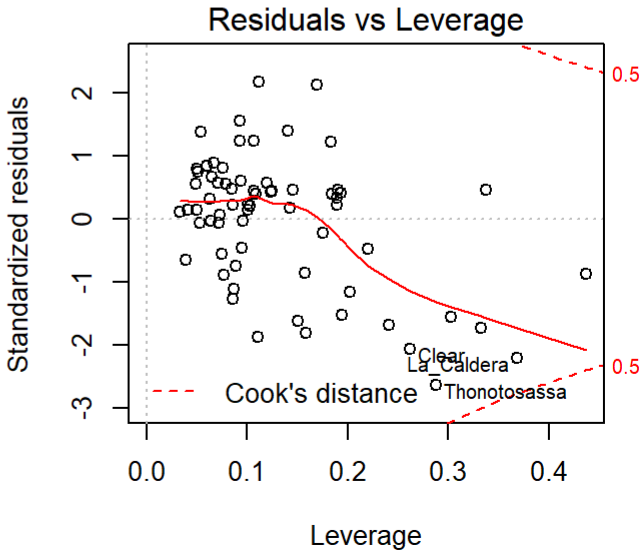
- Al agregar LogMaxDepth cuando LogMeanDepth ya está, no hay información nueva que aportar al modelo. Por lo tanto, aparenta no ser significativa en el resumen.
- Los gráficos de diagnóstico aparecen a continuación:

```
par(mfrow = c(2,2))
plot(lm.usual)
```









- Todo parece estar en orden salvo el tercer gráfico, donde hay cierta tendencia al lado derecho de la gráfica. Sin embargo, no preocuparía tanto porque en general hay pocas observaciones ahí.

Entrenamiento y validación

Hay que elegir un subconjunto de entrenamiento. Recordemos que glmnet necesita de una matriz en vez de los dataframes usuales.

```
lakes = na.omit(lakes)

library(glmnet)

X = model.matrix( Species ~ . ,data = lakes)[,-1]
y = lakes$Species

n = length(y)

set.seed(1024)
train = sample(n,round(2*n/3))

X_train = X[train,]
y_train = y[train]
```

Ridge

$$\hat{\beta}_{ridge} = \text{argmin}_{\beta} \left\{ \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}$$

Hallar λ

1. Suficientemente grande para obtener un modelo más simple que el original
2. Suficientemente pequeña para obtener algo más explicativo que el modelo naíf

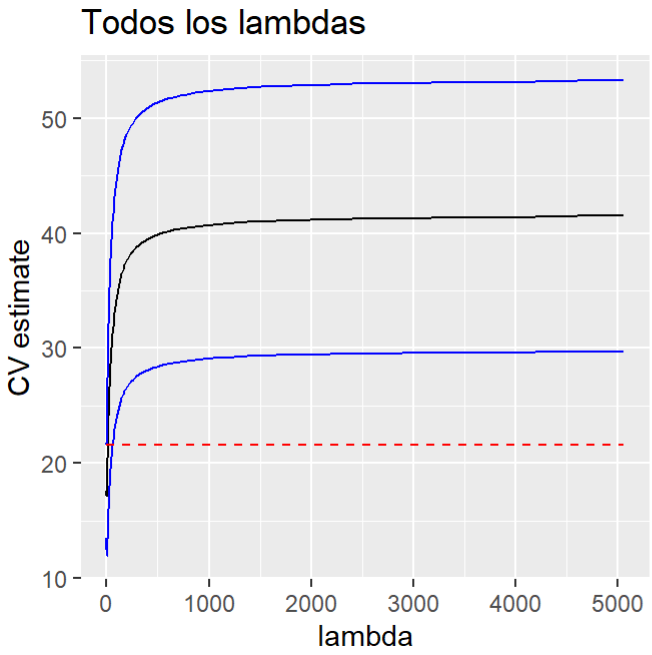
```
cvfit = cv.glmnet(X_train, y_train,nfolds = 10, alpha = 0)
```

