Modelamiento Estadístico y Sistemas Recomendadores: Foro 2

Patricio Águila Márquez

Instrucciones

Considere el conjunto de datos 'hr.csv', que contiene información de 14.999 empleados de una multinacional de EE.UU., generada con el objeto de estudiar las razones que explican la rotación de personal. Las variables que se encuentran en el conjunto de datos se describen en la siguiente tabla.

Variable	**Descripción**
Satisfaction_level	Nivel de satisfacción porcentual del empleado.
Last_evaluation	Puntaje porcentual obtenido en la última evaluación laboral.
Number_project	Número de proyectos en los que participa el empleado.
Average montly hours	Promedio de horas mensuales que trabajó el empleado.
Time spend company	Tiempo (en meses) de trabajo del empleado en la compañía.
Work_accident	Indica 1 si el empleado tuvo accidentes laborales y 0 si no.
Left	Indica 1 si el empleado dejó la compañía y 0 si no.
Promotion last 5 years	Indica 1 si el empleado fue ascendido en los últimos 5 años y 0 si no.
Area	Indica el área en la que se desempeña el empleador.
Salary	Indica el sueldo del empleado (medio, bajo o alto).

En este Foro entrenaremos y evaluaremos clasificadores diseñados para predecir la variable 'Left', la cual indica que un empleado dejó o no la compañía. Recuerden adicionalmente instalar las librerías: rpart, rpart.plot, e1071, adabag, randomForest, rminer y caret en R.

Luego, desarrolle las siguientes actividades:

1) Cargue el conjunto de datos en la sesión de trabajo de R usando la función `read.table`. Defina la variable Left como factor, utilizando la función `factor()`.

```
datos <- read.csv("../05 Foro 2/hr.csv",header=TRUE, sep=",")
datos$left <- factor(datos$left)
```

2) Seleccione de manera aleatoria 2/3 de los datos para crear sus datos de entrenamiento y guarde el tercio restante para objeto de validación. Para esto, simule valores 1 y 2 en proporciones 2/3 a 1/3 a través de la función `sample()`. Utilice aquellas tuplas de la base de datos asociadas al valor 1 para la base de entrenamiento, y las restantes para validación. Utilice la semilla 1, mediante la función `set.seed(1)`.

```
# Se define un valor "seed" para que todos trabajemos con la misma agrupación
# aleatoria de datos, tanto para el conjunto de entrenamiento como el de validación.
set.seed(1)

# Variable que contiene un vector de datos para posterior creación de los
# conjuntos de entrenamiento y validación.
ind <- sample(2, length(datos$left), replace=TRUE, prob=c(2/3, 1/3))
table(ind)
```

```
## ind
##    1    2
## 9954 5045
```

```
# Datos de entrenamiento
datos.trabajo <- datos[ind==1,]
dim(datos.trabajo)
```

```
## [1] 9954   10
```

```
# Datos de validación
datos.validacion <- datos[ind==2,]
dim(datos.validacion)
```

