El Paquete CARET

Comparación de modelos lineales y basados en árboles

Santander Meteorology Group

Objetivo:

En la presente práctica trataremos de profundizar en la utilización del paquete CARET. Para ello, consideraremos las familias de métodos basados en modelos lineales y los basados en árboles, y el dataset meteo.csv, que podéis descargaros en el GitHub dedicado a este Máster (meteo.csv) y que hemos utilizado en sesiones anteriores. Dicho dataset contiene en la primera columna el valor de precipitación observado en Lisboa en el periodo 1979-2008 mientras que en las restantes contiene los valores observados de diferentes variables atmosféricas en 40 puntos que cubren aproximadamente la Península Ibérica. Dichas variables serán los predictores del modelo mientras que la precipitación será nuestra variable objetivo. En particular los predictores son:

- Altura geopotencial en 500 hPa (columnas 2:41),
- Temperatura del aire en 850 hPa (columnas 42:81), 700 hPa (columnas 82:121) y 500 hPa (columnas 122:161),
- Temperatura del aire en superficie (columnas 162:201),
- Humedad específica del aire en 850 hPa (columnas 202:241) y 500 hPa (columnas 242:281) y
- Presión al nivel del mar (columnas 282:321)

Finalmente, para evaluar los resultados obtenidos con nuestros modelos consideraremos la librería hydroGOF. Nota: El uso de esta librería nos simplificará los cálculos si bien los estadísticos utilizados pueden implementarse de forma explícita o están disponibles en otras librerías.

A continuación cargamos (y/o instalamos) las librerías que utilizaremos a lo largo de la práctica:

```
library(caret) ## Training the model
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
library(fields) ## Plotting
```

```
## Loading required package: spam
## Loading required package: dotCall64
## Loading required package: grid
## Spam version 2.6-0 (2020-12-14) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.
##
## Attaching package: 'spam'
## The following objects are masked from 'package:base':
##
##     backsolve, forwardsolve
## See https://github.com/NCAR/Fields for
##  an extensive vignette, other supplements and source code
```

```
library(hydroGOF) ## Evaluation
```

```
## Loading required package: zoo
##
## Attaching package: 'zoo'
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
library(pROC) ## Classification Evaluation
```

```
## Type 'citation("pROC")' for a citation.

##

## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':

##

##     cov, smooth, var
```

```
library(rpart) ## Tree-based model
```

Preprocesado de los datos:

Carguemos en primer lugar los datos meteorológicos. Notad que predictores y predictando se encuentran en el mismo fichero de modo que separemos ambos por comodidad:

```
# loading data
data <- read.csv("~/Dropbox/M1966_DataMining/datasets/meteo.csv")
df.meteo <- data[, -1]; rm(data)
y <- df.meteo[, 1]  ## Predictando
x <- df.meteo[, -1]  ## Predictores
```

Un problema para algunos modelos deriva de la existencia de variables constantes o con varianza muy cercana a cero, lo cual puede evaluarse del siguiente modo:

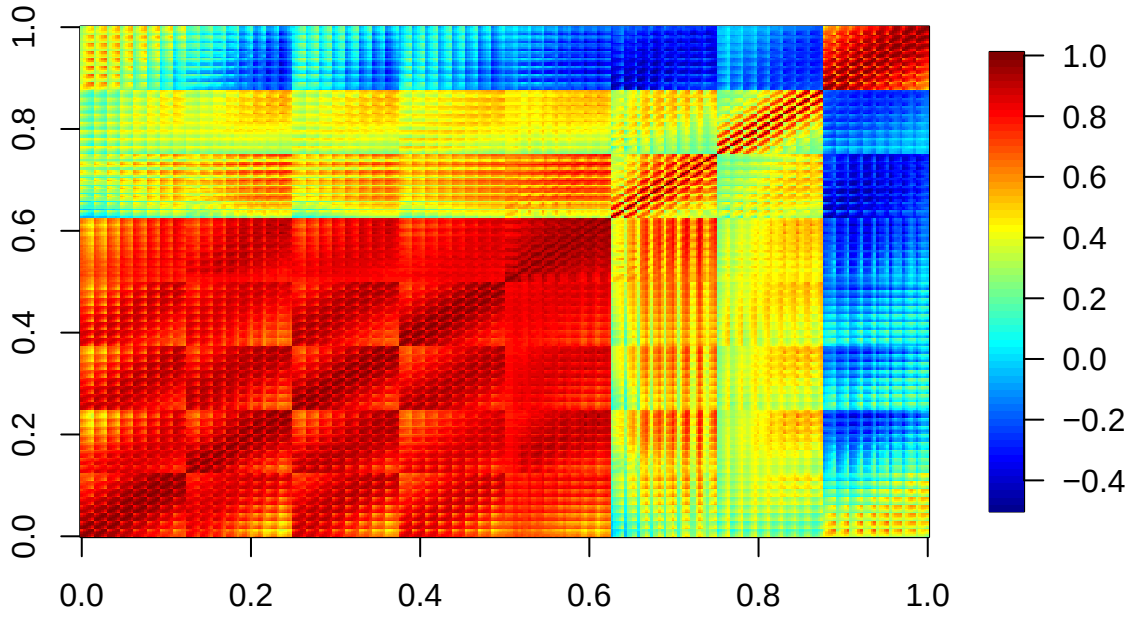
```
nzv <- nearZeroVar(x, saveMetrics= TRUE)
which(nzv$nzv)
```

```
## integer(0)
```

Al tratarse de datos de variables meteorológicas sobre un dominio espacial se pueden dar autocorrelaciones significativas entre los predictores, co-linealidades y rangos de valores muy dispares entre ellos. ¿Afecta alguna de estas características a los modelos basados en árboles? ¿Y a los modelos lineales?

Veamos primero la matriz de correlación cruzada entre los predictores:

```
xCorr <- cor(x)

par(mfrow=c(1,1))

image.plot(xCorr)
```



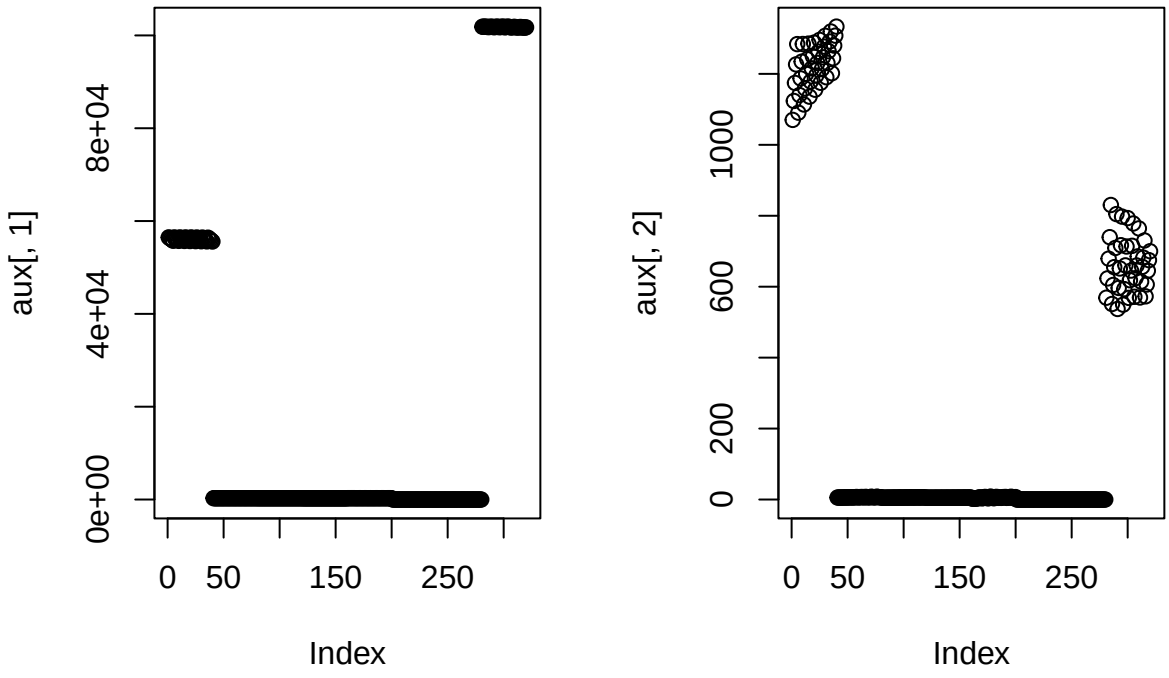
Los valores altos de correlación suelen implicar co-linealidad entre los predictores, lo cual podemos evaluarlo con:

```
comboInfo <- findLinearCombos(x)

comboInfo
```

```
## $linearCombos

## list()

##

## $remove

## NULL
```

