TFM - Kaggle House Prices: Advanced Regression Techniques with caret

04 Creación de modelos predictivos con caret

Juan Carlos Santiago Culebras

2019-09-22

En esta fase aplicaremos distintos algoritmos de machine learning para generar modelos de regresión, que sean capaces de predecir la variable objetivo (SalePrice).

Existen multitud de algoritmos ya implementados para entrenar modelos de regresión, el paquete Caret simplifica la llamada ofreciendo un interfaz único.

Primeros pasos

Librerías

Realizamos la carga de las librerías necesarias

```
if(!is.element("dplyr", installed.packages()[, 1]))
      install.packages("dplyr", repos = 'http://cran.us.r-project.org')
library(dplyr)

if(!is.element("tidyr", installed.packages()[, 1]))
      install.packages("tidyr", repos = 'http://cran.us.r-project.org')
library(tidyr)

if(!is.element("ggplot2", installed.packages()[, 1]))
      install.packages("ggplot2", repos = 'http://cran.us.r-project.org')
library(ggplot2)

if(!is.element("grid", installed.packages()[, 1]))
      install.packages("grid", repos = 'http://cran.us.r-project.org')
library(grid)

if(!is.element("gridExtra", installed.packages()[, 1]))
      install.packages("gridExtra", repos = 'http://cran.us.r-project.org')
library(gridExtra)

if(!is.element("ggpubr", installed.packages()[, 1]))
      install.packages("ggpubr", repos = 'http://cran.us.r-project.org')
library(ggpubr)

if(!is.element("tibble", installed.packages()[, 1]))
      install.packages("tibble", repos = 'http://cran.us.r-project.org')
library(tibble)

if(!is.element("randomForest", installed.packages()[, 1]))
```

```r
      install.packages("randomForest", repos = 'http://cran.us.r-project.org')
library(randomForest)

if(!is.element("recipes", installed.packages()[, 1]))
      install.packages("recipes", repos = 'http://cran.us.r-project.org')
library(recipes)

if(!is.element("caret", installed.packages()[, 1]))
      install.packages("caret", repos = 'http://cran.us.r-project.org')
library(caret)

if(!is.element("kernlab", installed.packages()[, 1]))
      install.packages("kernlab", repos = 'http://cran.us.r-project.org')
library(kernlab)

if(!is.element("ranger", installed.packages()[, 1]))
      install.packages("ranger", repos = 'http://cran.us.r-project.org')
library(ranger)

if(!is.element("gbm", installed.packages()[, 1]))
      install.packages("gbm", repos = 'http://cran.us.r-project.org')
library(gbm)

if(!is.element("e1071", installed.packages()[, 1]))
      install.packages("e1071", repos = 'http://cran.us.r-project.org')
library(e1071)

if(!is.element("elasticnet", installed.packages()[, 1]))
      install.packages("elasticnet", repos = 'http://cran.us.r-project.org')
library(elasticnet)

if(!is.element("xgboost", installed.packages()[, 1]))
      install.packages("xgboost", repos = 'http://cran.us.r-project.org')
library(xgboost)

if(!is.element("glmnet", installed.packages()[, 1]))
      install.packages("glmnet", repos = 'http://cran.us.r-project.org')
library(glmnet)
```

Funciones

```r
fnEstudioModelo <- function ( modelo , estudioParam = TRUE){

  # modelo
  # modelo$finalModel

  p1 <- ggplot(data = modelo$resample, aes(x = RMSE)) +
        geom_density(alpha = 0.5, fill = "gray50") +
        geom_vline(xintercept = mean(modelo$resample$RMSE),
                   linetype = "dashed") +
        theme_bw()
```

```
  p2 <- ggplot(data = modelo$resample, aes(x = 1, y = RMSE)) +
        geom_boxplot(outlier.shape = NA, alpha = 0.5, fill = "gray50") +
        geom_jitter(width = 0.05) +
        labs(x = "") +
        theme_bw() +
        theme(axis.text.x = element_blank(), axis.ticks.x = element_blank())

  #Estudio de hiperparámtros
  if (estudioParam){
    p3 <- plot(modelo)
  }

  # Error de test
  predicciones <- predict(modelo
                          , newdata = dsTrain.CV
                          , type = "raw")

  # RMSE(predicciones, dsTrain.CV$SalePrice)
  # MAE(predicciones, dsTrain.CV$SalePrice)
  # R2(predicciones, dsTrain.CV$SalePrice, form = "traditional")


  t1 <- capture.output(summary(modelo$resample$RMSE, digits=3))
  t1 <- paste("Summary resample$RMSE", " ", paste(t1, collapse="\n"), sep = "\n")
  t1 <- text_grob(t1, size = 10)

  t2 <- capture.output(postResample(pred = predicciones, obs = dsTrain.CV$SalePrice))
  t2 <- paste("Error de test", " ", paste(t2, collapse="\n"), sep = "\n")
  t2 <- text_grob(t2, size = 10)

  t3 <- capture.output(modelo$finalModel)
  t3 <- text_grob(paste(t3, collapse="\n"), size = 9)

  grid.arrange(t3, top="Modelo final")
  grid.arrange(p1, p2, t1, t2, nrow = 2, top="RMSE obtenido en la validación")

  if (estudioParam){
    grid.arrange(p3, nrow = 1, top="Evolución del RMSE del modelo en función de hiperparámetros")
  }

}
```

Cargamos datos

Partimos de un dataset, construido en las etapas anteriores, con los datos ya preparados y que contiene solo las variables predictivas.

```
# Conjunto seleccionado en paso anterior

strOrigenF2 <- 'F02_03_dsDataAll_Recipe'
strOrigenF3 <- 'F03_11_dsDataSelVar_rfe_MejorRendimiento_top18'
```

```r
file <- paste('./F03_SelPredictores/',strOrigenF2,'/',strOrigenF3,'.RData',sep='')

load(file)

dirSalida <- paste('./F04_Modelos/',strOrigenF2,sep='')

if (!file.exists(dirSalida)){
    dir.create(file.path(dirSalida))
}

dirSalida <- paste('./F04_Modelos/',strOrigenF2,'/',strOrigenF3,sep='')

if (!file.exists(dirSalida)){
    dir.create(file.path(dirSalida))
}

rm(file)
```

Lectura de modelos ya entrenados si se realiza es estudio posteriormente

```r
# dir <- './F04_Modelos/F02_03_dsDataAll_Recipe/F03_15_dsDataSelVar_Completo/'
# load(paste(dir,'modelo_gbm.RData',sep=''))
# load(paste(dir,'modelo_glm.RData',sep=''))
# load(paste(dir,'modelo_glmnet.RData',sep=''))
# load(paste(dir,'modelo_Knn.RData',sep=''))
# load(paste(dir,'modelo_lasso.RData',sep=''))
# load(paste(dir,'modelo_lm.RData',sep=''))
# load(paste(dir,'modelo_rf.RData',sep=''))
# load(paste(dir,'modelo_svmlineal.RData',sep=''))
# load(paste(dir,'modelo_svmRadial.RData',sep=''))
# load(paste(dir,'modelo_XGBoost.RData',sep=''))
```

Separamos los datos

Optenemos 4 dataset

dsTrain - Que a su vez se divide en dsTrain.training dsTrain.CV

dsTest

```r
dsTrain <- dsDataAllVarSel %>%
  filter(indTrain == 1) %>%
  select(SalePrice, everything()) %>%
  select(-c(Id,indTrain))

dim(dsTrain)
```

```
## [1] 1458   19
```

```r
set.seed(123)
iTrain  <- createDataPartition(y=dsTrain$SalePrice, p=0.7, list=F)
```

```
dsTrain.training <- dsTrain[iTrain, ]
dsTrain.CV       <- dsTrain[-iTrain, ]

dsTest <- dsDataAllVarSel %>%
  filter(indTrain == 0) %>%
  select(SalePrice, everything())
```

Modelos

En esta sección se entrenarán distintos modelos para evaluar cual puede ser el mejor.

Sobre cada modelo se realizará:

- Entrenamiento
- Ajuste de hiperparámetros
- Evaluación mediante validación cruzada

Definimos tipo de entrenamiento

```
particiones  <- 5
repeticiones <- 5

# Entrenamiento con conjunto de hiperparametros
fitControl <- trainControl(method = "repeatedcv",
                              number = particiones,
                              repeats = repeticiones,
                              returnResamp = "final",
                              verboseIter = FALSE)
```

Regresión lineal

En estos modelos se busca una función con los predictores como variables y una combinación de pesos que multiplicados por las variables den como resultado un modelo para la variable objetivo.

Estos algoritmos son muy rápidos y responden bien cuando el número de predictores es alto.

En nuestro caso hemos seleccionado dos ejemplos

Linear Regression

Regresión lineal, este modelo es el más sencillo de probar y me ha servido como línea base para ir evaluando el resto de los modelos

```
t <- proc.time() # Inicia el cronómetro
modelo_lm <- train(SalePrice ~ .
                          , data = dsTrain.training
                          , method = "lm"
                          #, tuneGrid = hiperparametros
                          , tuneLength = 10
                          , metric = "RMSE"
                          , trControl = fitControl)
proc.time()-t    # Detiene el cronómetro
```

```
##    user  system elapsed
##    0.95    0.02    0.97
```

```r
# Guardo resultado del calculo
fileOuput <- paste(dirSalida,'/','modelo_lm','.RData',sep='')
save(modelo_lm, file = fileOuput)

modelo_lm
```

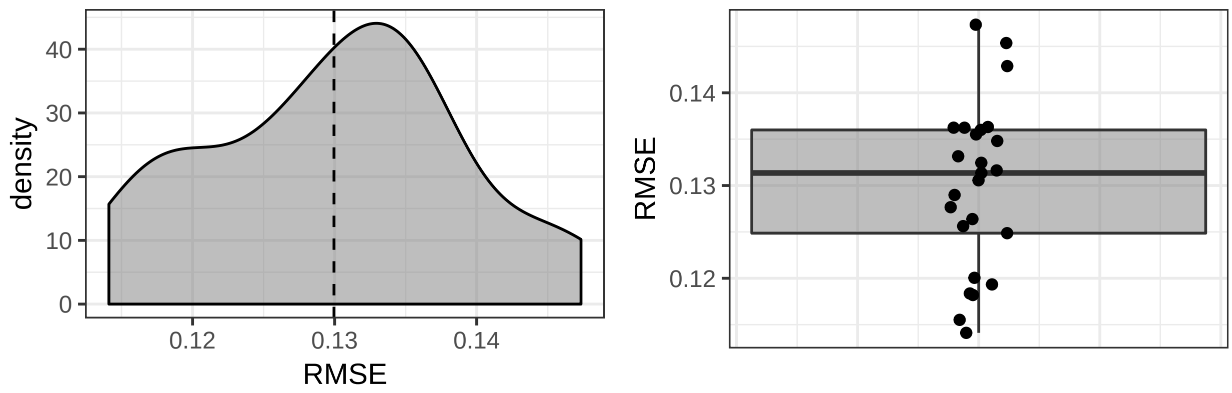
```
## Linear Regression
##
## 1023 samples
##   18 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 818, 820, 817, 819, 818, 819, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.1299566  0.8976907  0.0931044
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```r
# Presento estudio
fnEstudioModelo(modelo_lm, estudioParam = FALSE)
```