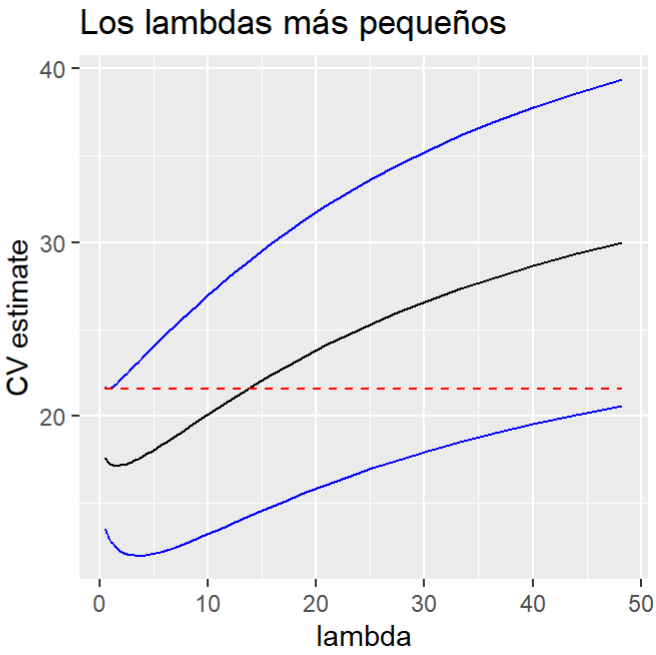
Hay que escoger λ

- Echemos un vistazo a las estimaciones error de validación cruzada

Pregunta: ¿Cuál sería el valor de acuerdo con lambda.min? ¿lambda.1se?

```
normal_plt = ggplot(data = NULL, aes(x = cvfit$lambda , y = cvfit$cvm)) +
  geom_line() + xlab("lambda") + ylab("CV estimate") + ggtitle("Todos los lambdas") +
  geom_line(aes(x = cvfit$lambda , y = cvfit$cvup), color = "blue") +
  geom_line(aes(x = cvfit$lambda , y = cvfit$cvlo), color = "blue") +
  geom_line(aes(x = cvfit$lambda , y = min(cvfit$cvup)), color = "red", linetype = 2)

zoom_plt = ggplot(data = NULL, aes(x = tail(cvfit$lambda, 50) , y = tail(cvfit$cvm ,50))) +
  geom_line() + xlab("lambda") + ylab("CV estimate") + ggtitle("Los lambdas más pequeños") +
  geom_line(aes(x = tail(cvfit$lambda, 50) , y = tail(cvfit$cvup ,50)), color = "blue") +
  geom_line(aes(x = tail(cvfit$lambda, 50) , y = tail(cvfit$cvlo ,50)), color = "blue") +
  geom_line(aes(x = tail(cvfit$lambda, 50) , y = min(cvfit$cvup)), color = "red", linetype = 2)

grid.arrange(normal_plt,zoom_plt,ncol = 2)
```





Los lambdas óptimos según los dos criterios son:

```
cvfit$lambda.min
```

```
## [1] 1.544517
```

```
cvfit$lambda.1se
```

```
## [1] 14.40421
```

¿Qué coeficientes resultan de ambos criterios?

Mínimo lambda

```
coef(cvfit, s = "lambda.min")
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept)  14.61503390
## LogMaxDepth   0.38173855
## LogMeanDepth  0.09777015
## Elev.25      -0.59808123
## Lat          -0.05592407
## Long         -0.03405718
## LogDist      -1.51100513
## LogNLakes     0.52892819
## LogArea       0.50853126
```

Lambda 1se

```
coef(cvfit, s = "lambda.1se")
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept)  10.531056388
## LogMaxDepth   0.355418097
## LogMeanDepth  0.312391509
## Elev.25      -0.281794873
## Lat          -0.009809614
## Long         -0.012929092
## LogDist      -0.691160782
## LogNLakes     0.382080766
## LogArea       0.207678912
```

Error in-sample

```
yhat=predict(cvfit,X_train,s="lambda.1se")
sqrt(mean((y_train-yhat)^2))
```

```
## [1] 4.244243
```

Se espera que lambda.min dé un menor error in-sample por ser un modelo más complejo

```
yhat2=predict(cvfit,X_train,s="lambda.min")
sqrt(mean((y_train-yhat2)^2))
```

```
## [1] 3.117574
```

Conjunto de validación:

Tomamos el resto de las observaciones como conjunto de validación

```
X_val = X[-train,]
y_val = y[-train]
```

Y ahora revisamos el error medio del conjunto de validación para ambos criterios

```
yhat=predict(cvfit,X_val,s="lambda.1se")
sqrt(mean((y_val-yhat)^2))
```

```
## [1] 4.358465
```

```
yhat2=predict(cvfit,X_val,s="lambda.min")
sqrt(mean((y_val-yhat2)^2))
```

```
## [1] 3.675025
```

En este caso, lambda.min devuelve una mejor capacidad de predicción.