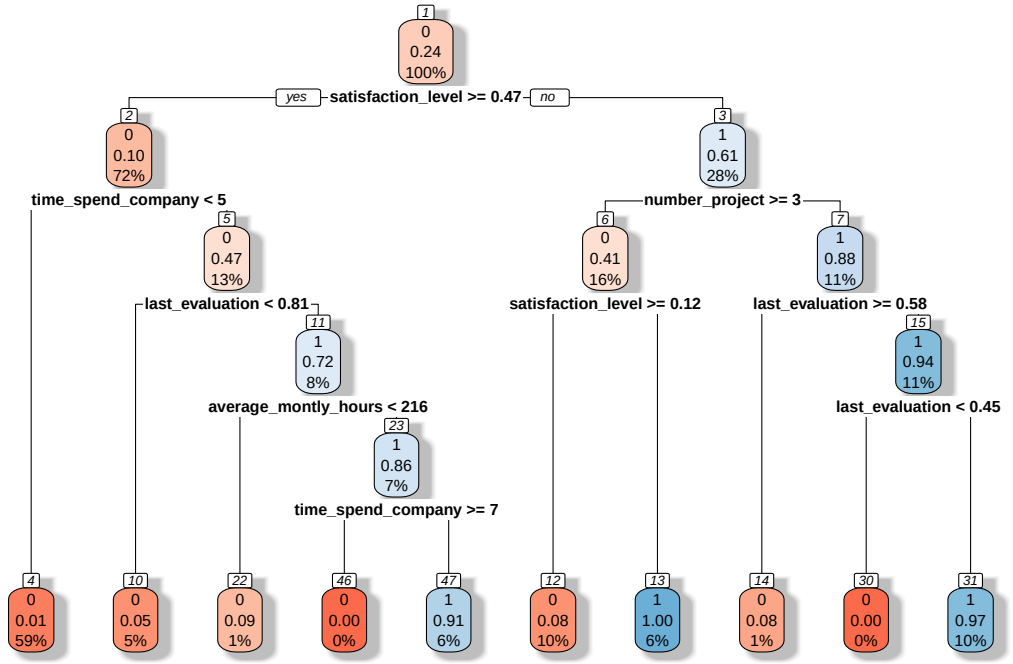
```
## [1] 5045   10
```

3) Construya un árbol de decisión para la variable Left, utilizando como criterio el índice de Gini. Realice el procedimiento completo, incluyendo la poda del árbol, usando los comandos `rpart()`, `cptable()` y `prune()` de la librería `rpart`. Use la opción `cp = 0.06` en el proceso de poda. ¿Qué diferencias y similitudes entre los dos árboles puede observar? CHEQUEAR ESE VALOR CP=0.06.

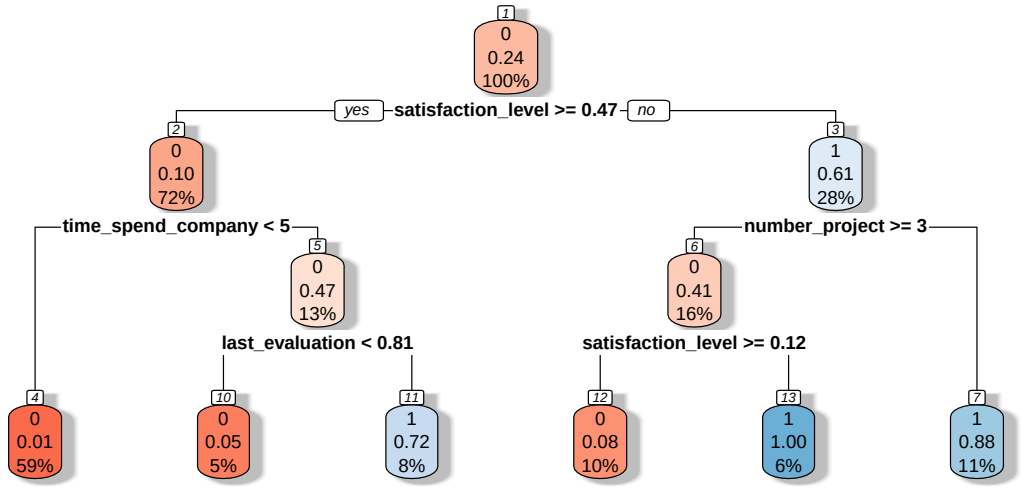
```
# Recordemos que las clases dispoibles son: 0 (se queda) y 1 (se va)
# Se deben importar las librerias necesarias.
library(rpart)
library(rpart.plot)

# Se entrena el Arbol de Decision que tiene como objetivo predecir la variable
# "left", usando el criterio de gini.
fit <- rpart(left ~ .,data=datos.trabajo,parms = list(split = "gini"))

# Se imprime el modelo. Esta visualizacion no entrega mucha información.
print(fit)
```

```
## n= 9954
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 9954 2381 0 (0.76079968 0.23920032)
##    2) satisfaction_level>=0.465 7180  700 0 (0.90250696 0.09749304)
##      4) time_spend_company< 4.5 5863   85 0 (0.98550230 0.01449770) *
##      5) time_spend_company>=4.5 1317  615 0 (0.53302961 0.46697039)
##       10) last_evaluation< 0.805 499   25 0 (0.94989980 0.05010020) *
##       11) last_evaluation>=0.805 818  228 1 (0.27872861 0.72127139)
##         22) average_montly_hours< 215.5 146   13 0 (0.91095890 0.08904110) *
##         23) average_montly_hours>=215.5 672   95 1 (0.14136905 0.85863095)
##           46) time_spend_company>=6.5 40    0 0 (1.00000000 0.00000000) *
##           47) time_spend_company< 6.5 632   55 1 (0.08702532 0.91297468) *
##    3) satisfaction_level< 0.465 2774 1093 1 (0.39401586 0.60598414)
##      6) number_project>=2.5 1634  677 0 (0.58567931 0.41432069)
##       12) satisfaction_level>=0.115 1035   78 0 (0.92463768 0.07536232) *
##       13) satisfaction_level< 0.115 599    0 1 (0.00000000 1.00000000) *
##      7) number_project< 2.5 1140  136 1 (0.11929825 0.88070175)
##       14) last_evaluation>=0.575 84    7 0 (0.91666667 0.08333333) *
##       15) last_evaluation< 0.575 1056   59 1 (0.05587121 0.94412879)
##         30) last_evaluation< 0.445 25    0 0 (1.00000000 0.00000000) *
##         31) last_evaluation>=0.445 1031   34 1 (0.03297769 0.96702231) *
```

```
# En el gráfico: Cada nodo del árbol contiene: la probabilidad de pertenecer a
# la clase positiva. En este caso, la clase positiva es: 1, ya que esa clase es
# la asociada al despido.
# Además, se muestra porcentaje de los datos que se encuentra en ese nodo.

rpart.plot(fit, box.palette="RdBu", shadow.col="gray", nn=TRUE)
```




```r
# Poda del árbol. Utilizamos función prune y un coeficiente cp.
pfit <- prune(fit, cp = 0.06)

# Graficamos árbol podado
rpart.plot(pfit, box.palette="RdBu", shadow.col="gray", nn=TRUE)
```