Modelo final

```
Call:
lm(formula = .outcome ~ ., data = dat)

Coefficients:
 (Intercept)    GrLivArea  OverallQual  TotalBsmtSF    X1stFlrSF
  12.0216986    0.1191885    0.0953092    0.0484734    0.0036513
  BsmtFinSF1      LotArea   GarageArea    X2ndFlrSF  OverallCond
   0.0326487    0.0367879    0.0168238    0.0065776    0.0578678
 FireplaceQu    YearBuilt  KitchenQual YearRemodAdd   GarageCars
   0.0243120    0.0822681    0.0190883    0.0255445    0.0323463
   ExterQual BsmtFinType1 TotRmsAbvGrd   Fireplaces
   0.0121054    0.0093991   -0.0008169    0.0119496
```

RMSE obtenido en la validación



Summary resample$RMSE

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.114	0.125	0.131	0.130	0.136	0.147

Error de test

RMSE	Rsquared	MAE
0.12362414	0.89944770	0.08982126

Generalized Linear Model

```
hiperparametros <- data.frame(parameter = "none")

t <- proc.time() # Inicia el cronómetro
modelo_glm <- train(SalePrice ~ .
                           , data = dsTrain.training
                           , method = "glm"
                           , tuneGrid = hiperparametros
                           , metric = "RMSE"
                           , trControl = fitControl)
proc.time()-t     # Detiene el cronómetro
```

```
##    user  system elapsed
##    0.99    0.03    1.01
```

```
# Guardo resultado del calculo
fileOuput <- paste(dirSalida,'/','modelo_glm','.RData',sep='')
save(modelo_glm, file = fileOuput)

modelo_glm
```

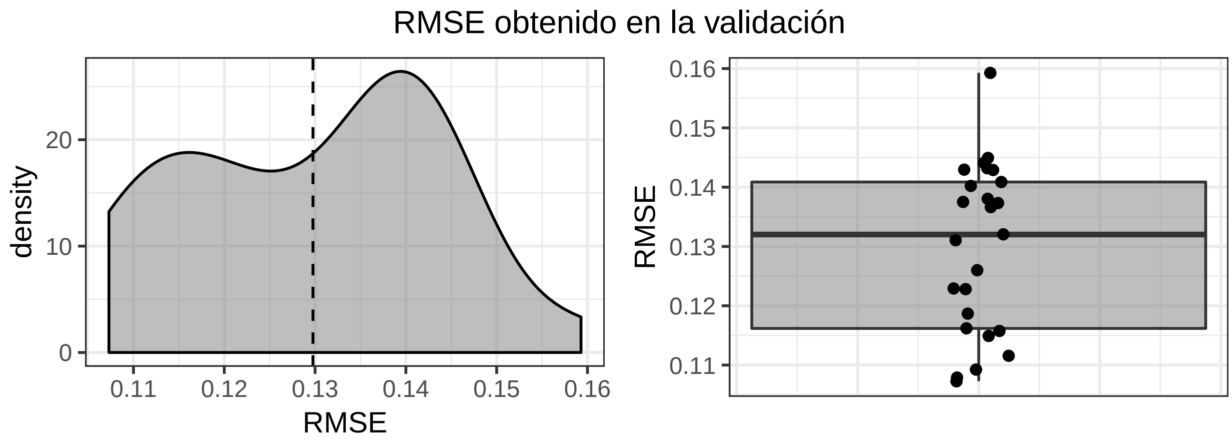
```
## Generalized Linear Model
```

```
##
## 1023 samples
##   18 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 819, 818, 819, 819, 817, 819, ...
## Resampling results:
##
##   RMSE       Rsquared  MAE
##   0.1297683  0.898214  0.09318374
```

```r
# Presento estudio
fnEstudioModelo(modelo_glm, estudioParam = FALSE)
```

Modelo final

```
Call:  NULL

Coefficients:
 (Intercept)     GrLivArea   OverallQual  TotalBsmtSF     X1stFlrSF
  12.0216986     0.1191885     0.0953092    0.0484734     0.0036513
  BsmtFinSF1       LotArea    GarageArea    X2ndFlrSF   OverallCond
   0.0326487     0.0367879     0.0168238    0.0065776     0.0578678
 FireplaceQu     YearBuilt   KitchenQual YearRemodAdd    GarageCars
   0.0243120     0.0822681     0.0190883    0.0255445     0.0323463
   ExterQual  BsmtFinType1  TotRmsAbvGrd    Fireplaces
   0.0121054     0.0093991    -0.0008169     0.0119496

Degrees of Freedom: 1022 Total (i.e. Null);  1004 Residual
Null Deviance:      167.8
Residual Deviance: 16.62  AIC: -1271
```



Summary resample$RMSE

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.107	0.116	0.132	0.130	0.141	0.159

Error de test

RMSE	Rsquared	MAE
0.12362414	0.89944770	0.08982126

Support Vector Machines

Aunque Las máquinas de vectores soporte fueron pensadas para resolver problemas de clasificación también pueden adaptarse para resolver problemas de regresión, estos modelos dan bastante buenos resultados cuando la variable objetivo no es separables linealmente dentro del espacio vectorial de los predictores y evitan en gran medida el problema del sobreajuste a los ejemplos de entrenamiento, por ello es una buena elección para este problema.

Las máquinas de soporte utilizan una función denominada Kernel para la búsqueda del hiperplano de separación, para ello mapean los datos en espacios de dimensiones superiores con la esperanza de que en este espacio de dimensiones superiores los datos puedan separarse más fácilmente o estar mejor estructurados.

En nuestro caso hemos probado con dos modelos con funciones Kernel distintas:

Support Vector Machines with Linear Kernel

Permite solo seleccionar líneas (o hiperplanos)

```
hiperparametros <- data.frame(C = c(0.0001, 0.001, 0.01, 0.1, 0.5))

t <- proc.time() # Inicia el cronómetro
modelo_svmlineal <- train(SalePrice ~ .
                          , data = dsTrain.training
                          , method = "svmLinear"
                          , tuneGrid = hiperparametros
```

```r
                          , metric = "RMSE"
                          , trControl = fitControl
                          , scale = FALSE )
proc.time()-t    # Detiene el cronómetro
```

```
##    user  system elapsed
##    9.30    0.11    9.41
```

```r
# Guardo resultado del calculo
fileOuput <- paste(dirSalida,'/','modelo_svmlineal','.RData',sep='')
save(modelo_svmlineal, file = fileOuput)

modelo_svmlineal
```

```
## Support Vector Machines with Linear Kernel
##
## 1023 samples
##   18 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 819, 818, 819, 819, 817, 819, ...
## Resampling results across tuning parameters:
##
##   C      RMSE       Rsquared   MAE
##   1e-04  0.1954332  0.8753939  0.14056770
##   1e-03  0.1343825  0.8933460  0.09532633
##   1e-02  0.1303350  0.8974270  0.09287770
##   1e-01  0.1304045  0.8972558  0.09319034
##   5e-01  0.1304532  0.8971945  0.09324501
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was C = 0.01.
```

```r
# Presento estudio
fnEstudioModelo(modelo_svmlineal)
```

Modelo final

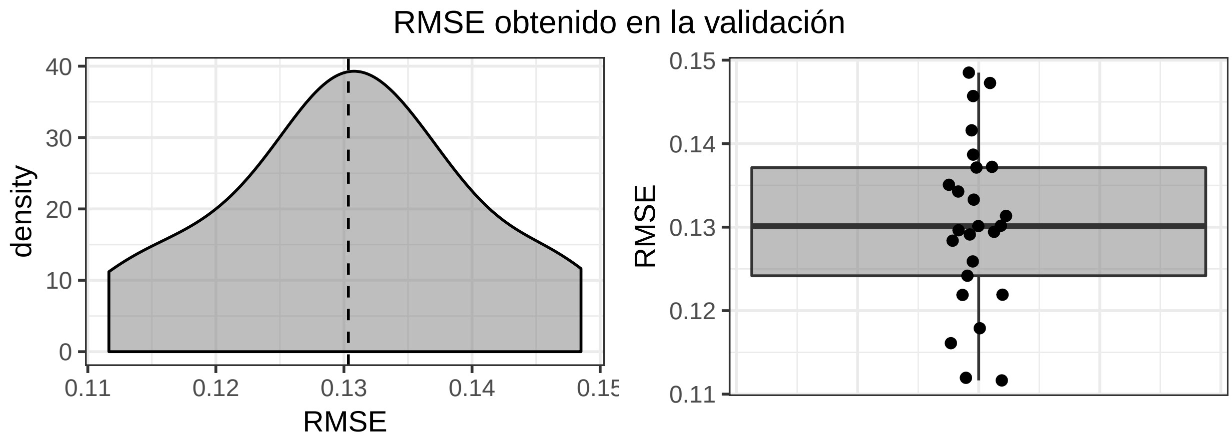
Support Vector Machine object of class "ksvm"

SV type: eps−svr (regression)
parameter : epsilon = 0.1 cost C = 0.01

Linear (vanilla) kernel function.