Veamos ahora qué ocurre con los rangos de las variables consideradas:

```
aux <- matrix(data = NA, nrow = ncol(x), 2)
aux[,1] <- apply(x, MARGIN = 2, FUN = mean, na.rm = TRUE)
aux[,2] <- apply(x, MARGIN = 2, FUN = sd, na.rm = TRUE)
par(mfrow=c(1,2))
plot(aux[,1])
plot(aux[,2])
```

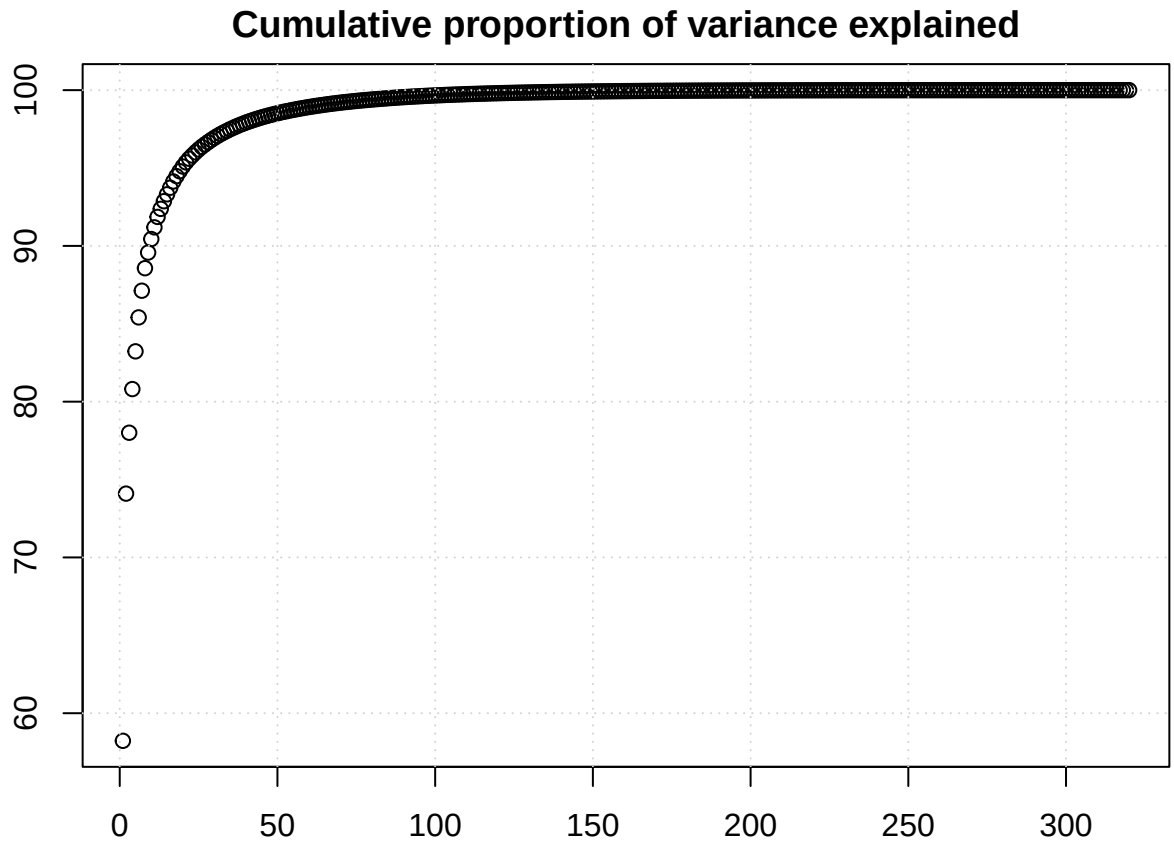


Como vemos, existe una gran autocorrelación entre los diferentes predictores, los cuales además presentan tanto rangos como variabilidades que difieren hasta en 4 órdenes de magnitud.

Para resolver ambos problemas, en caso de ser necesario para nuestros modelos, una técnica habitualmente aplicada es el Análisis de Componentes Principales sobre las variables normalizadas:

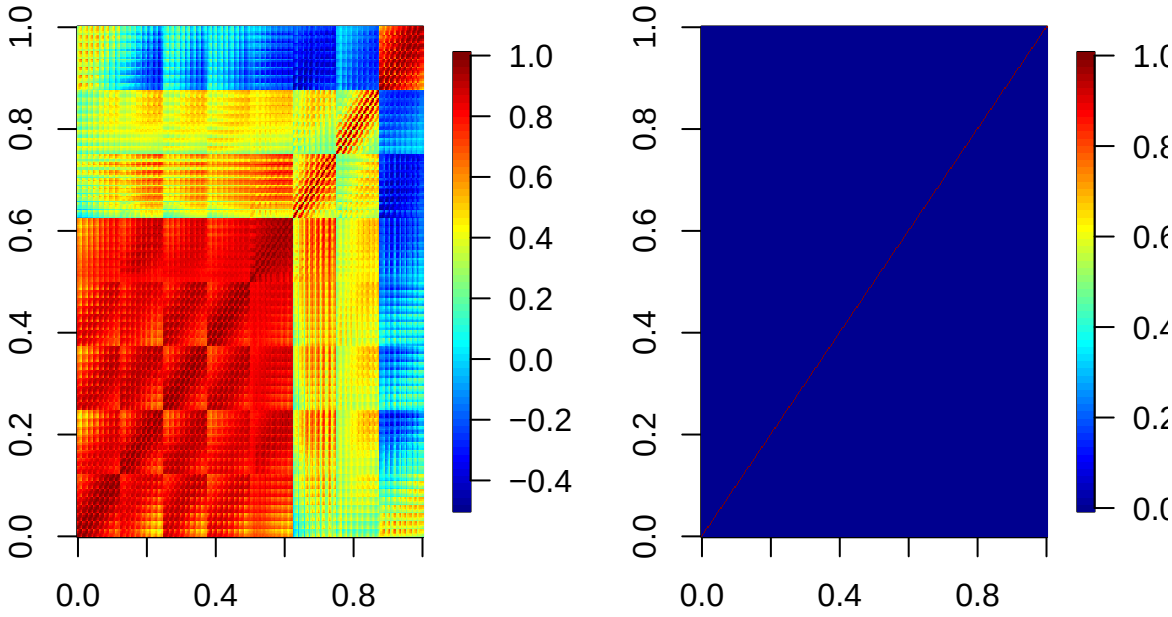
```
# PCA with R
PCA <- prcomp(x, center = TRUE, scale. = TRUE)
```

```
par(mfrow=c(1,1), mar=c(2.1,2.1,2.1,2.1))
plot(cumsum(PCA$sdev^2)/sum(PCA$sdev^2)*100,main="Cumulative proportion of variance explained")
grid()
```



Por un lado, la proporción de varianza explicada nos da un criterio para reducir la dimensión del problema, es decir reducir el número de predictores del modelo. Por otro lado, en el espacio de las PCAs las diferentes dimensiones son independientes:

```
pcaCorr <- cor(PCA$x)
par(mfrow=c(1,2))
image.plot(xCorr)
image.plot(pcaCorr)
```



En adelante consideraremos las 50 primeras PCAs para el entrenamiento y aplicación de los modelos. ¿Con qué varianza explicada se corresponde?

```
print(cumsum(PCA$sdev[1:50]^2)/sum(PCA$sdev^2)*100)
```

```
##  [1] 58.22748 74.09688 78.00651 80.80834 83.23083 85.41027 87.12340 88.56891
##  [9] 89.57316 90.44000 91.19128 91.86198 92.37748 92.86927 93.31144 93.73552
## [17] 94.13585 94.47848 94.79342 95.09655 95.36363 95.59608 95.80465 96.00553
## [25] 96.19090 96.35968 96.51440 96.66293 96.80242 96.93679 97.06009 97.17861
## [33] 97.28529 97.38907 97.48892 97.58256 97.67308 97.75856 97.83745 97.91617
## [41] 97.98793 98.05823 98.12651 98.18961 98.25215 98.30878 98.36299 98.41548
## [49] 98.46684 98.51634
```