4) Utilizando la función `naiveBayes()` de la librería `e1071` construya un clasificador de Bayes Ingenuo.

```
# Se importa la librería 'e1071'
library(e1071)
# Clasificador de Bayes Ingenuo
fitNB <- naiveBayes(left ~ .,data=datos.trabajo)
```

5) Construya un clasificador de la variable Left del tipo Bagging, usando la función `bagging()` de la librería `adabag`. Utilice la semilla 1, mediante la función `set.seed(1)`. Utilice 10 árboles, mediante la opción `mfinal=10`.

```
# Se importa la librería 'adabag'
library(adabag)
```

```
## Loading required package: caret
```

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

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

```
## Loading required package: foreach
```

```
## Loading required package: doParallel
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
set.seed(1)
# Clasificador tipo Bagging
# Probar con otros valores de 'mfinal'
fit.bagging <- bagging(left ~., data=datos.trabajo, mfinal=10)
```

6) Construya un clasificador de la variable Left del tipo Boosting, usando la función `boosting()` de la librería `adabag`. Utilice la semilla 1, mediante la función `set.seed(1)`. Utilice 10 árboles, mediante la opción `mfinal=10`.

```
set.seed(1)
# Clasificador tipo Boosting
# Probar con otros valores de 'mfinal'
fit.boosting <- boosting(left ~., data=datos.trabajo, boos=TRUE, mfinal=10)
```

7) Construya un clasificador de la variable left del tipo Random Forest, usando la función `randomForest()` de la librería `randomForest`. Utilice la semilla 1, mediante la función `set.seed(1)`. Utilice 100 árboles, mediante la opción `ntree=100`. ¿Cual de todos los algoritmos tarda más en entrenar?. Si su computador no es capaz de ejecutar esta instrucción, intente reducir el número de árboles.

```
library(randomForest)
```