Lasso

$$\hat{\beta}_{lasso} = \text{argmin}_{\beta} \left\{ \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}$$

[De nuevo,] hallar λ

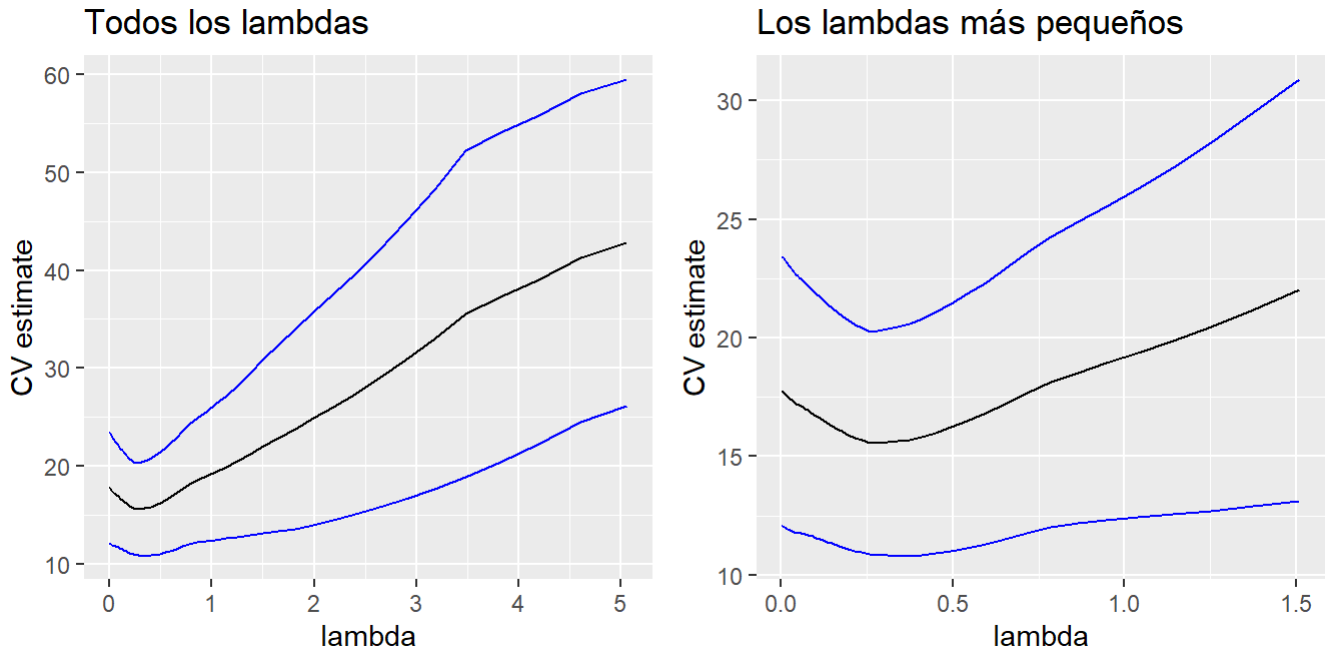
1. Suficientemente grande para obtener un modelo más simple que el original
2. Suficientemente pequeña para obtener un modelo más explicativo que el modelo naíf

```
cvfit = cv.glmnet(X_train, y_train,nfolds = 10, alpha = 1)
```

¿Cómo escogemos λ?