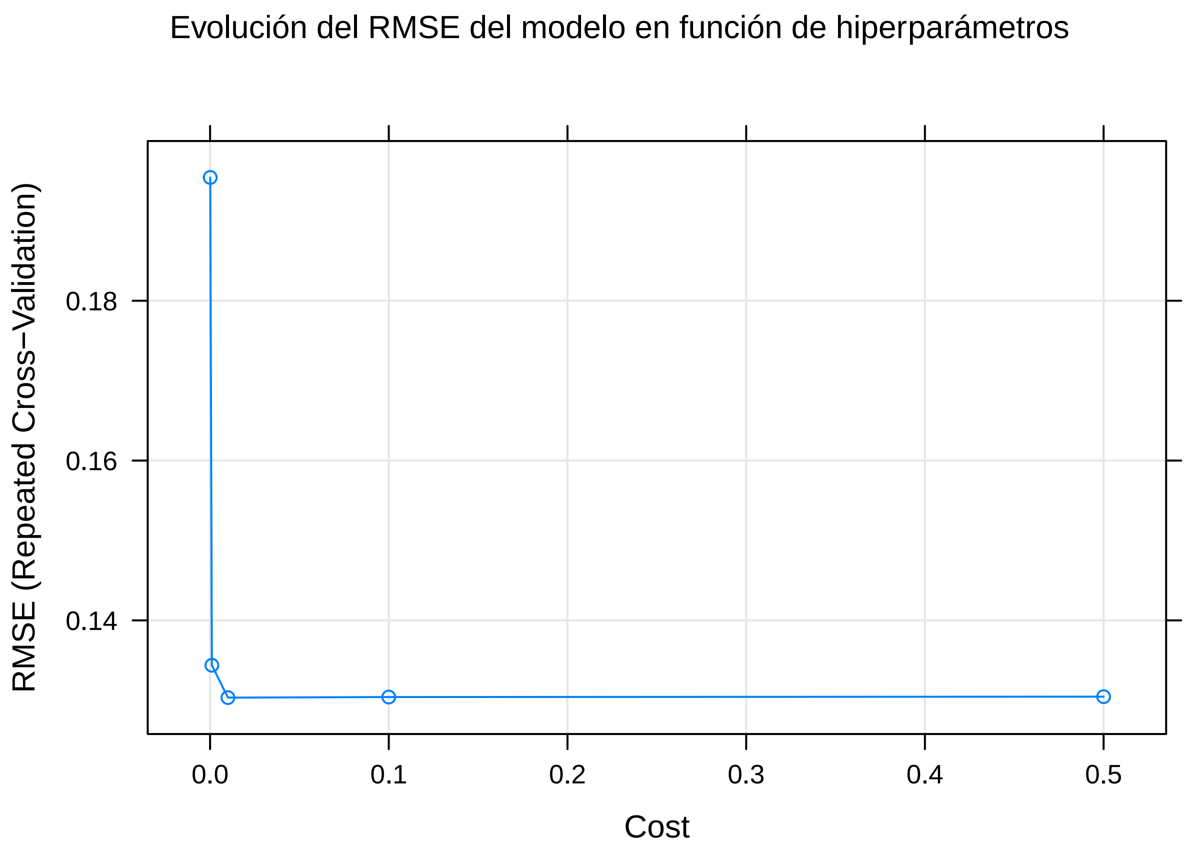
Number of Support Vectors : 349

Objective Function Value : −0.2898
Training error : 0.016503



Summary resample$RMSE					
Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.112	0.124	0.130	0.130	0.137	0.149

Error de test		
RMSE	Rsquared	MAE
0.12266020	0.90033055	0.08949161



Support Vector Machines with Radial Basis Function Kernel

Permiten seleccionar círculos (o hiperesferas)

```
hiperparametros <- expand.grid(sigma = c(0.0005, 0.001, 0.005)
                               ,C = c(1 , 20, 50, 100, 150, 200))

t <- proc.time() # Inicia el cronómetro
modelo_svmRadial <- train(SalePrice ~ .
                          , data = dsTrain.training
                          , method = "svmRadial"
                          , tuneGrid = hiperparametros
                          , metric = "RMSE"
                          , trControl = fitControl)
proc.time()-t    # Detiene el cronómetro
```

```
##    user  system elapsed
##   94.31    1.00   95.32
```

```
# Guardo resultado del calculo
fileOuput <- paste(dirSalida,'/','modelo_svmRadial','.RData',sep='')
save(modelo_svmRadial, file = fileOuput)

modelo_svmRadial
```

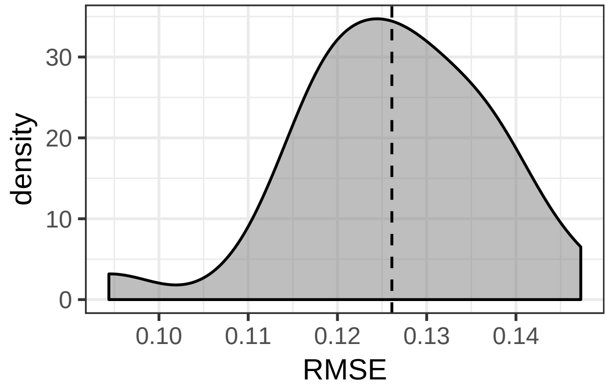
```
## Support Vector Machines with Radial Basis Function Kernel
##
## 1023 samples
##   18 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 818, 819, 819, 819, 817, 820, ...
## Resampling results across tuning parameters:
##
##   sigma  C    RMSE       Rsquared   MAE
##   5e-04    1  0.1412523  0.8896820  0.09829255
##   5e-04   20  0.1290817  0.9001458  0.08982726
##   5e-04   50  0.1276003  0.9022077  0.08862283
##   5e-04  100  0.1266084  0.9036535  0.08779753
##   5e-04  150  0.1263153  0.9041051  0.08744767
##   5e-04  200  0.1260968  0.9043727  0.08713926
##   1e-03    1  0.1355441  0.8939751  0.09411630
##   1e-03   20  0.1273470  0.9027064  0.08835688
##   1e-03   50  0.1262710  0.9042150  0.08732574
##   1e-03  100  0.1263330  0.9038234  0.08704188
##   1e-03  150  0.1266452  0.9032326  0.08724417
##   1e-03  200  0.1268319  0.9028378  0.08738384
##   5e-03    1  0.1319973  0.8974042  0.09133254
##   5e-03   20  0.1263575  0.9035468  0.08733542
##   5e-03   50  0.1283594  0.9003121  0.08908754
##   5e-03  100  0.1304894  0.8968835  0.09104080
##   5e-03  150  0.1321543  0.8942250  0.09218969
##   5e-03  200  0.1336779  0.8918734  0.09315568
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 5e-04 and C = 200.
```

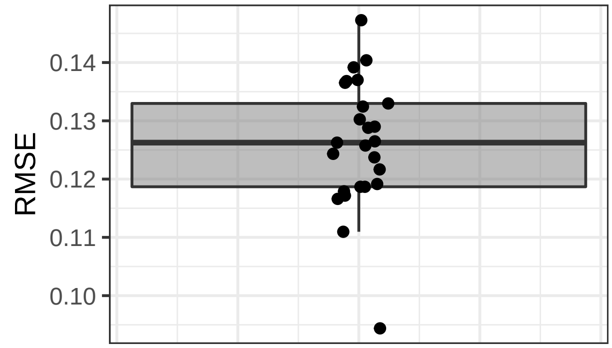
```r
# Presento estudio
fnEstudioModelo(modelo_svmRadial)
```

Modelo final

```
Support Vector Machine object of class "ksvm"

SV type: eps-svr  (regression)
parameter : epsilon = 0.1  cost C = 200

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  5e-04

Number of Support Vectors : 673

Objective Function Value : -25485.12
Training error : 0.088932
```

RMSE obtenido en la validación



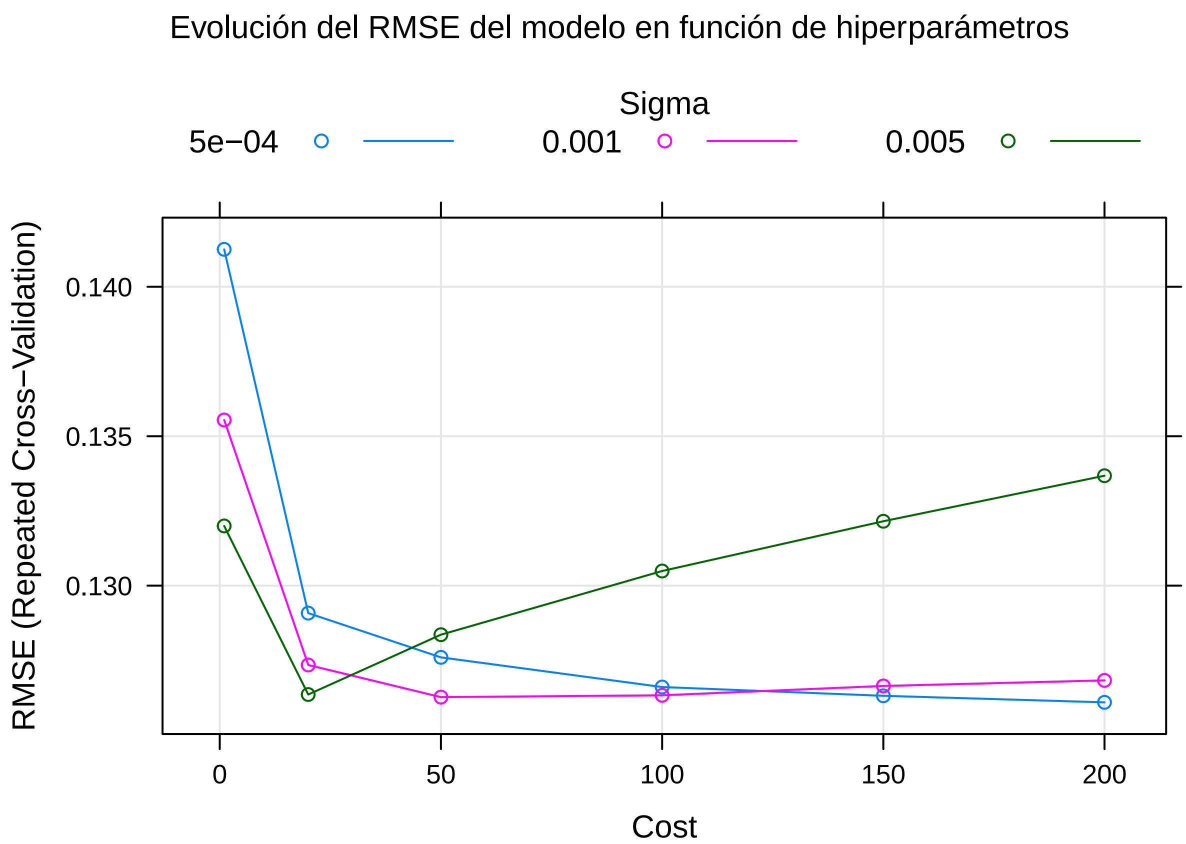


Summary resample$RMSE

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.0944	0.1190	0.1260	0.1260	0.1330	0.1470

Error de test

RMSE	Rsquared	MAE
0.11829999	0.90728931	0.08208742



Arboles de decisión

Estos modelos generan un conjunto de reglas para segmentar el espacio predictor en una serie de regiones simples, generando un árbol de decisiones.

El método es simple y puede servir bien para la interpretación de los datos, pero no tiene una gran precisión en la predicción. Sin embargo, la combinación de una gran cantidad de árboles puede mejorar mucho la predicción.

He realizado pruebas con:

XGBoost

XGBoost ha sido uno de los modelos más utilizados, esto es así porque se adapta fácilmente ya que es muy flexible, se puede usar tanto en regresión como en clasificación. Utiliza una combinación de modelos más simples (árboles de decisión) y potencia los resultados.