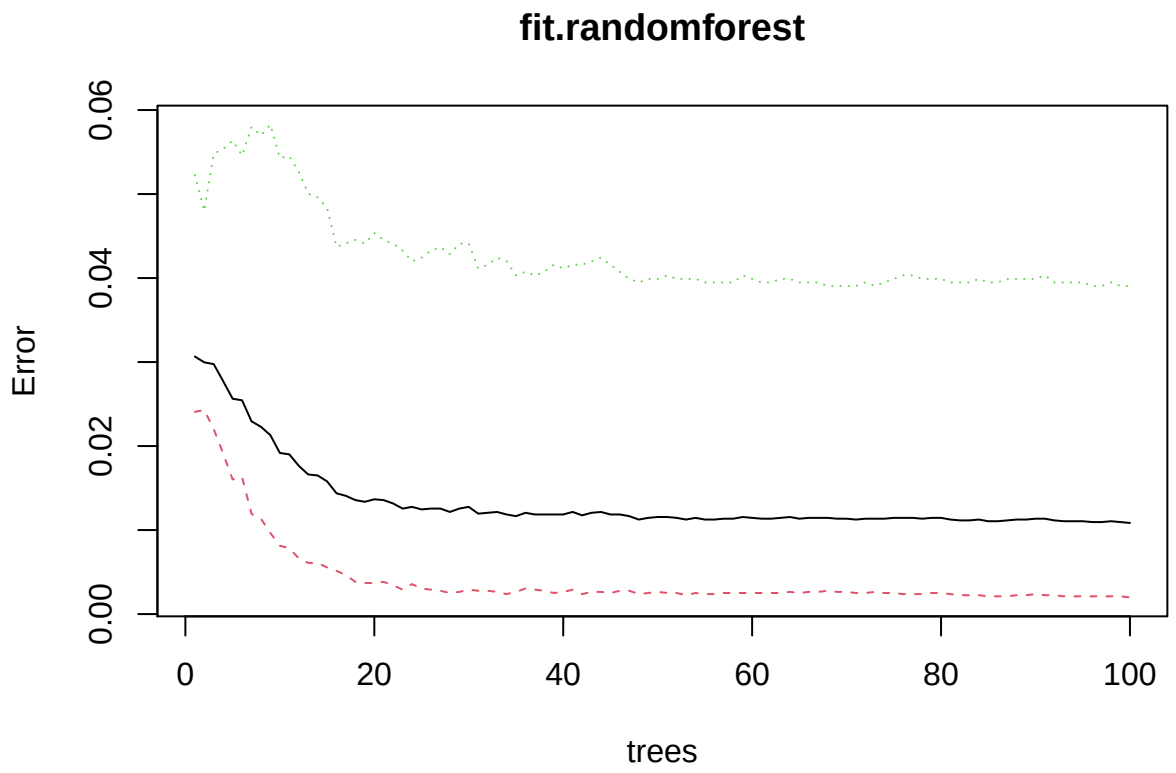
```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
set.seed(1)
#Clasificador tipo Random Forest
fit.randomforest <- randomForest(left ~ ., data=datos.trabajo, ntree=100, proximity=TRUE)
plot(fit.randomforest)
```



8) Utilizando los datos de validación, calcule:

a. La sensibilidad de cada procedimiento de clasificación, definida como el porcentaje de personas para las que el modelo predice que dejarán la compañía, dentro de todas aquellas que en realidad dejaron la compañía.

```
# Antes de calcular la sensibilidad, se crean las predicciones para cada modelo

# Predicción Árbol de Decisión
pred.tree <- predict(pfit, datos.validacion[,-7], type='class')

# Predicción Bayes Ingenuo
pred.nb <- predict(fitNB, datos.validacion[,-7], type="class")

# Predicción Bagging
pred.bagging <- predict.bagging(fit.bagging, newdata = datos.validacion)

# Predicción Boosting
pred.boosting <- predict.boosting(fit.boosting, newdata = datos.validacion)

# Predicción Random Forest
pred.randomforest <- predict(fit.randomforest, newdata = datos.validacion, type='prob')
```

```r
# Evaluación de los modelos con librería "rminer"
library(rminer)

# Sensibilidad o Recall: Tasa de Verdaderos Positivos
# Se especifica el número de clases (en este caso TC=2)
# Nos quedamos con el segundo valor, que representa el caso que definimos como positivo

# Sensibilidad modelo árbol de decisión
mmetric(datos.validacion[,7], pred.tree, "TPR", TC=2)
```

```
## [1] 94.91569 92.77311
```

```r
# Sensibilidad modelo Naïve Bayes
mmetric(datos.validacion[,7], pred.nb, "TPR", TC=2)
```

```
## [1] 81.68612 71.76471
```

```r
# Sensibilidad modelo Bagging
mmetric(datos.validacion[,7], pred.bagging$class, "TPR", TC=2)
```

```
## [1] 98.67704 92.01681
```

```r
# Sensibilidad modelo Boosting
mmetric(datos.validacion[,7], pred.boosting$class, "TPR", TC=2)
```

```
## [1] 99.04021 92.18487
```

```r
# Sensibilidad modelo Random Forest
mmetric(datos.validacion[,7], pred.randomforest, "TPR", TC=2)
```

```
## [1] 99.84436 97.89916
```