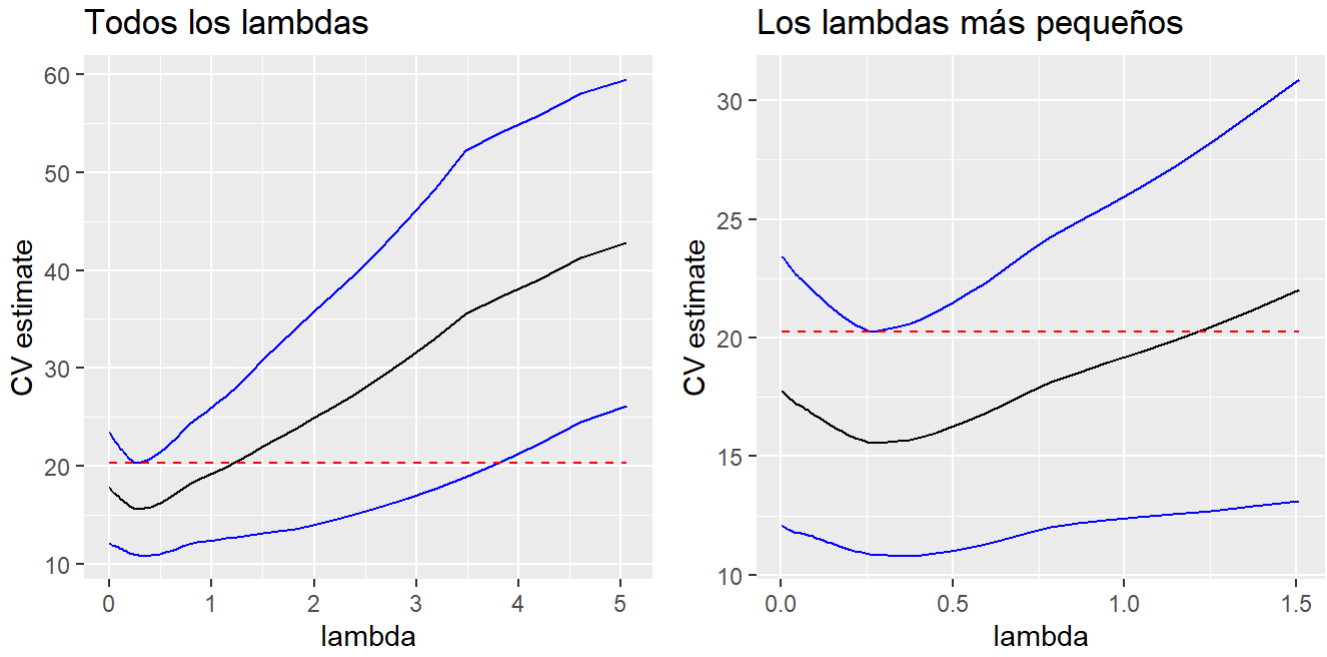
Pregunta: ¿Cuál sería el valor de acuerdo con lambda.min? ¿lambda.1se?

```
normal_plt = ggplot(data = NULL, aes(x = cvfit$lambda , y = cvfit$cvm)) +
  geom_line() + xlab("lambda") + ylab("CV estimate") + ggtitle("Todos los lambdas") +
  geom_line(aes(x = cvfit$lambda , y = cvfit$cvup), color = "blue") +
  geom_line(aes(x = cvfit$lambda , y = cvfit$cvlo), color = "blue")

zoom_plt = ggplot(data = NULL, aes(x = tail(cvfit$lambda, 67) , y = tail(cvfit$cvm ,67))) +
  geom_line() + xlab("lambda") + ylab("CV estimate") + ggtitle("Los lambdas más pequeños") +
  geom_line(aes(x = tail(cvfit$lambda, 67) , y = tail(cvfit$cvup ,67)), color = "blue") +
  geom_line(aes(x = tail(cvfit$lambda, 67) , y = tail(cvfit$cvlo ,67)), color = "blue")

grid.arrange(normal_plt,zoom_plt,ncol = 2)
```



Veamos la gráfica con la línea punteada:

```
normal_plt = ggplot(data = NULL, aes(x = cvfit$lambda , y = cvfit$cvm)) +
  geom_line() + xlab("lambda") + ylab("CV estimate") + ggtitle("Todos los lambdas") +
  geom_line(aes(x = cvfit$lambda , y = cvfit$cvup), color = "blue") +
  geom_line(aes(x = cvfit$lambda , y = cvfit$cvlo), color = "blue") +
  geom_line(aes(x = cvfit$lambda , y = min(cvfit$cvup)), color = "red", linetype = 2)

zoom_plt = ggplot(data = NULL, aes(x = tail(cvfit$lambda, 67) , y = tail(cvfit$cvm ,67))) +
  geom_line() + xlab("lambda") + ylab("CV estimate") + ggtitle("Los lambdas más pequeños") +
  geom_line(aes(x = tail(cvfit$lambda, 67) , y = tail(cvfit$cvup ,67)), color = "blue") +
  geom_line(aes(x = tail(cvfit$lambda, 67) , y = tail(cvfit$cvlo ,67)), color = "blue") +
  geom_line(aes(x = tail(cvfit$lambda, 67) , y = min(cvfit$cvup)), color = "red", linetype = 2)

grid.arrange(normal_plt,zoom_plt,ncol = 2)
```



lamda.min sería aproximadamente 0.25 y lambda.1se sería alrededor de 1.25

```
cvfit$lambda.min
```

```
## [1] 0.2827606
```

```
cvfit$lambda.1se
```

```
## [1] 1.141509
```

¿Qué coeficientes resultan de ambos criterios?