La gran desventaja de este modelo es el ajuste de su gran cantidad de parámetros.

```
hiperparametros <- expand.grid(
  nrounds = seq(from = 200, to = 500, by = 50),
  eta = c(0.025, 0.05, 0.1, 0.3),
  max_depth = c(2, 3, 4, 5, 6),
  gamma = 0,
  colsample_bytree = 1,
  min_child_weight = 1,
```

```r
  subsample = 1
)

t <- proc.time() # Inicia el cronómetro
modelo_XGBoost <- train(SalePrice ~ .
                        , data = dsTrain.training
                        , method = "xgbTree"
                        , tuneGrid = hiperparametros
                        , metric = "RMSE"
                        , trControl = fitControl)
proc.time()-t    # Detiene el cronómetro
```

```
##    user  system elapsed
## 1755.94  943.88 1159.72
```

```r
# Guardo resultado del calculo
fileOuput <- paste(dirSalida,'/','modelo_XGBoost','.RData',sep='')
save(modelo_XGBoost, file = fileOuput)

modelo_XGBoost
```

```
## eXtreme Gradient Boosting
##
## 1023 samples
##   18 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 818, 818, 819, 818, 819, 818, ...
## Resampling results across tuning parameters:
##
##   eta    max_depth  nrounds  RMSE       Rsquared   MAE
##   0.025  2          200      0.1686581  0.8731809  0.12881586
##   0.025  2          250      0.1451131  0.8807544  0.10432193
##   0.025  2          300      0.1395812  0.8852033  0.09924938
##   0.025  2          350      0.1370025  0.8881210  0.09724826
##   0.025  2          400      0.1353868  0.8901658  0.09609430
##   0.025  2          450      0.1342905  0.8916538  0.09529405
##   0.025  2          500      0.1334867  0.8928052  0.09464642
##   0.025  3          200      0.1613042  0.8821185  0.12312338
##   0.025  3          250      0.1393405  0.8871948  0.09986962
##   0.025  3          300      0.1353842  0.8902102  0.09609558
##   0.025  3          350      0.1339715  0.8919652  0.09478230
##   0.025  3          400      0.1332530  0.8929649  0.09402824
##   0.025  3          450      0.1329221  0.8934370  0.09362958
##   0.025  3          500      0.1327867  0.8936088  0.09339997
##   0.025  4          200      0.1601743  0.8835690  0.12215974
##   0.025  4          250      0.1380655  0.8887987  0.09896259
##   0.025  4          300      0.1345137  0.8913739  0.09514136
##   0.025  4          350      0.1335094  0.8925757  0.09399368
##   0.025  4          400      0.1330467  0.8932232  0.09343265
##   0.025  4          450      0.1328754  0.8934799  0.09315017
##   0.025  4          500      0.1328169  0.8935741  0.09298833
```

```
##    0.025  5           200      0.1591903  0.8857200  0.12146174
##    0.025  5           250      0.1368515  0.8907843  0.09794958
##    0.025  5           300      0.1335236  0.8928388  0.09422828
##    0.025  5           350      0.1328414  0.8935663  0.09332368
##    0.025  5           400      0.1326986  0.8937403  0.09300233
##    0.025  5           450      0.1326413  0.8938096  0.09292492
##    0.025  5           500      0.1326812  0.8937195  0.09294968
##    0.025  6           200      0.1594463  0.8865336  0.12182967
##    0.025  6           250      0.1369773  0.8910257  0.09783753
##    0.025  6           300      0.1336904  0.8928008  0.09430574
##    0.025  6           350      0.1331369  0.8932146  0.09361945
##    0.025  6           400      0.1330933  0.8931883  0.09343763
##    0.025  6           450      0.1330943  0.8931492  0.09340458
##    0.025  6           500      0.1331904  0.8929761  0.09347716
##    0.050  2           200      0.1354850  0.8898976  0.09620451
##    0.050  2           250      0.1336775  0.8924948  0.09472775
##    0.050  2           300      0.1327201  0.8939320  0.09381334
##    0.050  2           350      0.1325136  0.8942366  0.09341644
##    0.050  2           400      0.1324710  0.8942647  0.09319621
##    0.050  2           450      0.1325616  0.8940889  0.09311694
##    0.050  2           500      0.1325874  0.8940315  0.09296764
##    0.050  3           200      0.1334023  0.8926518  0.09416905
##    0.050  3           250      0.1328463  0.8934245  0.09350269
##    0.050  3           300      0.1327156  0.8935792  0.09320186
##    0.050  3           350      0.1326048  0.8937054  0.09300393
##    0.050  3           400      0.1326765  0.8935638  0.09288112
##    0.050  3           450      0.1328586  0.8932752  0.09293985
##    0.050  3           500      0.1329866  0.8930573  0.09294473
##    0.050  4           200      0.1334315  0.8925425  0.09357676
##    0.050  4           250      0.1330086  0.8931625  0.09306289
##    0.050  4           300      0.1329777  0.8931748  0.09286633
##    0.050  4           350      0.1331852  0.8928528  0.09284632
##    0.050  4           400      0.1333514  0.8925785  0.09293910
##    0.050  4           450      0.1335654  0.8922359  0.09315131
##    0.050  4           500      0.1338918  0.8916876  0.09349162
##    0.050  5           200      0.1336103  0.8923508  0.09358473
##    0.050  5           250      0.1336570  0.8922359  0.09345034
##    0.050  5           300      0.1338288  0.8919428  0.09350994
##    0.050  5           350      0.1340645  0.8915647  0.09371506
##    0.050  5           400      0.1343026  0.8911517  0.09388282
##    0.050  5           450      0.1345193  0.8908108  0.09410619
##    0.050  5           500      0.1346721  0.8905696  0.09427274
##    0.050  6           200      0.1335153  0.8926352  0.09357134
##    0.050  6           250      0.1337190  0.8922427  0.09368625
##    0.050  6           300      0.1340166  0.8917469  0.09399539
##    0.050  6           350      0.1343453  0.8912116  0.09437216
##    0.050  6           400      0.1345510  0.8908722  0.09466792
##    0.050  6           450      0.1347398  0.8905625  0.09488839
##    0.050  6           500      0.1348982  0.8903042  0.09511398
##    0.100  2           200      0.1333515  0.8927973  0.09397286
##    0.100  2           250      0.1332775  0.8928779  0.09372675
##    0.100  2           300      0.1334075  0.8926425  0.09360503
##    0.100  2           350      0.1336216  0.8923290  0.09360180
##    0.100  2           400      0.1338724  0.8919244  0.09360336
```

```
##    0.100  2          450      0.1340846  0.8916140  0.09367040
##    0.100  2          500      0.1343894  0.8911481  0.09380467
##    0.100  3          200      0.1338663  0.8916217  0.09329203
##    0.100  3          250      0.1340879  0.8912279  0.09345280
##    0.100  3          300      0.1344295  0.8906736  0.09368090
##    0.100  3          350      0.1348062  0.8901142  0.09384833
##    0.100  3          400      0.1352184  0.8894857  0.09412066
##    0.100  3          450      0.1355371  0.8889974  0.09437732
##    0.100  3          500      0.1358922  0.8884171  0.09467968
##    0.100  4          200      0.1341348  0.8911749  0.09347594
##    0.100  4          250      0.1344864  0.8906205  0.09366788
##    0.100  4          300      0.1347273  0.8902667  0.09397089
##    0.100  4          350      0.1349954  0.8898512  0.09426140
##    0.100  4          400      0.1352229  0.8895045  0.09449260
##    0.100  4          450      0.1354278  0.8891728  0.09471466
##    0.100  4          500      0.1356322  0.8888469  0.09495706
##    0.100  5          200      0.1356595  0.8887608  0.09480588
##    0.100  5          250      0.1360306  0.8881489  0.09522613
##    0.100  5          300      0.1363148  0.8876840  0.09556350
##    0.100  5          350      0.1365382  0.8873184  0.09581678
##    0.100  5          400      0.1366599  0.8871156  0.09598937
##    0.100  5          450      0.1367689  0.8869338  0.09612521
##    0.100  5          500      0.1368374  0.8868169  0.09622034
##    0.100  6          200      0.1359115  0.8885698  0.09561918
##    0.100  6          250      0.1361189  0.8882096  0.09583905
##    0.100  6          300      0.1362945  0.8879167  0.09601290
##    0.100  6          350      0.1364051  0.8877225  0.09613207
##    0.100  6          400      0.1364642  0.8876270  0.09620385
##    0.100  6          450      0.1365035  0.8875583  0.09625234
##    0.100  6          500      0.1365310  0.8875112  0.09629524
##    0.300  2          200      0.1403653  0.8811588  0.09920321
##    0.300  2          250      0.1412117  0.8797652  0.09956681
##    0.300  2          300      0.1423277  0.8779383  0.10031554
##    0.300  2          350      0.1428099  0.8771796  0.10072288
##    0.300  2          400      0.1431633  0.8765749  0.10096368
##    0.300  2          450      0.1435368  0.8759534  0.10125258
##    0.300  2          500      0.1440297  0.8751603  0.10161427
##    0.300  3          200      0.1425781  0.8773955  0.10043838
##    0.300  3          250      0.1431561  0.8764676  0.10100270
##    0.300  3          300      0.1435802  0.8757788  0.10138578
##    0.300  3          350      0.1437411  0.8755398  0.10158734
##    0.300  3          400      0.1439429  0.8752283  0.10187508
##    0.300  3          450      0.1441049  0.8749536  0.10201409
##    0.300  3          500      0.1442558  0.8746984  0.10216487
##    0.300  4          200      0.1439998  0.8751788  0.10241325
##    0.300  4          250      0.1442434  0.8747702  0.10264236
##    0.300  4          300      0.1443293  0.8746230  0.10278538
##    0.300  4          350      0.1443975  0.8745078  0.10285847
##    0.300  4          400      0.1444348  0.8744456  0.10290002
##    0.300  4          450      0.1444367  0.8744466  0.10290763
##    0.300  4          500      0.1444377  0.8744452  0.10290819
##    0.300  5          200      0.1466048  0.8703508  0.10389401
##    0.300  5          250      0.1466497  0.8702677  0.10395129
##    0.300  5          300      0.1466487  0.8702692  0.10395202
```

```
##   0.300  5          350      0.1466487  0.8702692  0.10395202
##   0.300  5          400      0.1466487  0.8702692  0.10395202
##   0.300  5          450      0.1466487  0.8702692  0.10395202
##   0.300  5          500      0.1466487  0.8702692  0.10395202
##   0.300  6          200      0.1460912  0.8712796  0.10394517
##   0.300  6          250      0.1460912  0.8712796  0.10394517
##   0.300  6          300      0.1460912  0.8712796  0.10394517
##   0.300  6          350      0.1460912  0.8712796  0.10394517
##   0.300  6          400      0.1460912  0.8712796  0.10394517
##   0.300  6          450      0.1460912  0.8712796  0.10394517
##   0.300  6          500      0.1460912  0.8712796  0.10394517
##
## Tuning parameter 'gamma' was held constant at a value of 0
##  1
## Tuning parameter 'min_child_weight' was held constant at a value of
##  1
## Tuning parameter 'subsample' was held constant at a value of 1
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nrounds = 400, max_depth = 2,
##  eta = 0.05, gamma = 0, colsample_bytree = 1, min_child_weight = 1
##  and subsample = 1.
```

```r
# Presento estudio
fnEstudioModelo(modelo_XGBoost, estudioParam = FALSE)
```

