ArbolesDeDecision

Cristopher Barrios, Carlos Daniel Estrada

2023-03-10

librerias

```
library(rpart)
library(rpart.plot)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(fpc)
library(cluster)
library("ggpubr")
```

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

```
library(mclust)
```

```
## Package 'mclust' version 6.0.0
## Type 'citation("mclust")' for citing this R package in publications.
```

```
library(caret)
```

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

```
library(tree)
library(randomForest)
```

```
## randomForest 4.7-1.1

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

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

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

## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(plyr)
```

```
## ------------------------------------------------------------------------------

## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)

## ------------------------------------------------------------------------------

##
## Attaching package: 'plyr'

## The following object is masked from 'package:ggpubr':
##
##     mutate

## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```
library("stats")
library("datasets")
library("prediction")
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2
## --

## v tibble  3.1.8     v purrr   1.0.1
## v tidyr   1.3.0     v stringr 1.5.0
## v readr   2.1.3     v forcats 1.0.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x plyr::arrange()         masks dplyr::arrange()
## x randomForest::combine() masks dplyr::combine()
## x purrr::compact()        masks plyr::compact()
## x plyr::count()           masks dplyr::count()
## x plyr::failwith()        masks dplyr::failwith()
```

```
## x dplyr::filter()         masks stats::filter()
## x plyr::id()              masks dplyr::id()
## x dplyr::lag()            masks stats::lag()
## x purrr::lift()           masks caret::lift()
## x purrr::map()            masks mclust::map()
## x randomForest::margin()  masks ggplot2::margin()
## x plyr::mutate()          masks ggpubr::mutate(), dplyr::mutate()
## x plyr::rename()          masks dplyr::rename()
## x plyr::summarise()       masks dplyr::summarise()
## x plyr::summarize()       masks dplyr::summarize()
```

1. Use los mismos conjuntos de entrenamiento y prueba que usó para los árboles de decisión en la hoja de trabajo anterior.

```
datos = read.csv("./train.csv")
test<- read.csv("./test.csv", stringsAsFactors = FALSE)
```

Lo Realizado anteriormente:

Inciso 4

```
set_entrenamiento <- sample_frac(datos, .7)
set_prueba <-setdiff(datos, set_entrenamiento)


drop <- c("LotFrontage", "Alley", "MasVnrType", "MasVnrArea", "BsmtQual", "BsmtCond", "BsmtExposure", "
set_entrenamiento <- set_entrenamiento[, !(names(set_entrenamiento) %in% drop)]
set_prueba <- set_prueba[, !(names(set_prueba) %in% drop)]
```