En el caso de la variable objetivo debemos tener en cuenta el carácter dual de la precipitación. Por un lado, la ocurrencia de precipitación da lugar a un problema de clasificación mientras que, para los días de precipitación, la aproximación de la cantidad de precipitación da lugar a un prob-

lema de predicción. Consideraremos el umbral de discretización 1 mm que define la ocurrencia de precipitación (Wet days). Puedes consultar más detalles de la definición en la web de ECA&D.

```
occu <- as.numeric(y > 1) ## Ocurrencia de precipitacion
occu[which(y <= 1)] <- 0
occu[which(y > 1)] <- 1
y[1:10]
```

```
##  [1] 10.9  0.6 13.0  0.0  0.0  1.2  1.1  0.0  0.0  0.7
```

```
occu[1:10]
```

```
##  [1] 1 0 1 0 0 1 1 0 0 0
```

Linear- vs Trees-based Models

Una vez hemos definido las variables predictoras y objetivo, pasemos a identificar las familias de modelos a utilizar dentro del paquete CARET. ¿Contiene alguna implementación para los modelos lineales y basados en árboles? En caso afirmativo, ¿se consideran ambos casos: Clasificación y Predicción?

Definamos ahora el esquema de validación:

```
? trainControl
```

Entre las opciones vemos que podemos establecer:

- Método de validación cruzada. ¿Qué método sería adecuado aplicar?
- Preprocesado de los datos. ¿Es necesario algún preprocesado? (`? preProcess`)
- Los parámetros de búsqueda del modelo óptimo, ¿qué parametros podemos ajustar?

En base a lo discutido anteriormente, definamos, por ejemplo, el siguiente flujo de entrenamiento:

```
trainParams <- trainControl(method = "cv", number = 10,
                            preProcOptions = list(pcaComp = 50))
```

Es decir, consideraremos un 10-fold sobre las 50 primeras PCAs. Veamos, a modo de ejemplo, como sería la aplicación sobre un fold:

```r
indTrain <- createDataPartition(y,p=0.9,list=FALSE)
xTrain <- x[indTrain,]
yTrain <- y[indTrain]

xTest <- x[-indTrain,]
yTest <- y[-indTrain]
c(length(y),length(yTrain),length(yTest))
```

```
## [1] 10958  9863  1095
```

Sobre el conjunto de **train** debemos obtener las 50 primeras PCAs:

```r
# PCA with R
PCA <- prcomp(xTrain, center = TRUE, scale. = TRUE)
pcaTrain <- PCA$x[,1:50]
dim(pcaTrain)
```

```
## [1] 9863   50
```

Proyectar las coordenadas del conjunto de **test** en el espacio de las PCAs:

```r
pcaTest <- predict(PCA, newdata = xTest)
dim(pcaTest)
```

```
## [1] 1095  320
```

Y seleccionar de nuevo las 50 primeras coordenadas:

```r
pcaTest <- pcaTest[,1:50]
```

Ahora aprendemos el modelo en el conjunto de **train**:

```r
## Regresion logistica
dfTrain <- data.frame(y = occu[indTrain], x=pcaTrain)
model <- glm(y~., data = dfTrain, family = binomial(link = "logit"))
out <- model$fitted.values
outbin <- as.double(out > 0.5)
```

```r
table(occu[indTrain], outbin)
```

```
##    outbin
##        0    1
##   0 7564  366
##   1  691 1242
```

```r
100*sum(diag(table(occu[indTrain], outbin))) / length(outbin)
```

```
## [1] 89.28318
```

y lo usamos para predecir los valores en el conjunto de `test`:

```r
dfTest <- data.frame(y = occu[-indTrain], x=pcaTest)
out1 <- predict(model, newdata = dfTest, type = "response")
outbin1 <- as.double(out1 > 0.5)
table(occu[-indTrain], outbin1)
```

```
##    outbin1
##       0   1
##   0 820  48
##   1  93 134
```

```r
100*sum(diag(table(occu[-indTrain], outbin1))) / length(outbin1)
```

```
## [1] 87.12329
```

Podemos hacer una evaluación más profusa con la función `confusionMatrix`:

- Train:

```r
## train
confusionMatrix(as.factor(occu[indTrain]),as.factor(outbin))
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction    0    1
##          0 7564  366
##          1  691 1242
##
##                Accuracy : 0.8928
##                  95% CI : (0.8866, 0.8989)
##     No Information Rate : 0.837
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6369
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9163
##             Specificity : 0.7724
##          Pos Pred Value : 0.9538
##          Neg Pred Value : 0.6425
##              Prevalence : 0.8370
##          Detection Rate : 0.7669
##    Detection Prevalence : 0.8040
##       Balanced Accuracy : 0.8443
##
##        'Positive' Class : 0
##
```

- Test:

```
## test
confusionMatrix(as.factor(occu[-indTrain]),as.factor(outbin1))
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction   0   1
##          0 820  48
##          1  93 134
##
##                Accuracy : 0.8712
##                  95% CI : (0.8499, 0.8905)
##     No Information Rate : 0.8338
##     P-Value [Acc > NIR] : 0.0003532
##
##                   Kappa : 0.5773
##
##  Mcnemar's Test P-Value : 0.0002110
##
##             Sensitivity : 0.8981
##             Specificity : 0.7363
##          Pos Pred Value : 0.9447
##          Neg Pred Value : 0.5903
##              Prevalence : 0.8338
##          Detection Rate : 0.7489
##    Detection Prevalence : 0.7927
##       Balanced Accuracy : 0.8172
##
##        'Positive' Class : 0
##
```

Clasificación:

Resolveremos en primer lugar el problema de clasificación asociado a la predicción de lluvia/no lluvia. Para ello consideraremos la regresión logística y los árboles de clasificación como representantes de ambas aproximaciones.

Realizaremos toda la fase de entrenamiento usando CARET y partiendo del flujo definido anteriormente, con las modificaciones pertinentes asociadas a ambos métodos.

Regresión Logística:

En virtud de lo visto en el ejemplo anterior sobre la regresión logística, ¿hay algún parámetro que debamos incluir en el ajuste del modelo? En base a ello, definiremos el flujo de entrenamiento partiendo del ya definido:

```
trainParams <- trainControl(method = "cv", number = 10,
                            preProcOptions = list(pcaComp = 50, thres = 0.95))
```

Definido el tipo de entrenamiento, ajustemos el modelo lineal:

```
dfTrain <- data.frame(y = as.factor(occu), x=x)
modelFit <- train(y ~ ., data=dfTrain, method="glm",
                  trControl=trainParams, preProcess = "pca")
## str(modelFit)
```

Evaluamos ahora el modelo respecto a toda la muestra:

```
yPred <- predict(modelFit,newdata = dfTrain)
confusionMatrix(as.factor(occu),yPred)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 8377  421
##          1  782 1378
##
##                Accuracy : 0.8902
##                  95% CI : (0.8842, 0.896)
##     No Information Rate : 0.8358
##     P-Value [Acc > NIR] : < 2.2e-16
```

```
##
##                  Kappa : 0.6298
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9146
##             Specificity : 0.7660
##          Pos Pred Value : 0.9521
##          Neg Pred Value : 0.6380
##              Prevalence : 0.8358
##          Detection Rate : 0.7645
##    Detection Prevalence : 0.8029
##       Balanced Accuracy : 0.8403
##
##        'Positive' Class : 0
##
```

Para algunos estadísticos de evaluación es necesario considerar las predicciones probabilísticas en lugar de las categóricas. Un ejemplo es la construcción de la curva ROC y la obtención a partir de ella del coeficiente AUC (Area Under the Curve):

```
yPred <- predict(modelFit,newdata = dfTrain, type="prob")
roc_obj <- roc(as.factor(occu), yPred[["1"]])
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
auc(roc_obj)
```

```
## Area under the curve: 0.9325
```