b. La especificidad de cada procedimiento de clasificación, definida como el porcentaje de personas para las que el modelo predice que no dejarán la compañía, dentro de todas aquellas que en realidad no dejaron la compañía.

```
# Especificidad: Tasa de Verdaderos Negativos
# Nos quedamos con el segundo valor, que representa el caso que definimos como positivo

# Especificidad modelo árbol de decisión
mmetric(datos.validacion[,7], pred.tree, "TNR", TC=2)
```

```
## [1] 92.77311 94.91569
```

```
# Especificidad modelo Naïve Bayes
mmetric(datos.validacion[,7], pred.nb, "TNR", TC=2)
```

```
## [1] 71.76471 81.68612
```

```
# Especificidad modelo Bagging
mmetric(datos.validacion[,7], pred.bagging$class, "TNR", TC=2)
```

```
## [1] 92.01681 98.67704
```

```
# Especificidad modelo Boosting
mmetric(datos.validacion[,7], pred.boosting$class, "TNR", TC=2)
```

```
## [1] 92.18487 99.04021
```

```
# Especificidad modelo Random Forest
mmetric(datos.validacion[,7], pred.randomforest, "TNR", TC=2)
```

```
## [1] 97.89916 99.84436
```

c. La exactitud de cada procedimiento de clasificación, definida como el porcentaje de personas clasificadas correctamente.

```
# Accuracy (exactitud)
# Capacidad global de clasificar correctamente a una observación, cualquiera sea su clase

# Exactitud modelo árbol de decisión
mmetric(datos.validacion[,7], pred.tree, "ACC")
```

```
## [1] 94.41031
```

```
# Exactitud modelo Naïve Bayes
mmetric(datos.validacion[,7], pred.nb, "ACC")
```

```
## [1] 79.34589
```

```
# Exactitud modelo Bagging
mmetric(datos.validacion[,7], pred.bagging$class, "ACC")
```

```
## [1] 97.10605
```

```
# Exactitud modelo Boosting
mmetric(datos.validacion[,7], pred.boosting$class, "ACC")
```

```
## [1] 97.42319
```

```
# Exactitud modelo Random Forest
mmetric(datos.validacion[,7], pred.randomforest, "ACC")
```

```
## [1] 99.38553
```

OTROS CÁLCULOS DE APOYO

```r
# Precision (precisión)
# Capacidad de no clasificar incorrectamente negativos como positivos.
# Nos quedamos con el segundo valor, que representa el caso que definimos como positivo

# Precisión modelo árbol de decisión
mmetric(datos.validacion[,7], pred.tree, "PRECISION")
```

```
## [1] 97.70360 84.92308
```

```r
# Precisión modelo Naïve Bayes
mmetric(datos.validacion[,7], pred.nb, "PRECISION")
```

```
## [1] 90.35868 54.74359
```

```r
# Precisión modelo Bagging
mmetric(datos.validacion[,7], pred.bagging$class, "PRECISION")
```

```
## [1] 97.56348 95.54974
```

```r
# Precisión modelo Boosting
mmetric(datos.validacion[,7], pred.boosting$class, "PRECISION")
```

```
## [1] 97.62209 96.73721
```

```r
# Precisión modelo Random Forest
mmetric(datos.validacion[,7], pred.randomforest, "PRECISION")
```

```
## [1] 99.35467 99.48762
```

```r
# Matriz de Confusión
library(caret)

# Matriz de Confusión para Árbol de Decisión
confusionMatrix(pred.tree, datos.validacion[,7], positive='1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3659   86
##          1  196 1104
##
##                Accuracy : 0.9441
##                  95% CI : (0.9374, 0.9503)
##     No Information Rate : 0.7641
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8497
##
##  Mcnemar's Test P-Value : 8.535e-11
##
##             Sensitivity : 0.9277
##             Specificity : 0.9492
##          Pos Pred Value : 0.8492
##          Neg Pred Value : 0.9770
##              Prevalence : 0.2359
##          Detection Rate : 0.2188
##    Detection Prevalence : 0.2577
##       Balanced Accuracy : 0.9384
##
##        'Positive' Class : 1
##
```

```r
# Matriz de Confusión para Naïve Bayes
confusionMatrix(pred.nb, datos.validacion[,7], positive='1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3149  336
##          1  706  854
##
##                Accuracy : 0.7935
##                  95% CI : (0.782, 0.8046)
##     No Information Rate : 0.7641
##     P-Value [Acc > NIR] : 3.471e-07
##
##                   Kappa : 0.4826
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
```

```
##             Sensitivity : 0.7176
##             Specificity : 0.8169
##          Pos Pred Value : 0.5474
##          Neg Pred Value : 0.9036
##              Prevalence : 0.2359
##          Detection Rate : 0.1693
##    Detection Prevalence : 0.3092
##       Balanced Accuracy : 0.7673
##
##        'Positive' Class : 1
##
```

```
# Matriz de Confusión para Bagging
confusionMatrix(table(pred.bagging$class, datos.validacion[,7]), positive='1')
```

```
## Confusion Matrix and Statistics
##
##
##        0    1
##   0 3804   95
##   1   51 1095
##
##                Accuracy : 0.9711
##                  95% CI : (0.9661, 0.9755)
##     No Information Rate : 0.7641
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9187
##
##  Mcnemar's Test P-Value : 0.0003727
##
##             Sensitivity : 0.9202
##             Specificity : 0.9868
##          Pos Pred Value : 0.9555
##          Neg Pred Value : 0.9756
##              Prevalence : 0.2359
##          Detection Rate : 0.2170
##    Detection Prevalence : 0.2272
##       Balanced Accuracy : 0.9535
##
##        'Positive' Class : 1
##
```

```
# Matriz de Confusión para Boosting
confusionMatrix(table(pred.boosting$class, datos.validacion[,7]), positive='1')
```