Mínimo lambda

```
coef(cvfit, s = "lambda.min")
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept)  13.68547628
## LogMaxDepth   .
## LogMeanDepth  .
## Elev.25      -0.43037078
## Lat           .
## Long         -0.03857947
## LogDist      -1.64069642
## LogNLakes     0.09561964
## LogArea       0.74665134
```

Lambda 1se

```
coef(cvfit, s = "lambda.1se")
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept)   8.7970322
## LogMaxDepth   .
## LogMeanDepth  .
## Elev.25       .
## Lat           .
## Long          .
## LogDist      -1.0524268
## LogNLakes     .
## LogArea       0.6651298
```

Como habíamos visto ayer, algunos coeficientes de la regresión son iguales a cero.

Error in-sample

```
yhat=predict(cvfit,X_train,s="lambda.1se")
sqrt(mean((y_train-yhat)^2))
```

```
## [1] 3.64061
```

lambda.min da un menor error in-sample, como se hubiera esperado

```
yhat2=predict(cvfit,X_train,s="lambda.min")
sqrt(mean((y_train-yhat2)^2))
```

```
## [1] 3.019867
```

El error in-sample no nos dice tanto como quisiéramos sobre la capacidad predictiva de un modelo.

Conjunto de validación:

Tomamos el conjunto de validación

```
X_val = X[-train,]
y_val = y[-train]
```

Revisamos el error medio del conjunto de validación para ambos criterios

```
yhat=predict(cvfit,X_val,s="lambda.1se")
sqrt(mean((y_val-yhat)^2))
```

```
## [1] 3.662219
```

```
yhat2=predict(cvfit,X_val,s="lambda.min")
sqrt(mean((y_val-yhat2)^2))
```

```
## [1] 3.709849
```

Notemos que para lambda.1se el error no cambia mucho, mientras que para el lambda.min sí lo hace.