Árboles de Clasificación:

Una vez resuelta la aproximación lineal, apliquemos un proceso similar para los árboles de clasificación. Notad que en esta ocasión el modelo en CARET viene definido por el método rpart. ¿En este caso debemos incluir la optimización de algún parámetro del árbol (`? rpart.control`)?

```
trainParams <- trainControl(method = "cv", number = 10,
                            preProcOptions = list(pcaComp = 50, thres = 0.95))
dfTrain <- data.frame(y = as.factor(occu), x=x)
treeGrid <- expand.grid(cp = 10^c(-1:-5))
treeFit <- train(y ~ ., data=dfTrain, method="rpart", trControl=trainParams,
                 tuneGrid = treeGrid, preProcess = "pca")
str(treeFit)
```

```
## List of 24
##  $ method      : chr "rpart"
##  $ modelInfo   :List of 15
##   ..$ label     : chr "CART"
##   ..$ library   : chr "rpart"
##   ..$ type      : chr [1:2] "Regression" "Classification"
##   ..$ parameters:'data.frame':   1 obs. of  3 variables:
##   .. ..$ parameter: chr "cp"
##   .. ..$ class    : chr "numeric"
##   .. ..$ label    : chr "Complexity Parameter"
##   ..$ grid      :function (x, y, len = NULL, search = "grid")
##   ..$ loop      :function (grid)
##   ..$ fit       :function (x, y, wts, param, lev, last, classProbs, ...)
##   ..$ predict   :function (modelFit, newdata, submodels = NULL)
##   ..$ prob      :function (modelFit, newdata, submodels = NULL)
##   ..$ predictors:function (x, surrogate = TRUE, ...)
##   ..$ varImp    :function (object, surrogates = FALSE, competes = TRUE, ...)
##   ..$ levels    :function (x)
```

```
##   ..$ trim      :function (x)
##   ..$ tags      : chr [1:4] "Tree-Based Model" "Implicit Feature Selection" "Handle Missing P
##   ..$ sort      :function (x)
##  $ modelType   : chr "Classification"
##  $ results     :'data.frame':    5 obs. of  5 variables:
##   ..$ cp        : num [1:5] 1e-05 1e-04 1e-03 1e-02 1e-01
##   ..$ Accuracy  : num [1:5] 0.825 0.825 0.836 0.841 0.803
##   ..$ Kappa     : num [1:5] 0.431 0.431 0.45 0.394 0
##   ..$ AccuracySD: num [1:5] 8.27e-03 8.27e-03 7.69e-03 6.21e-03 7.59e-05
##   ..$ KappaSD   : num [1:5] 0.0257 0.0257 0.0275 0.0473 0
##  $ pred        : NULL
##  $ bestTune    :'data.frame':    1 obs. of  1 variable:
##   ..$ cp: num 0.01
##  $ call        : language train.formula(form = y ~ ., data = dfTrain, method = "rpart", trCon
##  $ dots        : list()
##  $ metric      : chr "Accuracy"
##  $ control     :List of 27
##   ..$ method           : chr "cv"
##   ..$ number           : num 10
##   ..$ repeats          : logi NA
##   ..$ search           : chr "grid"
##   ..$ p                : num 0.75
##   ..$ initialWindow    : NULL
##   ..$ horizon          : num 1
##   ..$ fixedWindow      : logi TRUE
##   ..$ skip             : num 0
##   ..$ verboseIter      : logi FALSE
##   ..$ returnData       : logi TRUE
##   ..$ returnResamp     : chr "final"
##   ..$ savePredictions  : chr "none"
```

```
##   ..$ classProbs       : logi FALSE
##   ..$ summaryFunction  :function (data, lev = NULL, model = NULL)
##   ..$ selectionFunction: chr "best"
##   ..$ preProcOptions   :List of 2
##   .. ..$ pcaComp: num 50
##   .. ..$ thres  : num 0.95
##   ..$ sampling         : NULL
##   ..$ index            :List of 10
##   .. ..$ Fold01: int [1:9862] 1 2 3 4 5 6 7 8 9 10 ...
##   .. ..$ Fold02: int [1:9862] 1 2 4 5 6 7 8 10 11 12 ...
##   .. ..$ Fold03: int [1:9863] 1 2 3 4 5 6 7 9 10 11 ...
##   .. ..$ Fold04: int [1:9862] 1 2 3 4 5 6 7 8 9 10 ...
##   .. ..$ Fold05: int [1:9862] 1 3 4 5 6 8 9 10 11 12 ...
##   .. ..$ Fold06: int [1:9862] 1 2 3 4 6 7 8 9 10 11 ...
##   .. ..$ Fold07: int [1:9862] 1 2 3 4 5 6 7 8 9 10 ...
##   .. ..$ Fold08: int [1:9862] 1 2 3 4 5 7 8 9 10 12 ...
##   .. ..$ Fold09: int [1:9862] 1 2 3 4 5 6 7 8 9 10 ...
##   .. ..$ Fold10: int [1:9863] 2 3 5 6 7 8 9 11 12 13 ...
##   ..$ indexOut         :List of 10
##   .. ..$ Resample01: int [1:1096] 12 19 22 55 67 70 74 79 96 102 ...
##   .. ..$ Resample02: int [1:1096] 3 9 35 47 53 61 65 71 94 95 ...
##   .. ..$ Resample03: int [1:1095] 8 15 32 33 48 59 63 69 77 80 ...
##   .. ..$ Resample04: int [1:1096] 16 45 60 66 78 82 89 105 113 128 ...
##   .. ..$ Resample05: int [1:1096] 2 7 20 24 25 27 43 46 49 62 ...
##   .. ..$ Resample06: int [1:1096] 5 30 36 41 52 58 68 75 98 103 ...
##   .. ..$ Resample07: int [1:1096] 18 28 40 42 44 56 64 83 91 99 ...
##   .. ..$ Resample08: int [1:1096] 6 11 14 37 72 84 90 92 93 125 ...
##   .. ..$ Resample09: int [1:1096] 13 17 23 34 54 57 73 81 116 119 ...
##   .. ..$ Resample10: int [1:1095] 1 4 10 21 26 29 31 38 39 50 ...
##   ..$ indexFinal       : NULL
```

```
##   ..$ timingSamps      : num 0
##   ..$ predictionBounds : logi [1:2] FALSE FALSE
##   ..$ seeds            :List of 11
##   .. ..$ : int 10275
##   .. ..$ : int 590225
##   .. ..$ : int 735614
##   .. ..$ : int 409788
##   .. ..$ : int 936336
##   .. ..$ : int 336352
##   .. ..$ : int 476335
##   .. ..$ : int 780947
##   .. ..$ : int 114917
##   .. ..$ : int 318516
##   .. ..$ : int 403909
##   ..$ adaptive         :List of 4
##   .. ..$ min     : num 5
##   .. ..$ alpha   : num 0.05
##   .. ..$ method  : chr "gls"
##   .. ..$ complete: logi TRUE
##   ..$ trim             : logi FALSE
##   ..$ allowParallel    : logi TRUE
##  $ finalModel  :List of 19
##   ..$ frame              :'data.frame':  19 obs. of  9 variables:
##   .. ..$ var       : chr [1:19] "PC2" "PC3" "<leaf>" "PC8" ...
##   .. ..$ n         : int [1:19] 10958 8708 6744 1964 1214 750 127 623 148 475 ...
##   .. ..$ wt        : num [1:19] 10958 8708 6744 1964 1214 ...
##   .. ..$ dev       : num [1:19] 2160 1035 483 552 213 ...
##   .. ..$ yval      : num [1:19] 1 1 1 1 1 1 1 2 1 2 ...
##   .. ..$ complexity: num [1:19] 0.08079 0.01053 0.00324 0.01053 0.00671 ...
##   .. ..$ ncompete  : int [1:19] 4 4 0 4 0 4 0 4 0 4 ...
```

```
##   .. ..$ nsurrogate: int [1:19] 5 5 0 5 0 5 0 5 0 5 ...
##   .. ..$ yval2     : num [1:19, 1:6] 1 1 1 1 1 1 1 2 1 2 ...
##   .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. ..$ : NULL
##   .. .. .. ..$ : chr [1:6] "" "" "" "" ...
##   ..$ where              : int [1:10958] 11 3 12 19 17 3 3 3 3 3 ...
##   ..$ call               : language (function (formula, data, weights, subset, na.action = na
##   ..$ terms              :Classes 'terms', 'formula'  language .outcome ~ PC1 + PC2 + PC3 + P
##   .. .. ..- attr(*, "variables")= language list(.outcome, PC1, PC2, PC3, PC4, PC5, PC6, PC7,
##   .. .. ..- attr(*, "factors")= int [1:51, 1:50] 0 1 0 0 0 0 0 0 0 0 ...
##   .. .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. .. ..$ : chr [1:51] ".outcome" "PC1" "PC2" "PC3" ...
##   .. .. .. .. ..$ : chr [1:50] "PC1" "PC2" "PC3" "PC4" ...
##   .. .. ..- attr(*, "term.labels")= chr [1:50] "PC1" "PC2" "PC3" "PC4" ...
##   .. .. ..- attr(*, "order")= int [1:50] 1 1 1 1 1 1 1 1 1 1 ...
##   .. .. ..- attr(*, "intercept")= int 1
##   .. .. ..- attr(*, "response")= int 1
##   .. .. ..- attr(*, ".Environment")=<environment: 0x56406efd8378>
##   .. .. ..- attr(*, "predvars")= language list(.outcome, PC1, PC2, PC3, PC4, PC5, PC6, PC7, P
##   .. .. ..- attr(*, "dataClasses")= Named chr [1:51] "factor" "numeric" "numeric" "numeric" .
##   .. .. .. ..- attr(*, "names")= chr [1:51] ".outcome" "PC1" "PC2" "PC3" ...
##   ..$ cptable            : num [1:4, 1:3] 0.0808 0.0248 0.0105 0.01 0 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:4] "1" "2" "3" "4"
##   .. .. ..$ : chr [1:3] "CP" "nsplit" "rel error"
##   ..$ method             : chr "class"
##   ..$ parms              :List of 3
##   .. ..$ prior: num [1:2(1d)] 0.803 0.197
##   .. .. ..- attr(*, "dimnames")=List of 1
##   .. .. .. ..$ : chr [1:2] "1" "2"
```