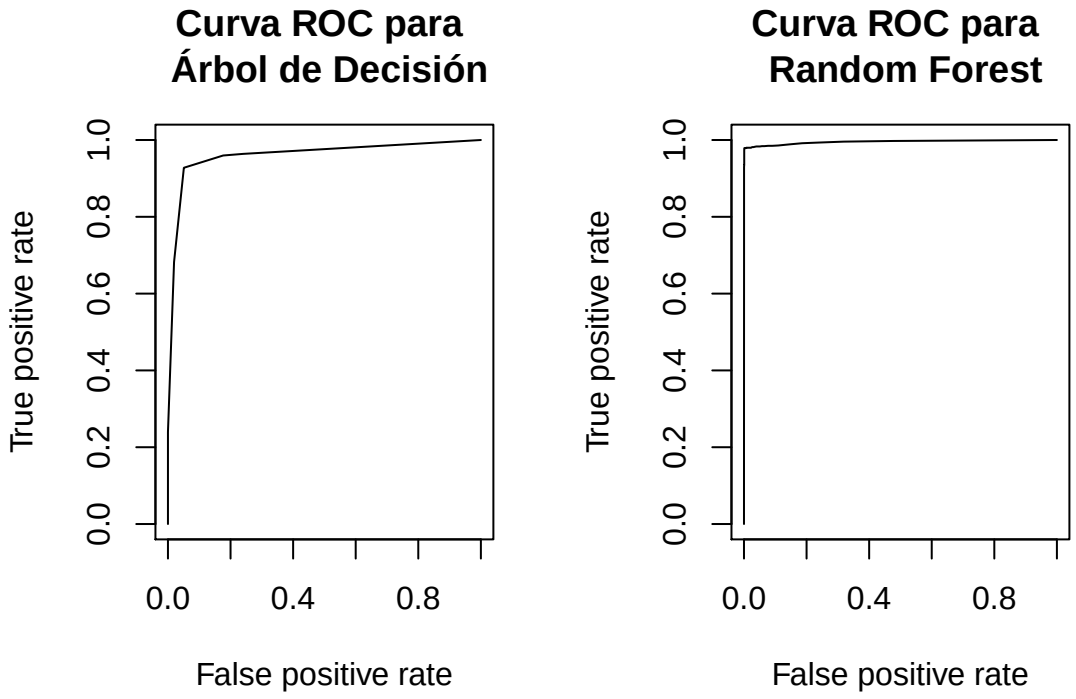
```
## Confusion Matrix and Statistics
##
##
##        0    1
##   0 3818   93
##   1   37 1097
##
```

```
##                Accuracy : 0.9742
##                  95% CI : (0.9695, 0.9784)
##     No Information Rate : 0.7641
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9273
##
##  Mcnemar's Test P-Value : 1.408e-06
##
##             Sensitivity : 0.9218
##             Specificity : 0.9904
##          Pos Pred Value : 0.9674
##          Neg Pred Value : 0.9762
##              Prevalence : 0.2359
##          Detection Rate : 0.2174
##    Detection Prevalence : 0.2248
##       Balanced Accuracy : 0.9561
##
##        'Positive' Class : 1
##
```

```
# Matriz de Confusión para Random Forest
confusionMatrix(predict(fit.randomforest, datos.validacion, type='response'),
                datos.validacion[,7], positive='1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3848   25
##          1    7 1165
##
##                Accuracy : 0.9937
##                  95% CI : (0.9911, 0.9957)
##     No Information Rate : 0.7641
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9823
##
##  Mcnemar's Test P-Value : 0.002654
##
##             Sensitivity : 0.9790
##             Specificity : 0.9982
##          Pos Pred Value : 0.9940
##          Neg Pred Value : 0.9935
##              Prevalence : 0.2359
##          Detection Rate : 0.2309
##    Detection Prevalence : 0.2323
##       Balanced Accuracy : 0.9886
##
##        'Positive' Class : 1
##
```

```r
# Curva ROC
# Representación gráfica de la sensibilidad frente a la especificidad para un
# sistema clasificador binario, según se varía el umbral de discriminación.

library(ROCR)
#Se agrupan los 2 gráficos en 1 fila y dos columnas
par(mfrow=c(1,2))

# ROC Árbol de Decisión
roc.tree <- predict(pfit, datos.validacion[,-7], type='prob')
pred1.tree <- prediction(roc.tree[,2], datos.validacion["left"])
perf1.tree <- performance(pred1.tree, "tpr", "fpr")
plot(perf1.tree, main = "Curva ROC para \n Árbol de Decisión")

# ROC Random Forest
pred1.randomforest <- prediction(pred.randomforest[,2], datos.validacion,"left"])
perf1.randomforest <- performance(pred1.randomforest, "tpr", "fpr")
plot(perf1.randomforest, main = "Curva ROC para \n Random Forest")
```




```r
# Mediante la comparación de ambas curvas podemos observar que Random Forest es
# un mejor sistema de clasificación respecto al Árbol de Decisión, ya que para
# cada valor modelado, sus ratios de sensibilidad y especificidad están más
# cercanos a 1.
```