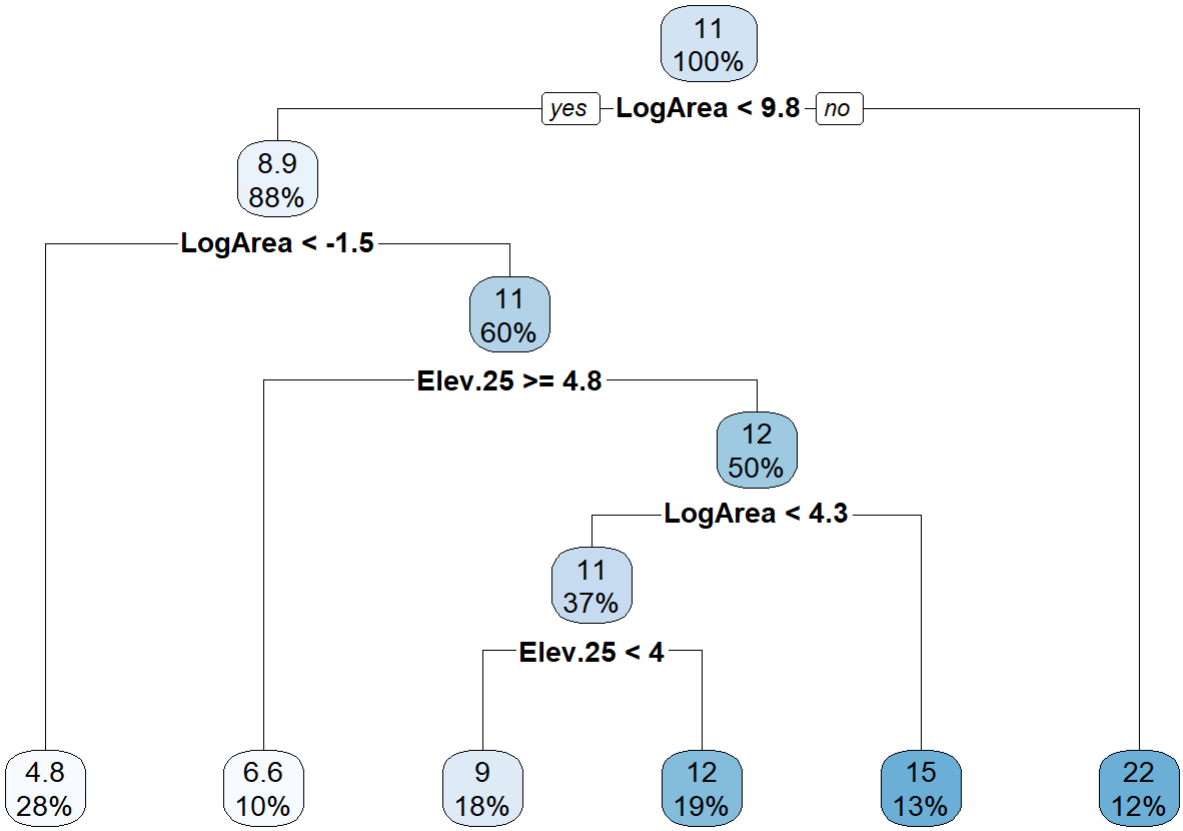
Otra técnica para elegir variables

Escoger un modelo utilizando árboles de decisión, con la función rpart

```
set.seed(7)
library(rpart)
library(rpart.plot)

fit = rpart(Species~.,data = lakes,method="anova",control = rpart.control(cp = 0.0001))

# El resumen no nos dice mucho y no es ilustrativo, por lo que lo omitimos
# summary(fit)
```

El árbol se vería así:

```
rpart.plot(fit)
```



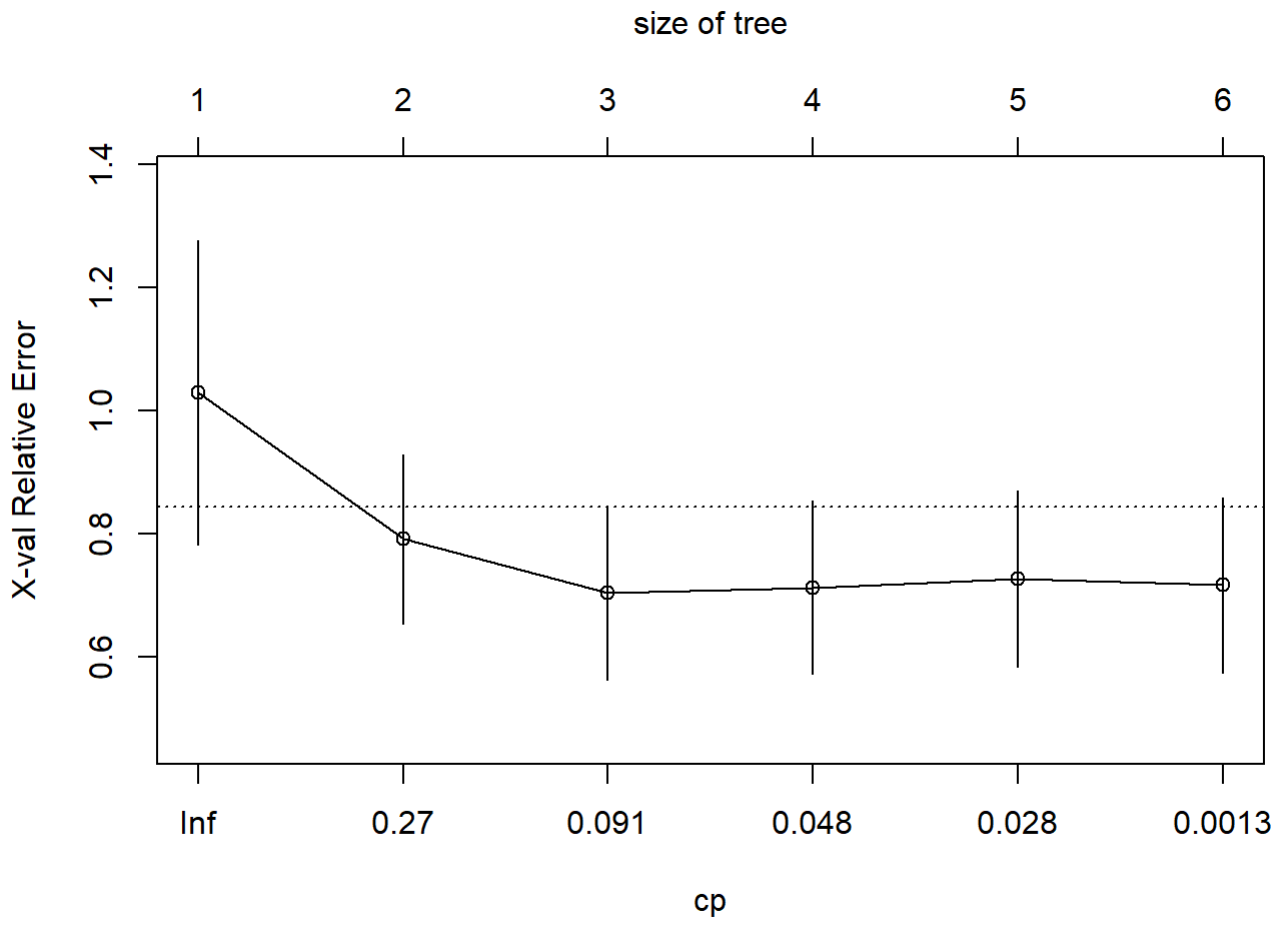
Las variables significativas según este árbol son Elev y LogArea