Modelo final

xgb.Booster

raw: 153.4 Kb

call:

xgboost::xgb.train(params = list(eta = param$eta, max_depth = param$max_depth,
gamma = param$gamma, colsample_bytree = param$colsample_bytree,
min_child_weight = param$min_child_weight, subsample = param$subsample),
data = x, nrounds = param$nrounds, objective = "reg:linear")

params (as set within xgb.train):

nax_depth = "2", gamma = "0", colsample_bytree = "1", min_child_weight = "1", subsample = "1", objective = "reg:line

xgb.attributes:

niter

callbacks:

cb.print.evaluation(period = print_every_n)

# of features: 18

niter: 400

nfeatures : 18

3smtFinSF1 LotArea GarageArea X2ndFlrSF OverallCond FireplaceQu YearBuilt KitchenQual YearRemodAdd Garage

problemType : Regression

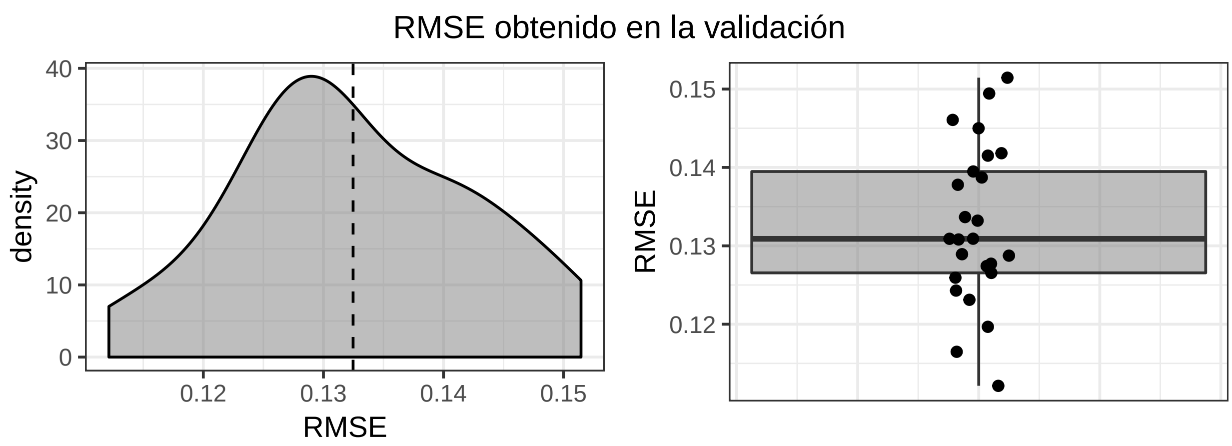
tuneValue :

	nrounds	max_depth	eta	gamma	colsample_bytree	min_child_weight
40	400	2	0.05	0	1	1

	subsample
40	1

obsLevels : NA

param :



Summary resample$RMSE

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.112	0.127	0.131	0.132	0.139	0.151

Error de test

RMSE	Rsquared	MAE
0.12939657	0.89027801	0.09035004

Random Forest

Utiliza una combinación de árboles, en este caso cada árbol depende de los valores de un vector aleatorio.

La ventaja de este método frente a XGBoost es que es más fácil de ajustar, aunque parece menos flexible. También en este modelo se ha detectado un sobreajuste al conjunto de entrenamiento, dando valores bastante buenos en los entrenamientos, pero bastante más altos en test.

```
t <- proc.time() # Inicia el cronómetro
modelo_rf <- train(SalePrice ~ .
                          , data = dsTrain.training
                          , method = "ranger"
                          #, tuneGrid = hiperparametros
                          , tuneLength = 10
                          , metric = "RMSE"
                          , trControl = fitControl
                          , num.trees = 500)
proc.time()-t     # Detiene el cronómetro
```

```
##     user  system elapsed
## 1430.61   12.83  225.70
```

```
# Guardo resultado del calculo
fileOuput <- paste(dirSalida,'/','modelo_rf','.RData',sep='')
save(modelo_rf, file = fileOuput)
```

```r
modelo_rf
```

```
## Random Forest
##
## 1023 samples
##   18 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 818, 819, 818, 818, 819, 819, ...
## Resampling results across tuning parameters:
##
##   mtry  splitrule   RMSE       Rsquared   MAE
##    2    variance    0.1404124  0.8890309  0.09656490
##    2    extratrees  0.1457740  0.8840146  0.10111345
##    3    variance    0.1386272  0.8899519  0.09535901
##    3    extratrees  0.1424959  0.8859265  0.09910765
##    5    variance    0.1379895  0.8893221  0.09479262
##    5    extratrees  0.1411151  0.8857655  0.09853838
##    7    variance    0.1383740  0.8880158  0.09508236
##    7    extratrees  0.1404508  0.8857805  0.09823688
##    9    variance    0.1392612  0.8861204  0.09590078
##    9    extratrees  0.1401361  0.8857055  0.09812964
##   10    variance    0.1391568  0.8860387  0.09582295
##   10    extratrees  0.1397212  0.8861267  0.09789956
##   12    variance    0.1403616  0.8837282  0.09669394
##   12    extratrees  0.1398033  0.8858183  0.09812776
##   14    variance    0.1413949  0.8815332  0.09735560
##   14    extratrees  0.1399545  0.8852515  0.09805367
##   16    variance    0.1424339  0.8794758  0.09809577
##   16    extratrees  0.1396919  0.8855031  0.09792580
##   18    variance    0.1443763  0.8756437  0.09931080
##   18    extratrees  0.1399648  0.8849904  0.09807894
##
## Tuning parameter 'min.node.size' was held constant at a value of 5
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were mtry = 5, splitrule =
##  variance and min.node.size = 5.
```

```r
# Presento estudio
fnEstudioModelo(modelo_rf)
```

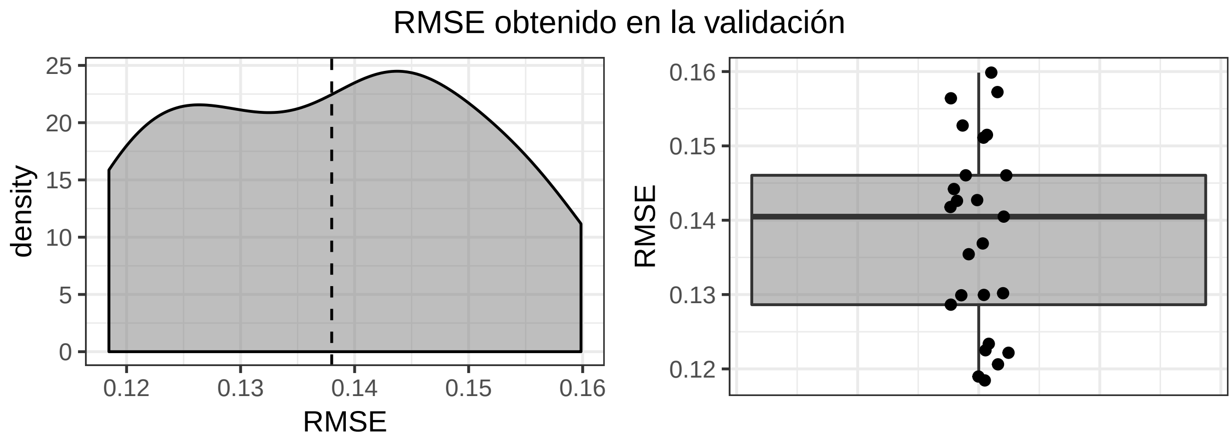
Modelo final

Ranger result

Call:

```
ι = x,      mtry = min(param$mtry, ncol(x)), min.node.size = param$min.node.size,      splitrule = as.character(param$sp
```

Type:	Regression
Number of trees:	500
Sample size:	1023
Number of independent variables:	18
Mtry:	5
Target node size:	5
Variable importance mode:	none
Splitrule:	variance
OOB prediction error (MSE):	0.0185743
R squared (OOB):	0.8868994

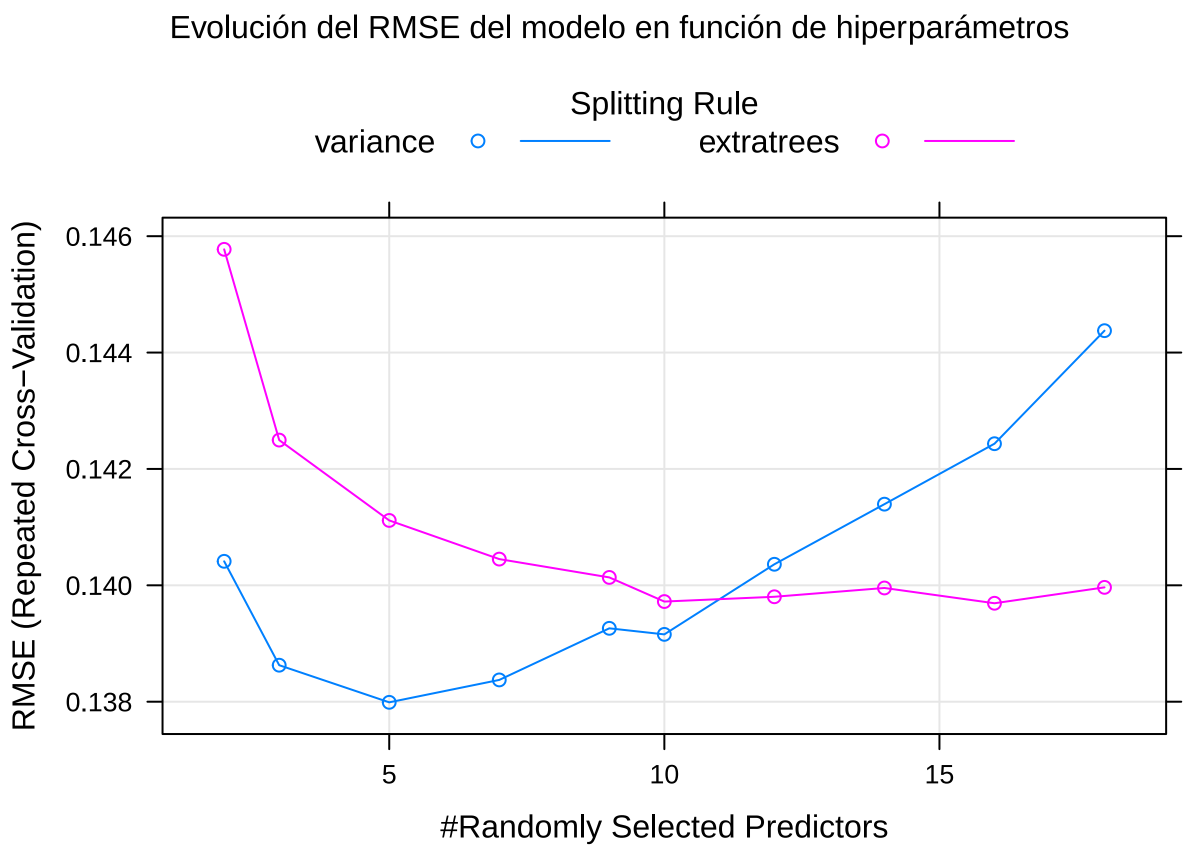


Summary resample$RMSE

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.118	0.129	0.140	0.138	0.146	0.160

Error de test

RMSE	Rsquared	MAE
0.13088786	0.88932557	0.09041785



Stochastic Gradient Boosting

GBM realiza un proceso iterativo donde se introducen nuevos modelos que se basan en los errores de las iteraciones anteriores para minimizar el error (aumento de gradiente) de una función objetivo.