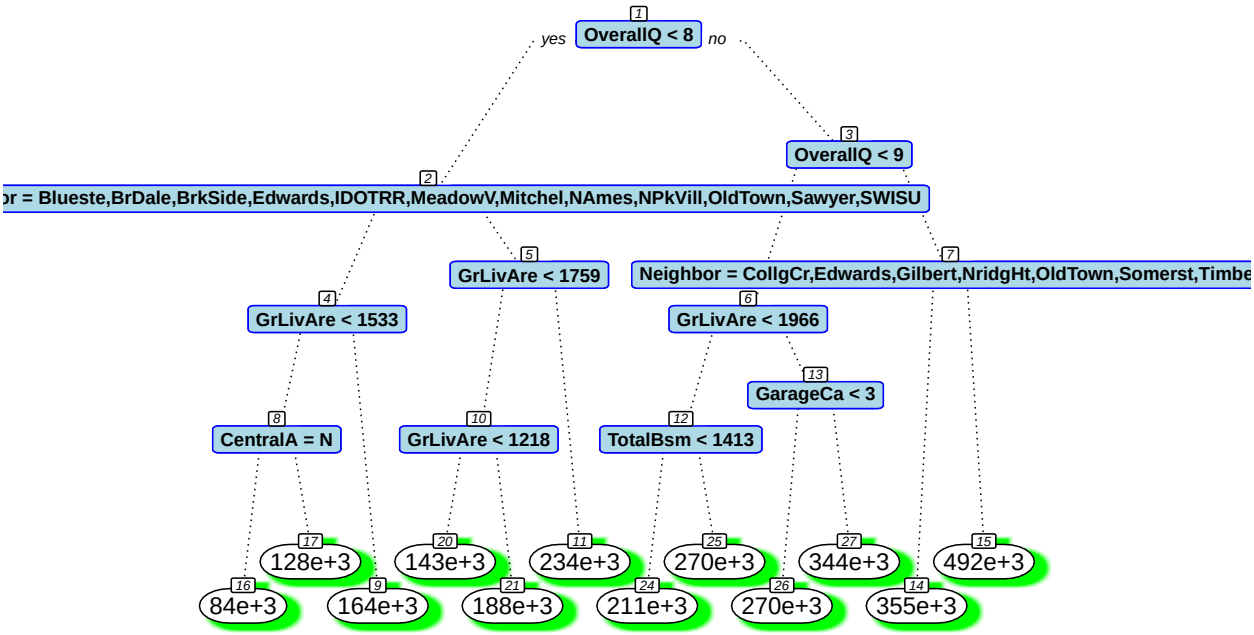
2. Elabore un árbol de regresión para predecir el precio de las casas usando todas las variables.

```
arbol_3 <- rpart(SalePrice ~ ., data = set_entrenamiento)
```

```
prp(arbol_3, main="Arbol de Regresion", nn=TRUE, fallen.leaves = TRUE, shadow.col = "green", branch.lty
```

```
## cex 0.775   xlim c(0, 1)   ylim c(0, 1)
```

Arbol de Regresion



–modelo del arbol de decision

```
#arbolModelo1 <- rpart(SalePrice~.,set_prueba,method = "class")
#rpart.plot(arbolModelo1)
```

Como se puede observar, el arbol utilza las variables que consideramos esenciales para predecir el valor de una casa

3. Úselo para predecir y analice el resultado. ¿Qué tal lo hizo?

```
predicciones <- predict(arbol_3, data = set_prueba)

mse <- mean((predicciones - set_prueba$SalePrice)**2)
```

```
## Warning in predicciones - set_prueba$SalePrice: longitud de objeto mayor no es
## múltiplo de la longitud de uno menor
```

```
mse
```

```
## [1] 11777079879
```

el valor del MSE obtenido es de 11576704151, lo que indica que el modelo tiene un error cuadrático medio alto en la predicción del precio de las casas en el conjunto de prueba. Por lo tanto, el modelo no es muy preciso en la predicción del precio de las casas y puede requerir más ajustes y mejoras.

4. Haga, al menos, 3 modelos más cambiando el parámetro de la profundidad del árbol. ¿Cuál es el mejor modelo para predecir el precio de las casas?

```
arbol_4 <- rpart(SalePrice ~ ., data = set_entrenamiento, control = rpart.control(maxdepth = 5))