```
##   .. ..$ loss : num [1:2, 1:2] 0 1 1 0
##   .. ..$ split: num 1
##   ..$ control            :List of 9
##   .. ..$ minsplit      : int 20
##   .. ..$ minbucket     : num 7
##   .. ..$ cp            : num 0
##   .. ..$ maxcompete    : int 4
##   .. ..$ maxsurrogate  : int 5
##   .. ..$ usesurrogate  : int 2
##   .. ..$ surrogatestyle: int 0
##   .. ..$ maxdepth      : int 30
##   .. ..$ xval          : num 0
##   ..$ functions          :List of 3
##   .. ..$ summary:function (yval, dev, wt, ylevel, digits)
##   .. ..$ print  :function (yval, ylevel, digits)
##   .. ..$ text   :function (yval, dev, wt, ylevel, digits, n, use.n)
##   ..$ numresp            : int 4
##   ..$ splits             : num [1:90, 1:5] 10958 10958 10958 10958 10958 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:90] "PC2" "PC1" "PC3" "PC8" ...
##   .. .. ..$ : chr [1:5] "count" "ncat" "improve" "index" ...
##   ..$ variable.importance: Named num [1:28] 577.3 171.7 132.4 83.1 67.9 ...
##   .. ..- attr(*, "names")= chr [1:28] "PC2" "PC3" "PC6" "PC8" ...
##   ..$ y                  : int [1:10958] 2 1 2 1 1 2 2 1 1 1 ...
##   ..$ ordered            : Named logi [1:50] FALSE FALSE FALSE FALSE FALSE FALSE ...
##   .. ..- attr(*, "names")= chr [1:50] "PC1" "PC2" "PC3" "PC4" ...
##   ..$ xNames             : chr [1:50] "PC1" "PC2" "PC3" "PC4" ...
##   ..$ problemType        : chr "Classification"
##   ..$ tuneValue          :'data.frame':  1 obs. of  1 variable:
##   .. ..$ cp: num 0.01
```

```
##   ..$ obsLevels          : chr [1:2] "0" "1"
##   .. ..- attr(*, "ordered")= logi FALSE
##   ..$ param              : list()
##   ..- attr(*, "xlevels")= Named list()
##   ..- attr(*, "ylevels")= chr [1:2] "0" "1"
##   ..- attr(*, "class")= chr "rpart"
##  $ preProcess  :List of 22
##   ..$ dim              : int [1:2] 10958 320
##   ..$ bc               : NULL
##   ..$ yj               : NULL
##   ..$ et               : NULL
##   ..$ invHyperbolicSine: NULL
##   ..$ mean             : Named num [1:320] 56484 56315 56141 55956 55750 ...
##   .. ..- attr(*, "names")= chr [1:320] "x.X1" "x.X2" "x.X3" "x.X4" ...
##   ..$ std              : Named num [1:320] 1070 1124 1175 1227 1284 ...
##   .. ..- attr(*, "names")= chr [1:320] "x.X1" "x.X2" "x.X3" "x.X4" ...
##   ..$ ranges           : NULL
##   ..$ rotation         : num [1:320, 1:50] -0.0574 -0.0557 -0.0536 -0.0509 -0.0474 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:320] "x.X1" "x.X2" "x.X3" "x.X4" ...
##   .. .. ..$ : chr [1:50] "PC1" "PC2" "PC3" "PC4" ...
##   ..$ method           :List of 4
##   .. ..$ pca   : chr [1:320] "x.X1" "x.X2" "x.X3" "x.X4" ...
##   .. ..$ center: chr [1:320] "x.X1" "x.X2" "x.X3" "x.X4" ...
##   .. ..$ scale : chr [1:320] "x.X1" "x.X2" "x.X3" "x.X4" ...
##   .. ..$ ignore: chr(0)
##   ..$ thresh           : num 0.95
##   ..$ pcaComp          : num 50
##   ..$ numComp          : num 50
##   ..$ ica              : NULL
```

```
##   ..$ wildcards        :List of 2
##   .. ..$ PCA: chr(0)
##   .. ..$ ICA: chr(0)
##   ..$ k                : num 5
##   ..$ knnSummary       :function (x, ...)
##   ..$ bagImp           : NULL
##   ..$ median           : NULL
##   ..$ data             : NULL
##   ..$ rangeBounds      : num [1:2] 0 1
##   ..$ call             : chr "scrubed"
##   ..- attr(*, "class")= chr "preProcess"
##  $ trainingData:'data.frame':    10958 obs. of  321 variables:
##   ..$ .outcome: Factor w/ 2 levels "0","1": 2 1 2 1 1 2 2 1 1 1 ...
##   ..$ x.X1    : num [1:10958] 57043 56963 56523 54628 53584 ...
##   ..$ x.X2    : num [1:10958] 56535 56493 55971 53980 53391 ...
##   ..$ x.X3    : num [1:10958] 55884 55931 55304 53494 53310 ...
##   ..$ x.X4    : num [1:10958] 55176 55340 54498 53073 53293 ...
##   ..$ x.X5    : num [1:10958] 54458 54769 53725 52616 53340 ...
##   ..$ x.X6    : num [1:10958] 56980 56857 56515 54776 53782 ...
##   ..$ x.X7    : num [1:10958] 56461 56362 55978 54089 53498 ...
##   ..$ x.X8    : num [1:10958] 55800 55777 55327 53517 53307 ...
##   ..$ x.X9    : num [1:10958] 55051 55164 54570 53013 53213 ...
##   ..$ x.X10   : num [1:10958] 54319 54585 53863 52501 53157 ...
##   ..$ x.X11   : num [1:10958] 56900 56724 56467 54908 54015 ...
##   ..$ x.X12   : num [1:10958] 56368 56187 55965 54205 53674 ...
##   ..$ x.X13   : num [1:10958] 55721 55587 55360 53594 53378 ...
##   ..$ x.X14   : num [1:10958] 54946 54947 54646 53021 53200 ...
##   ..$ x.X15   : num [1:10958] 54145 54367 53999 52451 53055 ...
##   ..$ x.X16   : num [1:10958] 56816 56568 56401 55059 54327 ...
##   ..$ x.X17   : num [1:10958] 56257 55987 55902 54295 53911 ...
```

```
##   ..$ x.X18   : num [1:10958] 55559 55360 55339 53657 53529 ...
##   ..$ x.X19   : num [1:10958] 54788 54690 54720 53074 53260 ...
##   ..$ x.X20   : num [1:10958] 53973 54125 54110 52486 53065 ...
##   ..$ x.X21   : num [1:10958] 56717 56378 56318 55218 54670 ...
##   ..$ x.X22   : num [1:10958] 56191 55769 55827 54480 54173 ...
##   ..$ x.X23   : num [1:10958] 55436 55094 55258 53787 53726 ...
##   ..$ x.X24   : num [1:10958] 54555 54384 54695 53144 53383 ...
##   ..$ x.X25   : num [1:10958] 53771 53843 54135 52556 53162 ...
##   ..$ x.X26   : num [1:10958] 56569 56149 56179 55318 54984 ...
##   ..$ x.X27   : num [1:10958] 56045 55520 55701 54622 54459 ...
##   ..$ x.X28   : num [1:10958] 55331 54806 55120 53892 53963 ...
##   ..$ x.X29   : num [1:10958] 54340 54074 54545 53227 53560 ...
##   ..$ x.X30   : num [1:10958] 53472 53527 54046 52668 53303 ...
##   ..$ x.X31   : num [1:10958] 56460 55912 55999 55373 55250 ...
##   ..$ x.X32   : num [1:10958] 55840 55206 55483 54656 54735 ...
##   ..$ x.X33   : num [1:10958] 55120 54439 54892 53930 54202 ...
##   ..$ x.X34   : num [1:10958] 54201 53729 54327 53323 53745 ...
##   ..$ x.X35   : num [1:10958] 53217 53129 53857 52814 53440 ...
##   ..$ x.X36   : num [1:10958] 56366 55680 55801 55400 55462 ...
##   ..$ x.X37   : num [1:10958] 55708 54900 55218 54724 54966 ...
##   ..$ x.X38   : num [1:10958] 54886 54069 54602 54033 54396 ...
##   ..$ x.X39   : num [1:10958] 54003 53359 54067 53419 53902 ...
##   ..$ x.X40   : num [1:10958] 53029 52674 53594 52913 53556 ...
##   ..$ x.X41   : num [1:10958] 283 284 284 277 276 ...
##   ..$ x.X42   : num [1:10958] 281 282 281 275 275 ...
##   ..$ x.X43   : num [1:10958] 278 279 279 274 273 ...
##   ..$ x.X44   : num [1:10958] 276 277 278 274 272 ...
##   ..$ x.X45   : num [1:10958] 275 276 277 274 271 ...
##   ..$ x.X46   : num [1:10958] 282 283 283 278 276 ...
##   ..$ x.X47   : num [1:10958] 281 281 280 275 275 ...
```

```