Este método es muy versátil, pudiendo resolver una gran variedad de problemas, sus desventajas son que es sensible al sobreajuste, tiene un gran número de hiperparámetros, por lo que es complicado de ajustar y el tiempo de entrenamiento es bastante alto.

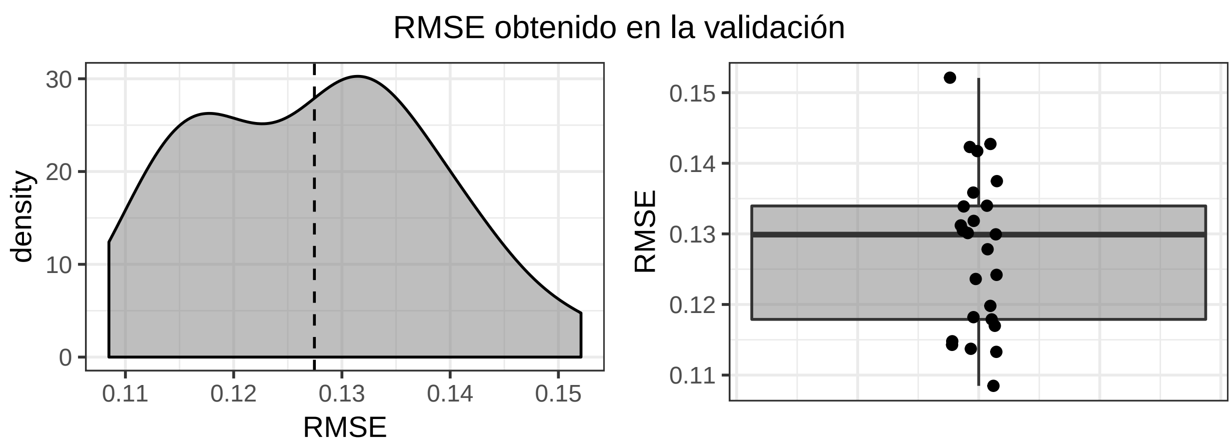
```
t <- proc.time() # Inicia el cronómetro
hiperparametros <- expand.grid(interaction.depth = c(2,3),
                               n.trees = c(2000, 3000, 4000),
                               shrinkage = c( 0.01, 0.1),
                               n.minobsinnode = c(2, 5, 10))

t <- proc.time() # Inicia el cronómetro
modelo_gbm <- train(SalePrice ~ .
                          , data = dsTrain.training
                          , method = "gbm"
                          , tuneGrid = hiperparametros
                          , metric = "RMSE"
                          #, distribution
                          , trControl = fitControl
                          , verbose = FALSE # Para que no se muestre cada iteración por pantalla
                    )
proc.time()-t    # Detiene el cronómetro
```

```
##    user  system elapsed
##  745.45    0.00  745.64
```

```
# Guardo resultado del calculo
fileOuput <- paste(dirSalida,'/','modelo_gbm','.RData',sep='')
save(modelo_gbm, file = fileOuput)

# Presento estudio
fnEstudioModelo(modelo_gbm)
```

Modelo final

A gradient boosted model with gaussian loss function.
2000 iterations were performed.
There were 18 predictors of which 18 had non−zero influence.

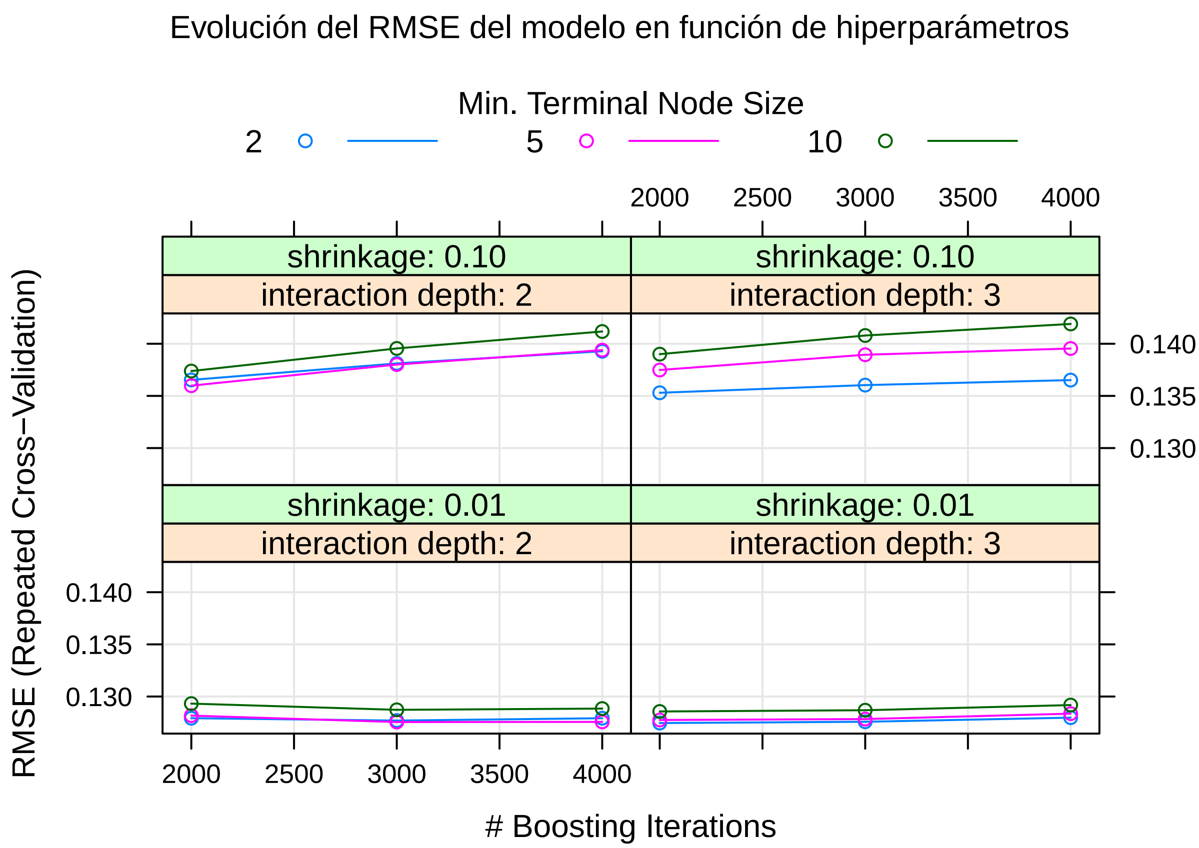


Summary resample$RMSE

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.108	0.118	0.130	0.127	0.134	0.152

Error de test

RMSE	Rsquared	MAE
0.12858819	0.89263459	0.08885095



k-Nearest Neighbors

Es un método tanto de clasificación como de regresión, bastante sencillo y supervisado, una característica principal es que está basado en instancia, esto quiere decir que no se genera un modelo real, sino que se guardan las observaciones.

El algoritmo busca las observaciones más cercanas a la que se está tratando y predice el valor de interés mediante los datos que le rodean. El parámetro k indica cuantos puntos "vecinos" se deben de tener en cuenta para ajustar.

KNN tiende a funcionar mejor con dataset pequeños y con pocos predictores, ya que utiliza todo el conjunto de datos para entrenar. Además, es muy costoso tanto en uso de CPU como en memoria.

```
hiperparametros <- data.frame(k = c(3:20))

t <- proc.time() # Inicia el cronómetro
modelo_knn <- train(SalePrice ~ .
                          , data = dsTrain.training
                          , method = "knn"
                          , tuneGrid = hiperparametros
                          , metric = "RMSE"
                          , trControl = fitControl)
proc.time()-t    # Detiene el cronómetro
```

```
##    user  system elapsed
##    4.57    0.00    4.58
```

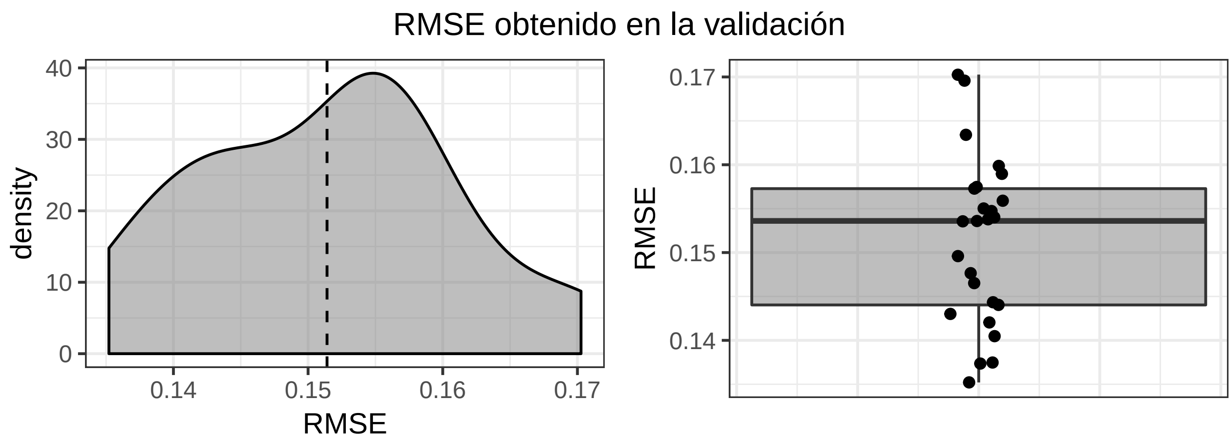
```r
# Guardo resultado del calculo
fileOuput <- paste(dirSalida,'/','modelo_knn','.RData',sep='')
save(modelo_knn, file = fileOuput)

modelo_knn
```

```
## k-Nearest Neighbors
##
## 1023 samples
##   18 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 819, 818, 819, 819, 817, 817, ...
## Resampling results across tuning parameters:
##
##   k   RMSE       Rsquared   MAE
##    3  0.1569542  0.8507359  0.1132504
##    4  0.1540684  0.8573454  0.1114176
##    5  0.1523508  0.8620346  0.1101791
##    6  0.1515969  0.8647515  0.1098665
##    7  0.1514066  0.8663030  0.1094428
##    8  0.1520887  0.8661936  0.1098689
##    9  0.1530425  0.8658250  0.1102273
##   10  0.1539387  0.8651285  0.1106684
##   11  0.1550598  0.8645766  0.1109630
##   12  0.1557512  0.8645682  0.1114859
##   13  0.1567840  0.8638818  0.1121311
##   14  0.1576137  0.8635890  0.1125655
##   15  0.1582433  0.8633928  0.1131618
##   16  0.1590034  0.8628817  0.1137138
##   17  0.1595060  0.8629585  0.1139534
##   18  0.1599803  0.8628213  0.1141247
##   19  0.1604251  0.8627435  0.1143184
##   20  0.1609858  0.8622596  0.1147608
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 7.
```

```r
# Presento estudio
fnEstudioModelo(modelo_knn)
```