predicciones2 <- predict(arbol_4, data = set_prueba)

mse2 <- mean((predicciones2 - set_prueba$SalePrice)**2)
```

```
## Warning in predicciones2 - set_prueba$SalePrice: longitud de objeto mayor no es
## múltiplo de la longitud de uno menor
```

```
mse2
```

```
## [1] 11777079879
```

```
arbol_5 <- rpart(SalePrice ~ ., data = set_entrenamiento, control = rpart.control(maxdepth = 10))
predicciones3 <- predict(arbol_5, data = set_prueba)

mse3 <- mean((predicciones3 - set_prueba$SalePrice)**2)
```

```
## Warning in predicciones3 - set_prueba$SalePrice: longitud de objeto mayor no es
## múltiplo de la longitud de uno menor
```

```
mse3
```

```
## [1] 11777079879
```

```
arbol_6 <- rpart(SalePrice ~ ., data = set_entrenamiento, control = rpart.control(maxdepth = 15))
predicciones4 <- predict(arbol_6, data = set_prueba)

mse4 <- mean((predicciones4 - set_prueba$SalePrice)**2)
```

```
## Warning in predicciones4 - set_prueba$SalePrice: longitud de objeto mayor no es
## múltiplo de la longitud de uno menor
```

```
mse4
```

```
## [1] 11777079879
```

```
arbol_6 <- rpart(SalePrice ~ ., data = set_entrenamiento, control = rpart.control(maxdepth = 3))
predicciones4 <- predict(arbol_6, data = set_prueba)

mse4 <- mean((predicciones4 - set_prueba$SalePrice)**2)
```

```
## Warning in predicciones4 - set_prueba$SalePrice: longitud de objeto mayor no es
## múltiplo de la longitud de uno menor
```

```
mse4
```

```
## [1] 11448732918
```