##   ..$ x.X48   : num [1:10958] 277 278 278 274 274 ...
##   ..$ x.X49   : num [1:10958] 275 276 277 274 272 ...
##   ..$ x.X50   : num [1:10958] 274 274 276 274 271 ...
##   ..$ x.X51   : num [1:10958] 282 281 282 278 276 ...
##   ..$ x.X52   : num [1:10958] 280 280 279 275 275 ...
##   ..$ x.X53   : num [1:10958] 276 277 277 274 273 ...
##   ..$ x.X54   : num [1:10958] 275 274 276 274 271 ...
##   ..$ x.X55   : num [1:10958] 273 272 276 275 272 ...
##   ..$ x.X56   : num [1:10958] 282 281 280 279 278 ...
##   ..$ x.X57   : num [1:10958] 278 279 277 275 275 ...
##   ..$ x.X58   : num [1:10958] 275 276 276 274 272 ...
##   ..$ x.X59   : num [1:10958] 275 273 275 274 272 ...
##   ..$ x.X60   : num [1:10958] 272 270 275 275 272 ...
##   ..$ x.X61   : num [1:10958] 283 281 280 281 280 ...
##   ..$ x.X62   : num [1:10958] 279 278 276 277 276 ...
##   ..$ x.X63   : num [1:10958] 276 274 274 275 273 ...
##   ..$ x.X64   : num [1:10958] 275 271 274 275 272 ...
##   ..$ x.X65   : num [1:10958] 271 268 273 275 273 ...
##   ..$ x.X66   : num [1:10958] 283 280 280 282 283 ...
##   ..$ x.X67   : num [1:10958] 281 277 275 279 278 ...
##   ..$ x.X68   : num [1:10958] 277 274 273 276 275 ...
##   ..$ x.X69   : num [1:10958] 274 270 271 275 273 ...
##   ..$ x.X70   : num [1:10958] 270 266 271 275 273 ...
##   ..$ x.X71   : num [1:10958] 281 279 278 282 285 ...
##   ..$ x.X72   : num [1:10958] 281 275 274 280 280 ...
##   ..$ x.X73   : num [1:10958] 278 272 272 277 276 ...
##   ..$ x.X74   : num [1:10958] 275 268 270 276 274 ...
##   ..$ x.X75   : num [1:10958] 270 263 269 274 274 ...
##   ..$ x.X76   : num [1:10958] 281 277 274 282 286 ...
##   ..$ x.X77   : num [1:10958] 281 274 272 280 282 ...
```

```
##   ..$ x.X78   : num [1:10958] 278 271 270 278 277 ...
##   ..$ x.X79   : num [1:10958] 275 266 269 276 274 ...
##   ..$ x.X80   : num [1:10958] 269 261 267 272 273 ...
##   ..$ x.X81   : num [1:10958] 277 276 276 269 266 ...
##   ..$ x.X82   : num [1:10958] 274 274 274 266 264 ...
##   ..$ x.X83   : num [1:10958] 272 271 271 264 264 ...
##   ..$ x.X84   : num [1:10958] 268 267 269 264 263 ...
##   ..$ x.X85   : num [1:10958] 266 265 268 264 263 ...
##   ..$ x.X86   : num [1:10958] 277 276 276 270 266 ...
##   ..$ x.X87   : num [1:10958] 274 273 273 267 265 ...
##   ..$ x.X88   : num [1:10958] 271 270 271 265 264 ...
##   ..$ x.X89   : num [1:10958] 267 267 268 264 263 ...
##   ..$ x.X90   : num [1:10958] 265 264 268 264 263 ...
##   ..$ x.X91   : num [1:10958] 276 275 276 271 268 ...
##   ..$ x.X92   : num [1:10958] 274 273 273 268 266 ...
##   ..$ x.X93   : num [1:10958] 271 269 271 266 264 ...
##   ..$ x.X94   : num [1:10958] 267 266 268 265 263 ...
##   ..$ x.X95   : num [1:10958] 265 263 267 265 263 ...
##   ..$ x.X96   : num [1:10958] 276 275 275 273 270 ...
##   ..$ x.X97   : num [1:10958] 274 272 272 269 267 ...
##   ..$ x.X98   : num [1:10958] 270 268 270 266 265 ...
##   .. [list output truncated]
##  $ ptype       :'data.frame':    0 obs. of  320 variables:
##   ..$ x.X1  : num(0)
##   ..$ x.X2  : num(0)
##   ..$ x.X3  : num(0)
##   ..$ x.X4  : num(0)
##   ..$ x.X5  : num(0)
##   ..$ x.X6  : num(0)
##   ..$ x.X7  : num(0)
```

```
##   ..$ x.X8  : num(0)
##   ..$ x.X9  : num(0)
##   ..$ x.X10 : num(0)
##   ..$ x.X11 : num(0)
##   ..$ x.X12 : num(0)
##   ..$ x.X13 : num(0)
##   ..$ x.X14 : num(0)
##   ..$ x.X15 : num(0)
##   ..$ x.X16 : num(0)
##   ..$ x.X17 : num(0)
##   ..$ x.X18 : num(0)
##   ..$ x.X19 : num(0)
##   ..$ x.X20 : num(0)
##   ..$ x.X21 : num(0)
##   ..$ x.X22 : num(0)
##   ..$ x.X23 : num(0)
##   ..$ x.X24 : num(0)
##   ..$ x.X25 : num(0)
##   ..$ x.X26 : num(0)
##   ..$ x.X27 : num(0)
##   ..$ x.X28 : num(0)
##   ..$ x.X29 : num(0)
##   ..$ x.X30 : num(0)
##   ..$ x.X31 : num(0)
##   ..$ x.X32 : num(0)
##   ..$ x.X33 : num(0)
##   ..$ x.X34 : num(0)
##   ..$ x.X35 : num(0)
##   ..$ x.X36 : num(0)
##   ..$ x.X37 : num(0)
```

```
##   ..$ x.X38 : num(0)
##   ..$ x.X39 : num(0)
##   ..$ x.X40 : num(0)
##   ..$ x.X41 : num(0)
##   ..$ x.X42 : num(0)
##   ..$ x.X43 : num(0)
##   ..$ x.X44 : num(0)
##   ..$ x.X45 : num(0)
##   ..$ x.X46 : num(0)
##   ..$ x.X47 : num(0)
##   ..$ x.X48 : num(0)
##   ..$ x.X49 : num(0)
##   ..$ x.X50 : num(0)
##   ..$ x.X51 : num(0)
##   ..$ x.X52 : num(0)
##   ..$ x.X53 : num(0)
##   ..$ x.X54 : num(0)
##   ..$ x.X55 : num(0)
##   ..$ x.X56 : num(0)
##   ..$ x.X57 : num(0)
##   ..$ x.X58 : num(0)
##   ..$ x.X59 : num(0)
##   ..$ x.X60 : num(0)
##   ..$ x.X61 : num(0)
##   ..$ x.X62 : num(0)
##   ..$ x.X63 : num(0)
##   ..$ x.X64 : num(0)
##   ..$ x.X65 : num(0)
##   ..$ x.X66 : num(0)
##   ..$ x.X67 : num(0)
```

```
##   ..$ x.X68 : num(0)
##   ..$ x.X69 : num(0)
##   ..$ x.X70 : num(0)
##   ..$ x.X71 : num(0)
##   ..$ x.X72 : num(0)
##   ..$ x.X73 : num(0)
##   ..$ x.X74 : num(0)
##   ..$ x.X75 : num(0)
##   ..$ x.X76 : num(0)
##   ..$ x.X77 : num(0)
##   ..$ x.X78 : num(0)
##   ..$ x.X79 : num(0)
##   ..$ x.X80 : num(0)
##   ..$ x.X81 : num(0)
##   ..$ x.X82 : num(0)
##   ..$ x.X83 : num(0)
##   ..$ x.X84 : num(0)
##   ..$ x.X85 : num(0)
##   ..$ x.X86 : num(0)
##   ..$ x.X87 : num(0)
##   ..$ x.X88 : num(0)
##   ..$ x.X89 : num(0)
##   ..$ x.X90 : num(0)
##   ..$ x.X91 : num(0)
##   ..$ x.X92 : num(0)
##   ..$ x.X93 : num(0)
##   ..$ x.X94 : num(0)
##   ..$ x.X95 : num(0)
##   ..$ x.X96 : num(0)
##   ..$ x.X97 : num(0)
```

```
##   ..$ x.X98 : num(0)
##   ..$ x.X99 : num(0)
##   .. [list output truncated]
##  $ resample    :'data.frame':    10 obs. of  3 variables:
##   ..$ Accuracy: num [1:10] 0.848 0.851 0.842 0.83 0.833 ...
##   ..$ Kappa   : num [1:10] 0.439 0.393 0.421 0.319 0.328 ...
##   ..$ Resample: chr [1:10] "Fold08" "Fold03" "Fold01" "Fold09" ...
##  $ resampledCM :'data.frame':    50 obs. of  6 variables:
##   ..$ cp      : num [1:50] 1e-05 1e-04 1e-03 1e-02 1e-01 1e-05 1e-04 1e-03 1e-02 1e-01 ...
##   ..$ cell1   : num [1:50] 793 793 813 833 880 808 808 823 827 880 ...
##   ..$ cell2   : num [1:50] 87 87 67 47 0 72 72 57 53 0 ...
##   ..$ cell3   : num [1:50] 105 105 102 126 216 105 105 117 122 216 ...
##   ..$ cell4   : num [1:50] 111 111 114 90 0 111 111 99 94 0 ...
##   ..$ Resample: chr [1:50] "Fold01" "Fold01" "Fold01" "Fold01" ...
##  $ perfNames   : chr [1:2] "Accuracy" "Kappa"
##  $ maximize    : logi TRUE
##  $ yLimits     : NULL
##  $ times       :List of 3
##   ..$ everything: 'proc_time' Named num [1:5] 81.2 69.9 60.5 0 0
##   .. ..- attr(*, "names")= chr [1:5] "user.self" "sys.self" "elapsed" "user.child" ...
##   ..$ final     : 'proc_time' Named num [1:5] 7 6.34 4.21 0 0
##   .. ..- attr(*, "names")= chr [1:5] "user.self" "sys.self" "elapsed" "user.child" ...
##   ..$ prediction: logi [1:3] NA NA NA
##  $ levels      : chr [1:2] "0" "1"
##   ..- attr(*, "ordered")= logi FALSE
##  $ terms       :Classes 'terms', 'formula'  language y ~ x.X1 + x.X2 + x.X3 + x.X4 + x.X5 + x
##   .. ..- attr(*, "variables")= language list(y, x.X1, x.X2, x.X3, x.X4, x.X5, x.X6, x.X7, x.X
##   .. ..- attr(*, "factors")= int [1:321, 1:320] 0 1 0 0 0 0 0 0 0 0 ...
##   .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. ..$ : chr [1:321] "y" "x.X1" "x.X2" "x.X3" ...
```

```
##   .. .. .. ..$ : chr [1:320] "x.X1" "x.X2" "x.X3" "x.X4" ...
##   .. ..- attr(*, "term.labels")= chr [1:320] "x.X1" "x.X2" "x.X3" "x.X4" ...
##   .. ..- attr(*, "order")= int [1:320] 1 1 1 1 1 1 1 1 1 1 ...
##   .. ..- attr(*, "intercept")= int 1
##   .. ..- attr(*, "response")= int 1
##   .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. ..- attr(*, "predvars")= language list(y, x.X1, x.X2, x.X3, x.X4, x.X5, x.X6, x.X7, x.X8
##   .. ..- attr(*, "dataClasses")= Named chr [1:321] "factor" "numeric" "numeric" "numeric" ...
##   .. .. ..- attr(*, "names")= chr [1:321] "y" "x.X1" "x.X2" "x.X3" ...
##  $ coefnames   : chr [1:320] "x.X1" "x.X2" "x.X3" "x.X4" ...
##  $ xlevels     : Named list()
##  - attr(*, "class")= chr [1:2] "train" "train.formula"
```