Modelo final

7−nearest neighbor regression model

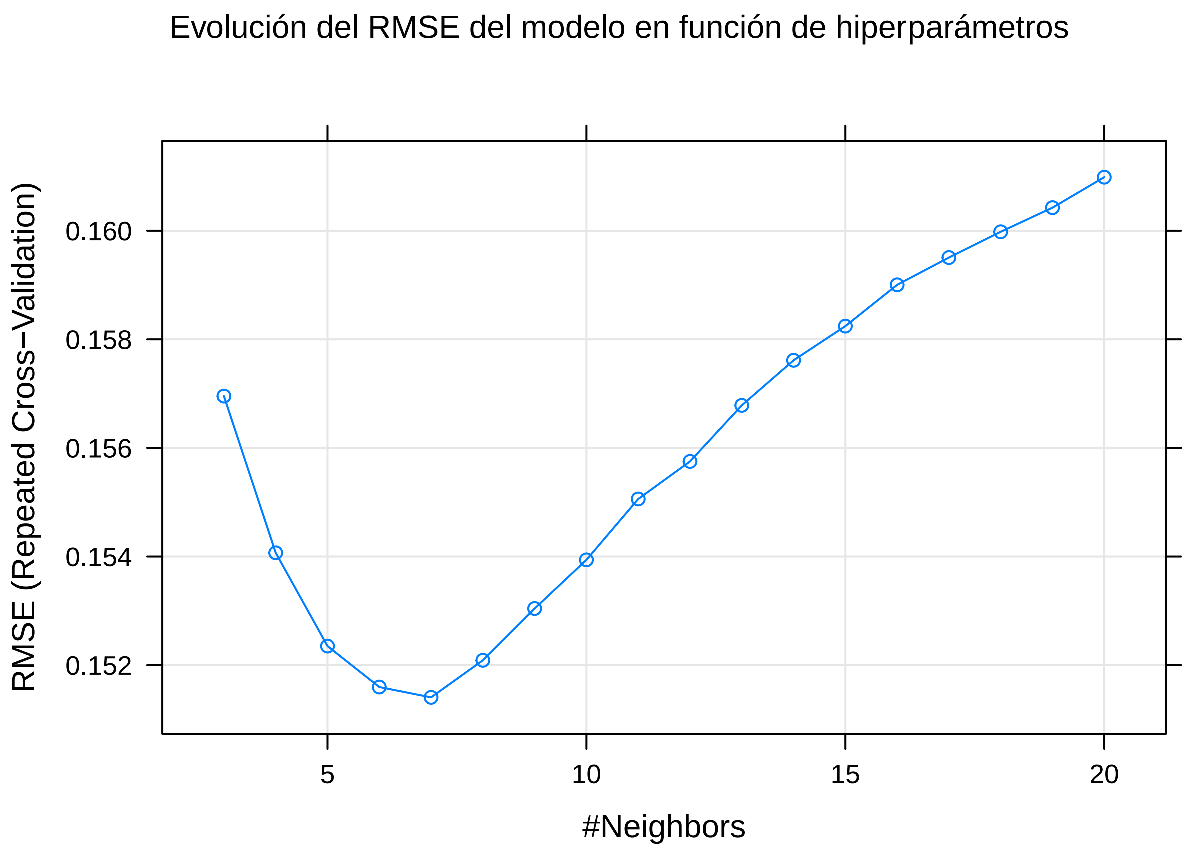


Summary resample$RMSE

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.135	0.144	0.154	0.151	0.157	0.170

Error de test

RMSE	Rsquared	MAE
0.1502120	0.8538856	0.1083629



LASSO

0perador de mínima contracción y selección absoluta. (least absolute shrinkage and selection operator) se utiliza para modelos de sistemas no lineales.

Realiza selección de variables y regularización para mejorar la exactitud e interpretabilidad del modelo. Establece algunos coeficientes a cero lo que permite eliminar variables.

```
hiperparametros <- expand.grid(fraction=c(1,0.1,0.01,0.001))

t <- proc.time() # Inicia el cronómetro
modelo_lasso <- train(SalePrice ~ .
                          , data = dsTrain.training
                          , method = "lasso"
                          , tuneGrid = hiperparametros
                          #, tuneLength = 10
                          , metric = "RMSE"
                          , trControl = fitControl)
proc.time()-t    # Detiene el cronómetro
```

```
##    user  system elapsed
##    1.22    0.00    1.22
```

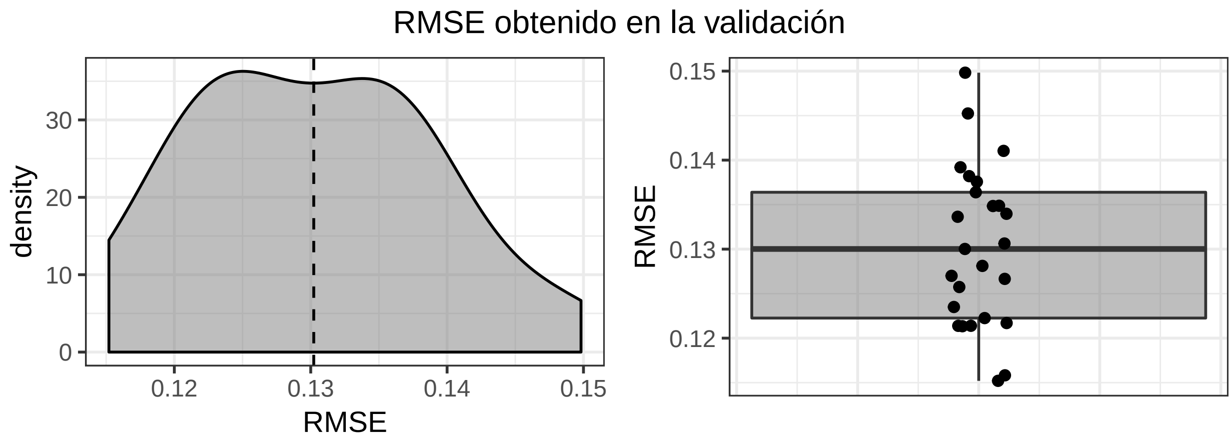
```r
# Guardo resultado del calculo
fileOuput <- paste(dirSalida,'/','modelo_lasso','.RData',sep='')
save(modelo_lasso, file = fileOuput)

modelo_lasso
```

```
## The lasso
##
## 1023 samples
##   18 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 818, 818, 817, 819, 820, 819, ...
## Resampling results across tuning parameters:
##
##   fraction  RMSE       Rsquared   MAE
##   0.001     0.4041699  0.6784465  0.31297604
##   0.010     0.3993983  0.6784465  0.30902357
##   0.100     0.3534029  0.6784465  0.27084673
##   1.000     0.1302223  0.8972826  0.09317991
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was fraction = 1.
```

```r
# Presento estudio
fnEstudioModelo(modelo_lasso, estudioParam = FALSE)
```

Modelo final

```
Call:
elasticnet::enet(x = as.matrix(x), y = y, lambda = 0)
Cp statistics of the Lasso fit
2714.166 1945.090 1806.648 1663.940 1315.678 1250.929 1198.045  838.956  709.114  399.987  371.374  349.568
DF:  1  2  3  4  5  6  7  8  9 10 11 12 12 12 13 14 15 16 17 18 19
Sequence of  moves:
     OverallQual GrLivArea GarageCars TotalBsmtSF KitchenQual YearRemodAdd
Var            2         1         14           3          12           13
Step           1         2          3           4           5            6
     GarageArea YearBuilt X1stFlrSF FireplaceQu BsmtFinSF1 LotArea
Var           7        11         4          10          5       6
Step          7         8         9          10         11      12
     X1stFlrSF Fireplaces OverallCond ExterQual BsmtFinType1 X2ndFlrSF
Var         −4         18           9        15           16         8
Step        13         14          15        16           17        18
     TotRmsAbvGrd X1stFlrSF
Var            17         4 21
Step           19        20 21
```



Summary resample$RMSE

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.115	0.122	0.130	0.130	0.136	0.150

Error de test

RMSE	Rsquared	MAE
0.12362414	0.89944770	0.08982126

Elasticnet

Es una combinación de LASSO y Ridge regression, donde predictores altamente correlacionados presentan coeficientes estimados similares.

```
hiperparametros <- expand.grid(alpha=seq(0,1,by=.5),lambda=seq(0,0.2,by=.1))

t <- proc.time() # Inicia el cronómetro
modelo_glmnet <- train(SalePrice ~ .
                          , data = dsTrain.training
                          , method = "glmnet"
                          , tuneGrid = hiperparametros
                          , metric = "RMSE"
                          , trControl = fitControl)
proc.time()-t    # Detiene el cronómetro
```

```
##    user  system elapsed
##    2.00    0.01    2.01
```