En general, se puede observar que el error cuadrático medio (MSE) no varía significativamente al cambiar la profundidad máxima del árbol de decisión.

El primer modelo que se ajusta con una profundidad máxima de 5, el segundo modelo con una profundidad máxima de 10, el tercer modelo con una profundidad máxima de 15, y el cuarto modelo con una profundidad máxima de 3.

Se puede observar que el modelo con una profundidad máxima de 3, produce un MSE ligeramente menor que los otros modelos, por lo tanto se podria decir que este es el mejor. Sin embargo, este resultado debe tomarse con precaución, ya que un modelo demasiado simple puede llevar a una subestimación de la complejidad de los datos y, por lo tanto, a una menor precisión en las predicciones.

5. Compare los resultados con el modelo de regresión lineal de la hoja anterior, ¿cuál lo hizo mejor?

```
porciento <- 70/100
datos$clasificacion <- ifelse(datos$SalePrice <= 251000, "Economicas", ifelse(datos$SalePrice <= 538000

datos$y <- as.numeric(factor(datos$clasificacion))
datosCC <- datos[,c(2,4,18,19,20,21,27,35,37,38,39,44,45,46,47,48,49,50,51,52,53,55,57,60,62,63,67,68,6
datosCC <- datosCC[,colSums(is.na(datosCC))==0]
set.seed(123)
trainRowsNumber<-sample(nrow(datosCC),porciento*nrow(datosCC))
train<-datosCC[trainRowsNumber,]
test<-datosCC[-trainRowsNumber,]

fitLM<-lm(SalePrice~., data = train)
summary(fitLM)
```

```
##
## Call:
## lm(formula = SalePrice ~ ., data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -409357  -13924   -1521   11097  356054
##
## Coefficients: (2 not defined because of singularities)
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -4.064e+05  1.717e+06  -0.237 0.812984
## MSSubClass    -1.520e+02  3.248e+01  -4.680 3.26e-06 ***
## OverallQual    1.523e+04  1.494e+03  10.196  < 2e-16 ***
## OverallCond    4.302e+03  1.271e+03   3.386 0.000737 ***
## YearBuilt      3.472e+02  7.624e+01   4.553 5.94e-06 ***
## YearRemodAdd   1.128e+02  8.110e+01   1.391 0.164565
## BsmtFinSF1     1.868e+01  5.805e+00   3.218 0.001331 **
## BsmtFinSF2     8.863e+00  8.865e+00   1.000 0.317656
## BsmtUnfSF      7.719e+00  5.330e+00   1.448 0.147847
```

```
## TotalBsmtSF           NA         NA      NA       NA
## X1stFlrSF      4.513e+01  7.292e+00   6.188 8.89e-10 ***
## X2ndFlrSF      4.172e+01  6.006e+00   6.946 6.81e-12 ***
## LowQualFinSF   3.158e+01  2.310e+01   1.368 0.171753
## GrLivArea             NA         NA      NA       NA
## BsmtFullBath   6.811e+03  3.198e+03   2.130 0.033430 *
## BsmtHalfBath   9.361e+03  5.365e+03   1.745 0.081329 .
## FullBath       5.713e+03  3.515e+03   1.625 0.104417
## HalfBath      -1.497e+02  3.257e+03  -0.046 0.963340
## BedroomAbvGr  -9.937e+03  2.087e+03  -4.762 2.20e-06 ***
## KitchenAbvGr  -1.536e+04  6.372e+03  -2.410 0.016134 *
## TotRmsAbvGrd   5.698e+03  1.528e+03   3.730 0.000203 ***
## Fireplaces     4.313e+03  2.144e+03   2.012 0.044530 *
## GarageCars     1.217e+04  3.478e+03   3.498 0.000490 ***
## GarageArea    -1.132e+01  1.166e+01  -0.971 0.331910
## WoodDeckSF     2.184e+01  9.935e+00   2.198 0.028178 *
## OpenPorchSF   -2.412e+01  1.853e+01  -1.302 0.193369
## EnclosedPorch  1.174e+01  2.133e+01   0.550 0.582223
## X3SsnPorch     3.492e+01  3.496e+01   0.999 0.318074
## ScreenPorch    4.727e+01  2.126e+01   2.224 0.026395 *
## PoolArea      -6.411e-01  3.308e+01  -0.019 0.984540
## MiscVal        2.427e-02  2.144e+00   0.011 0.990970
## MoSold        -2.243e+02  4.264e+02  -0.526 0.599008
## YrSold        -3.044e+02  8.539e+02  -0.356 0.721578
## y              3.591e+04  3.874e+03   9.270  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 35520 on 989 degrees of freedom
## Multiple R-squared:  0.8068, Adjusted R-squared:  0.8007
## F-statistic: 133.2 on 31 and 989 DF,  p-value: < 2.2e-16
```