Evaluamos ahora el modelo respecto a toda la muestra:

```
yTree <- predict(treeFit,newdata = dfTrain)
confusionMatrix(as.factor(occu),yTree)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 8433  365
##          1 1216  944
##
##                Accuracy : 0.8557
##                  95% CI : (0.849, 0.8623)
##     No Information Rate : 0.8805
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.4646
##
```

```
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.8740
##             Specificity : 0.7212
##          Pos Pred Value : 0.9585
##          Neg Pred Value : 0.4370
##              Prevalence : 0.8805
##          Detection Rate : 0.7696
##    Detection Prevalence : 0.8029
##       Balanced Accuracy : 0.7976
##
##        'Positive' Class : 0
##
```

Incluyendo el AUC para comparar adecuadamente ambos modelos:

```
yTree <- predict(treeFit,newdata = dfTrain, type="prob")
roc_obj <- roc(as.factor(occu), yTree[["1"]])
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
auc(roc_obj)
```

```
## Area under the curve: 0.8009
```

Muestras Desbalanceadas: A la vista de los resultados anteriores, ¿se observa algún problema asociado a la frecuencia de los casos positivos y negativos? Este problema se conoce como desbalanceo de la muestra y hay diferentes aproximaciones para resolverlo, algunas de las cuales están recogidas en CARET. Ver la Sección 11.