¿Cómo sabemos si el árbol es más profundo de lo necesario? **CV**

```
printcp(fit)
```

```
##
## Regression tree:
## rpart(formula = Species ~ ., data = lakes, method = "anova",
##     control = rpart.control(cp = 1e-04))
##
## Variables actually used in tree construction:
## [1] Elev.25 LogArea
##
## Root node error: 2963/68 = 43.573
##
## n= 68
##
##         CP nsplit rel error  xerror    xstd
## 1 0.439557      0   1.00000 1.02924 0.24693
## 2 0.159846      1   0.56044 0.79101 0.13775
## 3 0.051648      2   0.40060 0.70317 0.14011
## 4 0.044826      3   0.34895 0.71257 0.14049
## 5 0.017994      4   0.30412 0.72617 0.14297
## 6 0.000100      5   0.28613 0.71635 0.14252
```

```
plotcp(fit)
```

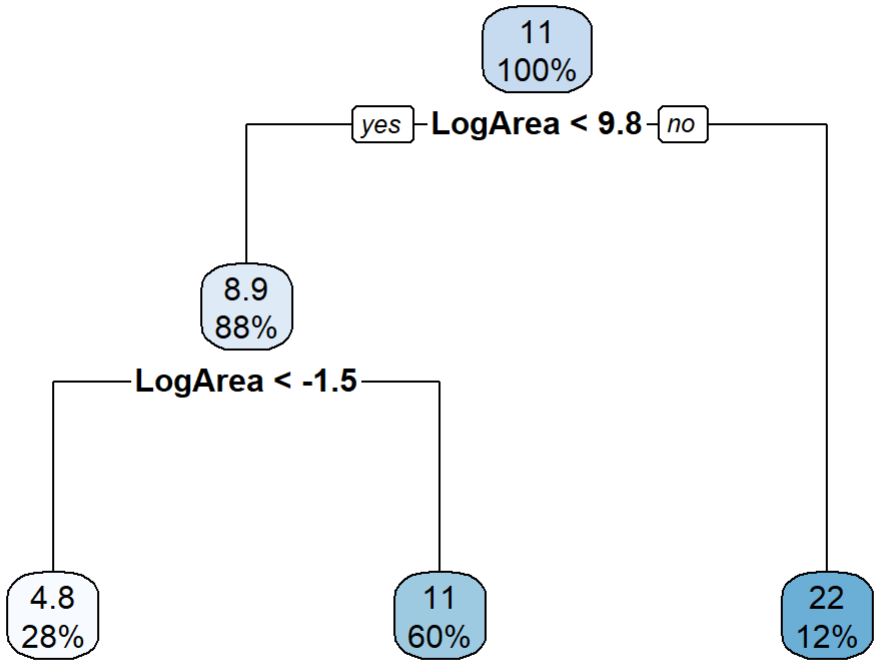


Parece que 0.27 es razonable para podar nuestro árbol, según validación cruzada

```
fit.pruned = prune(fit,cp=.09)
#fit.pruned
#summary(fit.pruned)
```

Ya podado, el árbol se vería así:

```
rpart.plot(fit.pruned)
```



La unica variable significativa según el árbol ya podado es LogArea