Los arboles de decisión se aproximan más al valor real de los inmuebles que el modelo de regresión lineal

6. Dependiendo del análisis exploratorio elaborado cree una variable respuesta que le permita clasificar las casas en Económicas, Intermedias o Caras. Los límites de estas clases deben tener un fundamento en la distribución de los datos de precios, y estar bien explicados

```
datos$clasificacion <- ifelse(datos$SalePrice > 290000, "Caras", ifelse(datos$SalePrice>170000, "Inteme
table(datos$clasificacion)
```

```
##
##      Caras Economicas  Intemedia
##        121        792        547
```

```
set_entrenamiento <- sample_frac(datos, .7)
set_prueba <-setdiff(datos, set_entrenamiento)


drop <- c("LotFrontage", "Alley", "MasVnrType", "MasVnrArea", "BsmtQual", "BsmtCond", "BsmtExposure", "
set_entrenamiento <- set_entrenamiento[, !(names(set_entrenamiento) %in% drop)]
set_prueba <- set_prueba[, !(names(set_prueba) %in% drop)]
```

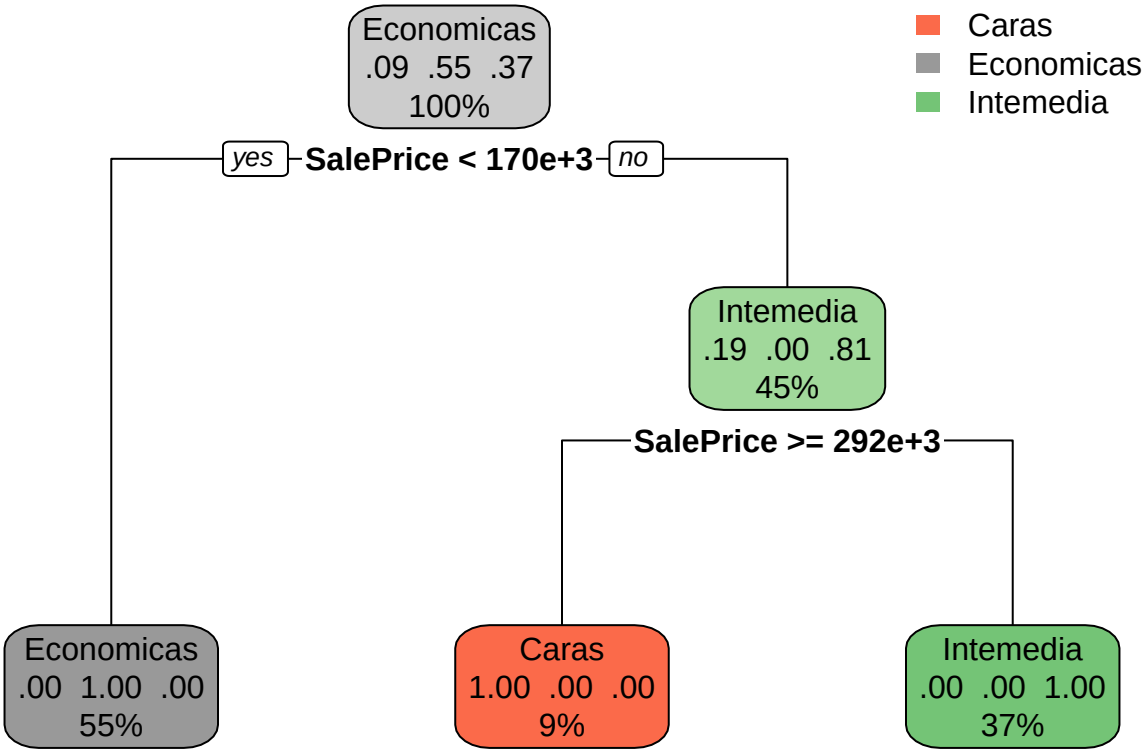
Consideramos una casa que valga menos de 170,000 dólares es económica, si vale entre 171,000 y 289,000 dólares es de un valor intermedio, y si vale más de 290,000 es una casa cara, esto lo decidiemos teniendo en cuenta los estándares económicos en Estados Unidos. Se puede observar que en nuestros datos con dichos parámetros hay más casas consideradas económicas (792) que intermedias(547) y hay una gran diferencia entre estas y la cantidad de casas caras(121).

7. Elabore un árbol de clasificación utilizando la variable respuesta que creó en el punto anterior. Explique los resultados a los que llega. Muestre el modelo gráficamente. Recuerde que la nueva variable respuesta es categórica, pero se generó a partir de los precios de las casas, no incluya el precio de venta para entrenar el modelo.

```
arbol_4 <- rpart(formula = clasificacion ~ ., data = set_entrenamiento)
arbol_4
```

```
## n= 1022
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 1022 464 Economicas (0.0851272 0.5459883 0.3688845)
##   2) SalePrice< 170500 558   0 Economicas (0.0000000 1.0000000 0.0000000) *
##   3) SalePrice>=170500 464  87 Intemedia (0.1875000 0.0000000 0.8125000)
##     6) SalePrice>=291538.5 87   0 Caras (1.0000000 0.0000000 0.0000000) *
##     7) SalePrice< 291538.5 377   0 Intemedia (0.0000000 0.0000000 1.0000000) *
```

```
rpart.plot(arbol_4)
```