Clasificación:

Una vez resuelta la parte binaria de la precipitación, pasamos a predecir la cantidad para aquellos días en los cuales se haya dado precipitación. En el caso de los modelos lineales esta división es necesaria por las hipótesis impuestas sobre las funciones de enlace, el comportamiento de los residuos, etc... ¿Ocurre lo mismo para los modelos basados en árboles?

Consideremos la muestra asociada a los días de precipitación:

```
indRain <- which(y > 1)
dfTrain <- data.frame(y = y[indRain], x=x[indRain,])
```

Notar que la variable `indRain` contiene las posiciones de los días con precipitación y nos permitirá asignar los valores predichos a las posiciones correspondientes del vector de predicciones sobre todo el periodo.

Modelos Lineales Generalizados:

Como vimos en el problema de clasificación, realmente no tenemos ningún parámetro extra que ajustar en el GLM, de modo que podemos replicar el ajuste, esta vez considerando únicamente los datos de precipitación:

```
trainParams <- trainControl(method = "cv", number = 10,
                            preProcOptions = list(pcaComp = 50, thres = 0.95))
modelFit <- train(y ~ ., data=dfTrain, method="glm",
                  trControl=trainParams, preProcess = "pca")
## str(modelFit)
```

Obtenemos ahora las predicciones sobre los días de precipitación:

```
yPred <- predict(modelFit,newdata = dfTrain)
```

Validemos ahora la predicción obtenida con diferentes parámetros:

- Day-to-day correspondence: ¿qué medidas conoces que evalúen la correspondencia entre ambas series?
- Accuracy: ¿qué medidas de precisión conoces?

- Distributional similarity: ¿qué medidas de comparación de la distribución estadística conoces?

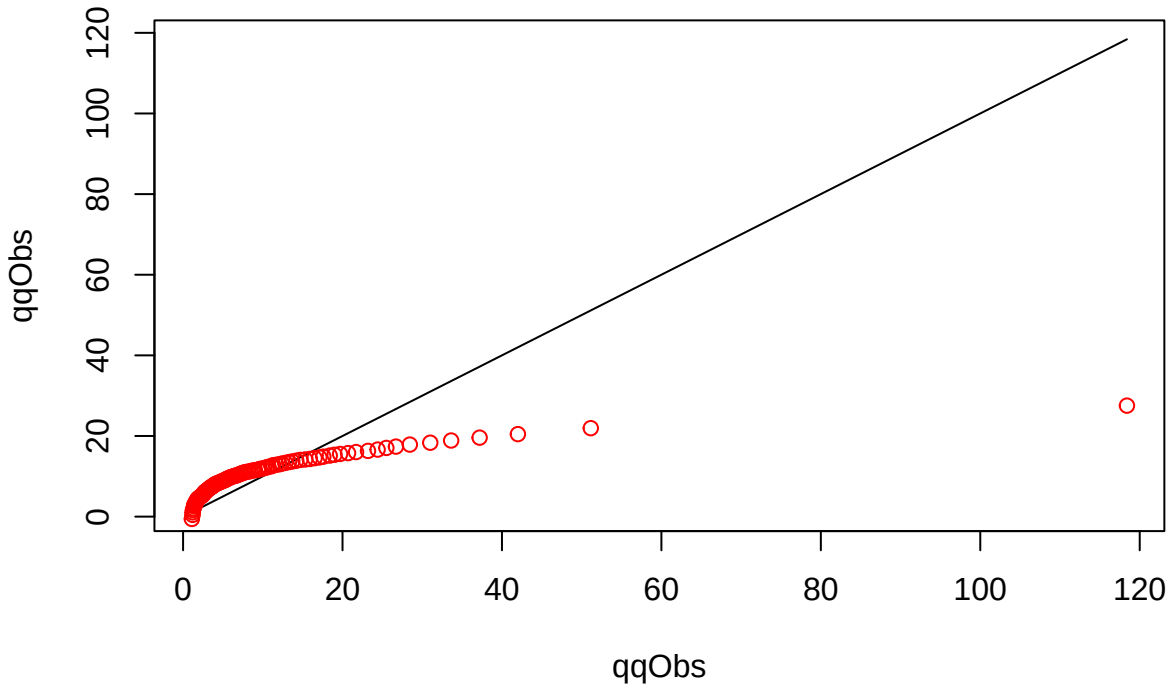
Revisar los parámetros de validación incluidos en el paquete `hydroGOF` y ver si alguno se corresponde con las características antes expuestas.

```
## Por ejemplo, RMSE:
RMSEglm=rmse(yPred,dfTrain$y)
print(RMSEglm)
```

```
## [1] 9.815054
```

El Kolmogorov-Smirnov Test (? ks.test) ¿con qué característica de las anteriores se corresponde?

```
qqObs <- quantile(dfTrain$y, probs = seq(0, 1, 0.01), na.rm = TRUE)
qqPrd <- quantile(yPred, probs = seq(0, 1, 0.01), na.rm = TRUE)
par(mfrow=c(1,1))
plot(qqObs,qqObs, type = "l")
points(qqObs,qqPrd, col = "red")
```



A la vista del gráfico anterior, ¿cómo se está comportando nuestro modelo?

Árboles de Regresión:

En base a lo visto en los puntos anterior, extiende el análisis a los métodos basados en árboles y comparar con los resultados obtenidos para el modelo lineal.

Session Info:

```
## R version 4.1.1 (2021-08-10)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.3 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
## LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=es_ES.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=es_ES.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=es_ES.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=es_ES.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] grid      stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] rpart_4.1-15    pROC_1.18.0     hydroGOF_0.4-0  zoo_1.8-9
##  [5] fields_11.6     spam_2.6-0      dotCall64_1.0-0 caret_6.0-90
```

```
##  [9] lattice_0.20-44 ggplot2_3.3.5
##
## loaded via a namespace (and not attached):
##  [1] maps_3.3.0           splines_4.1.1        foreach_1.5.1
##  [4] prodlim_2019.11.13   highr_0.8            sp_1.4-5
##  [7] stats4_4.1.1         yaml_2.2.1           globals_0.14.0
## [10] ipred_0.9-12         pillar_1.6.4         automap_1.0-14
## [13] glue_1.4.2           digest_0.6.27        colorspace_2.0-2
## [16] recipes_0.1.17       htmltools_0.5.1.1    Matrix_1.3-4
## [19] plyr_1.8.6           hydroTSM_0.6-0       timeDate_3043.102
## [22] pkgconfig_2.0.3      listenv_0.8.0        purrr_0.3.4
## [25] scales_1.1.1         intervals_0.15.2     gower_0.2.2
## [28] lava_1.6.10          gstat_2.0-7          tibble_3.1.6
## [31] proxy_0.4-26         generics_0.1.0       ellipsis_0.3.2
## [34] withr_2.4.3          nnet_7.3-16          survival_3.2-13
## [37] magrittr_2.0.1       crayon_1.4.2         maptools_1.1-2
## [40] evaluate_0.14        future_1.23.0        fansi_0.5.0
## [43] parallelly_1.29.0    nlme_3.1-152         MASS_7.3-54
## [46] xts_0.12.1           foreign_0.8-81       class_7.3-19
## [49] FNN_1.1.3            tools_4.1.1          data.table_1.14.2
## [52] lifecycle_1.0.1      stringr_1.4.0        munsell_0.5.0
## [55] compiler_4.1.1       e1071_1.7-8          spacetime_1.2-5
## [58] rlang_0.4.12         iterators_1.0.13     rmarkdown_2.11
## [61] gtable_0.3.0         ModelMetrics_1.2.2.2 codetools_0.2-18
## [64] reshape_0.8.8        reshape2_1.4.4       R6_2.5.0
## [67] lubridate_1.8.0      knitr_1.31           dplyr_1.0.7
## [70] future.apply_1.8.1   utf8_1.2.2           stringi_1.5.3
## [73] parallel_4.1.1       Rcpp_1.0.6           vctrs_0.3.8
## [76] tidyselect_1.1.1     xfun_0.29
```