9) Discuta sus resultados con sus compañeros mediante el foro. ¿Qué se puede concluir de esta actividad?

Análisis y principales conclusiones.

A partir del modelo de **árbol de decisión**, se puede determinar qué variables condicionan, mayormente, la decisión de un empleado para abandonar la empresa. Estas son:

- Nivel de satisfacción.
- Número de proyectos realizados.
- Tiempo de permanencia en la compañía.
- Puntaje en última evaluación laboral.

En el **árbol de decisión podado** se puede observar cómo interactúan estas variables.

Dejan la compañía:

- Quienes realizaron menos de 3 proyectos y cuyo nivel de satisfacción está bajo 0.47.
- Quienes hicieron más de 3 proyectos y cuyo nivel de satisfacción está por debajo de 0.12
- Quienes han pasado menos de 5 meses en la compañía y obtuvieron más de 0.81 puntos porcentuales en la última evaluación laboral.

Si bien no es de los modelos con mejor tasa de especificidad, exactitud y precisión, sí aporta claridad para determinar qué variables tienen más peso [1]. En este caso se pudo determinar que 5 de 9 variables no generaban un impacto en la decisión de abandonar. Estas son: horas promedio mensuales trabajadas, accidente laboral, ascenso en los últimos 5 años, área y salario.

Respecto al valor de CP=0.06, este nos garantiza una poda al árbol original, lo cual no ocurriría si hubiéramos conservado el valor por defecto de CP=0.01. Llama la atención que a un valor de CP más pequeño se gana sensibilidad, pero se pierde especificidad, exactitud y precisión. Además, se puede caer en sobreajuste.

Por otra parte, se observa que el modelo que mejor clasifica los datos es **Random Forest**, obteniendo los ratios con mejor desempeño de sensibilidad, especificidad, exactitud y precisión. Además, este método de clasificación no tiene problemas de *overfitting* (sobreajuste), y es más robusto frente a errores y *outliers* [2].

También se puede observar que este es el algoritmo que más tarda en ser entrenado, debido a la cantidad de árboles utilizados. Además, se comprobó que el error se mantiene estable a partir de 20 árboles en adelante (los ratios de sensibilidad y especificidad fueron muy parecidos a cuando el número de árboles era igual a 100).

[1] [Árboles de Predicción, https://rpubs.com/Joaquin_AR/255596, Ventajas y Desventajas]

[2] [Parte 2: Métodos de Clasificación, Random Forest, Diapositiva 78]