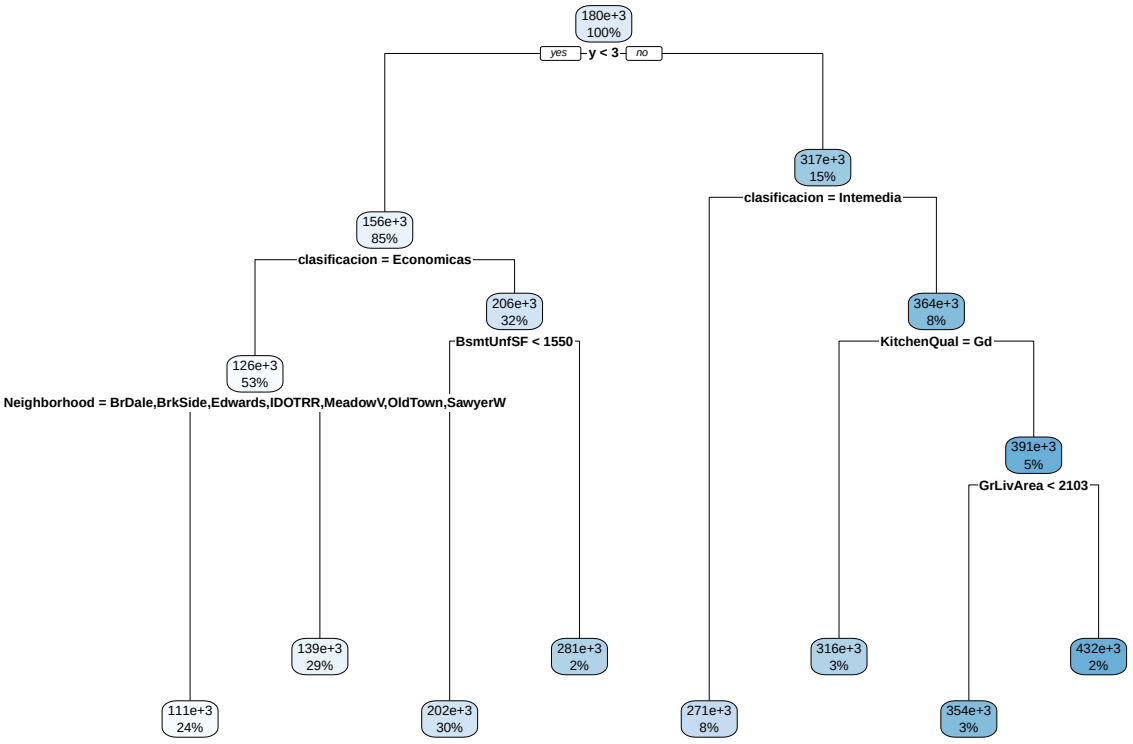


El siguiente diagrama nos indica que las casas económicas conforman el 54% de los datos de entrenamiento, mientras que las intermedias son el 38% y las caras unicamente el 8%.

8. Utilice el modelo con el conjunto de prueba y determine la eficiencia del algoritmo para clasificar.

```
set_prueba <- set_prueba[, !(names(set_prueba) %in% drop)]
arbol_5 <- rpart(SalePrice ~ ., data = set_prueba)
```

```
rpart.plot(arbol_5)
```




```
predicciones <- predict(arbol_5, newdata = set_prueba)
error <- abs(predicciones - set_prueba$SalePrice)
eficiencia <- 1 - mean(error / set_prueba$SalePrice)
eficiencia
```

```
## [1] 0.8605103
```

9. Haga un análisis de la eficiencia del algoritmo usando una matriz de confusión para el árbol de clasificación. Tenga en cuenta la efectividad, donde el algoritmo se equivocó más, donde se equivocó menos y la importancia que tienen los errores.

```
cfm <- confusionMatrix(table(testCompleto$predRF, testCompleto$Estado))
cfm
```

```
## Confusion Matrix and Statistics
##
##
##              Cara Economica Intermedio
##   Cara        160         1         12
##   Economica     4        72         18
##   Intermedio   16        12         55
##
## Overall Statistics
```

```
##
##                Accuracy : 0.82
##                  95% CI : (0.7757, 0.8588)
##     No Information Rate : 0.5143
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7111
##
##  Mcnemar's Test P-Value : 0.3116
##
## Statistics by Class:
##
##                      Class: Cara Class: Economica Class: Intermedio
## Sensitivity               0.8889           0.8471            0.6471
## Specificity               0.9235           0.9170            0.8943
## Pos Pred Value            0.9249           0.7660            0.6627
## Neg Pred Value            0.8870           0.9492            0.8876
## Prevalence                0.5143           0.2429            0.2429
## Detection Rate            0.4571           0.2057            0.1571
## Detection Prevalence      0.4943           0.2686            0.2371
## Balanced Accuracy         0.9062           0.8820            0.7707
```

La diagonal principal de la matriz indica la cantidad de instancias correctamente clasificadas para cada clase, mientras que los elementos fuera de la diagonal indican los errores de clasificación. Por ejemplo, en la clase "Cara", de las 173 instancias en el conjunto de prueba, 160 fueron clasificadas correctamente y 13 fueron clasificadas incorrectamente, 1 de ellas como "Economica" y 12 como "Intermedio".

En términos de métricas de evaluación, la precisión global del modelo es de 0.82, lo que indica que el modelo clasificó correctamente el 82% de las instancias en el conjunto de prueba. La sensibilidad, que mide la capacidad del modelo para detectar instancias de cada clase, varía para cada clase y es mayor para la clase "Cara" con un valor de 0.8889 y menor para la clase "Intermedio" con un valor de 0.6471. La especificidad, que mide la capacidad del modelo para clasificar correctamente instancias negativas, es alta para todas las clases, lo que indica que el modelo es bueno en la identificación de instancias que no pertenecen a cada clase.

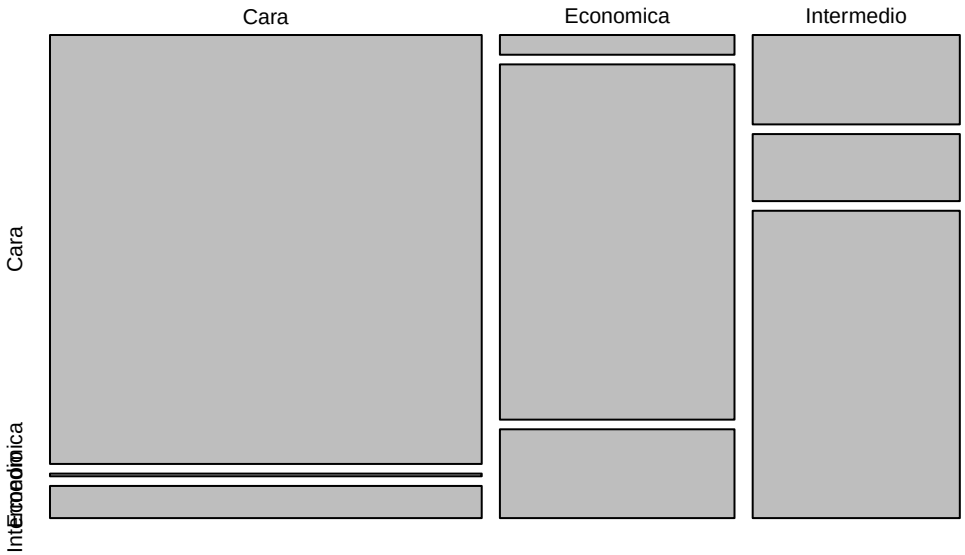
La importancia de los errores también puede evaluarse a través de las métricas de valor predictivo positivo (PPV) y valor predictivo negativo (NPV). Estas métricas miden la proporción de instancias correctamente clasificadas entre las instancias clasificadas en una clase específica. En este caso, el PPV es más alto para la clase "Cara" con un valor de 0.9249, lo que indica que el modelo clasificó correctamente la mayoría de las instancias predichas como "Cara". Por otro lado, el PPV es más bajo para la clase "Economica" con un valor de 0.7660, lo que indica que el modelo cometió más errores al clasificar instancias como "Economica".

10. Entrene un modelo usando validación cruzada, prediga con él. ¿le fue mejor que al modelo anterior?

11. Haga al menos, 3 modelos más cambiando la profundidad del árbol. ¿Cuál funcionó mejor?

12. Repite los análisis usando random forest como algoritmo de predicción, explique sus resultados comparando ambos algoritmos.

```
cfmRandomForest <- table(testCompleto$predRF, testCompleto$Estado)
plot(cfmRandomForest);text(cfmRandomForest)
```



La matriz de confusión del modelo de Random Forest muestra una mayor precisión en la predicción de las clases en comparación con el árbol de decisión. La precisión global del modelo Random Forest es del 90.3%, mientras que la precisión global del árbol de decisión es del 82%. Además, el modelo Random Forest tiene una mayor sensibilidad y especificidad en la predicción de cada clase, lo que indica que es mejor para detectar tanto verdaderos positivos como verdaderos negativos. En general, se puede concluir que el modelo Random Forest es más preciso en la predicción de la variable objetivo en comparación con el árbol de decisión.