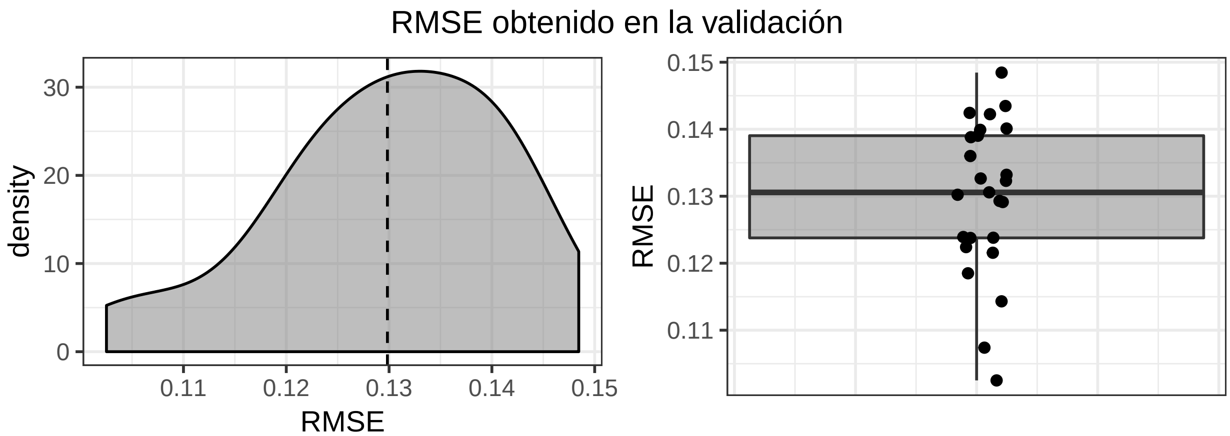
```
# Guardo resultado del calculo
fileOuput <- paste(dirSalida,'/','modelo_glmnet','.RData',sep='')
save(modelo_glmnet, file = fileOuput)

modelo_glmnet
```

```
## glmnet
##
## 1023 samples
##   18 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 819, 818, 819, 818, 818, 818, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda  RMSE       Rsquared   MAE
##   0.0    0.0     0.1303301  0.8969996  0.09330854
##   0.0    0.1     0.1328103  0.8943733  0.09483529
##   0.0    0.2     0.1371617  0.8908158  0.09771586
##   0.5    0.0     0.1298714  0.8975497  0.09311314
##   0.5    0.1     0.1648468  0.8717091  0.11637271
##   0.5    0.2     0.2148445  0.8462325  0.15622913
##   1.0    0.0     0.1298400  0.8975964  0.09308700
##   1.0    0.1     0.2087889  0.8234549  0.15261446
##   1.0    0.2     0.2988081  0.7260932  0.22621028
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 1 and lambda = 0.
```

```r
# Presento estudio
fnEstudioModelo(modelo_glmnet)
```

Modelo final

[17,] 11 0.7923 0.075310
[18,] 11 0.8070 0.068620
[19,] 12 0.8193 0.062520
[20,] 12 0.8306 0.056970
[21,] 11 0.8399 0.051910
[22,] 11 0.8477 0.047300
[23,] 11 0.8542 0.043100
[24,] 11 0.8596 0.039270
[25,] 12 0.8640 0.035780
[26,] 14 0.8697 0.032600
[27,] 15 0.8750 0.029700
[28,] 15 0.8794 0.027070
[29,] 15 0.8830 0.024660
[30,] 15 0.8861 0.022470
[31,] 15 0.8886 0.020470
[32,] 15 0.8907 0.018650
[33,] 15 0.8924 0.017000
[34,] 15 0.8939 0.015490
[35,] 15 0.8951 0.014110
[36,] 15 0.8961 0.012860
[37,] 15 0.8969 0.011720
[38,] 15 0.8976 0.010680
[39,] 15 0.8982 0.009727
[40,] 15 0.8986 0.008863
[41,] 15 0.8990 0.008075

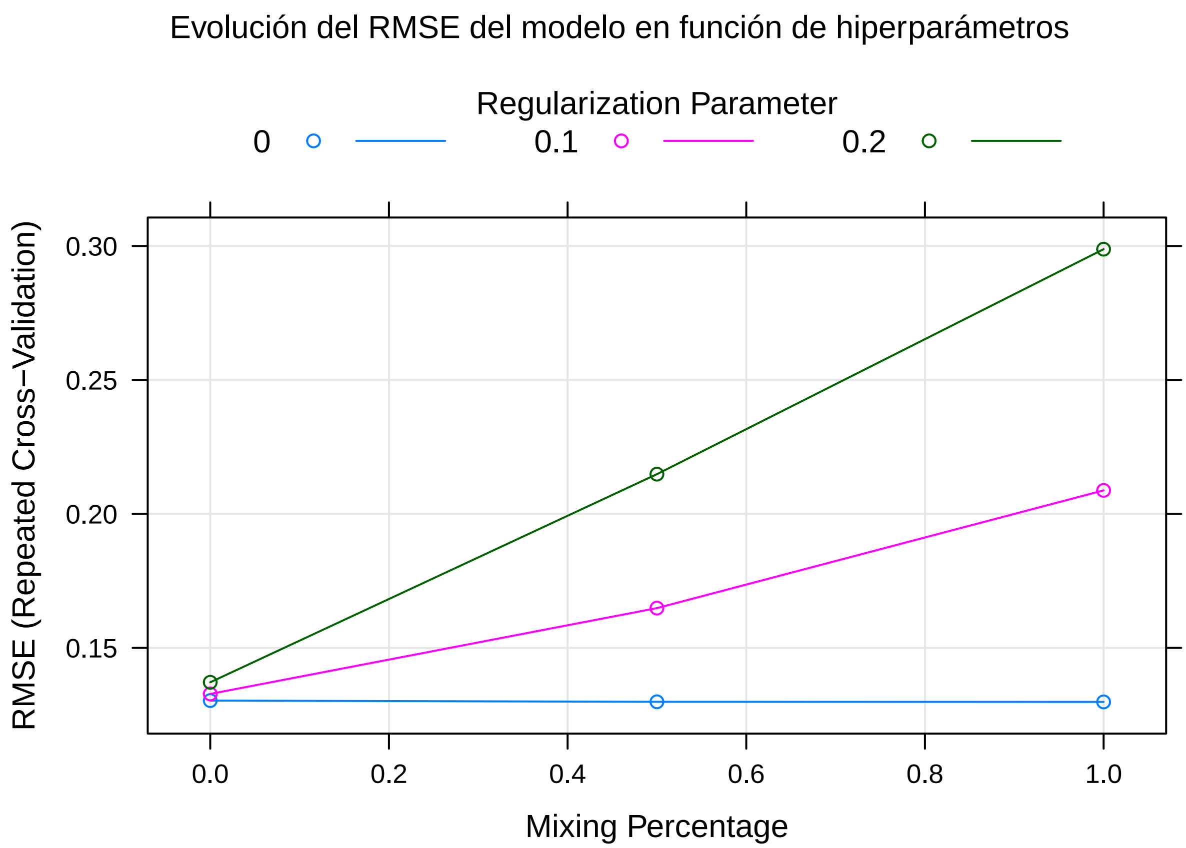


Summary resample$RMSE

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.103	0.124	0.131	0.130	0.139	0.148

Error de test

RMSE	Rsquared	MAE
0.12376926	0.89929167	0.08990196



Comparación de modelos

En este punto trataremos de identificar cual de los modelos es mejor para ello tendremos en cuenta las metricas de validación calculadas en el entrenamiento y el error de test.

Utilizare la función resamples() para extraer las metricas de los modelos entrenados.

Métricas

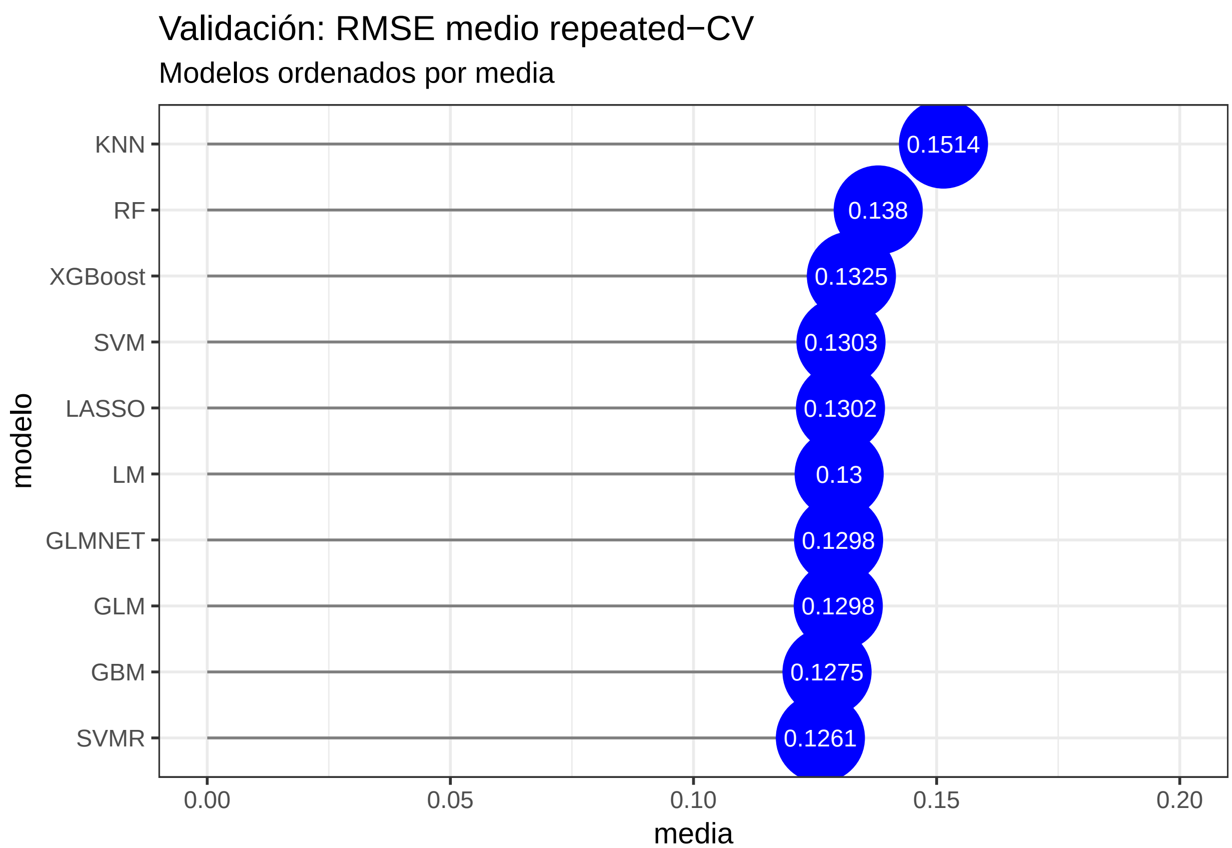
```
# creamos una lista con los modelos entrenados
modelos <- list(GBM = modelo_gbm
                , GLM = modelo_glm
                , LM = modelo_lm
                , KNN = modelo_knn
                , RF = modelo_rf
                , SVM = modelo_svmlineal
                , SVMR = modelo_svmRadial
                , LASSO = modelo_lasso
                , XGBoost = modelo_XGBoost
                , GLMNET = modelo_glmnet)

resultados_resamples <- resamples(modelos)
```

```r
# Se trasforma el dataframe devuelto por resamples() para separar el nombre del
# modelo y las métricas en columnas distintas.
metricas_resamples <- resultados_resamples$values %>%
                         gather(key = "modelo", value = "valor", -Resample) %>%
                         separate(col = "modelo", into = c("modelo", "metrica"),
                                  sep = "~", remove = TRUE)

# Se obtienen las medias por modelo
metricas_resamples %>%
  group_by(modelo, metrica) %>%
  summarise(media = mean(valor)) %>%
  spread(key = metrica, value = media) %>%
  arrange(RMSE)
```

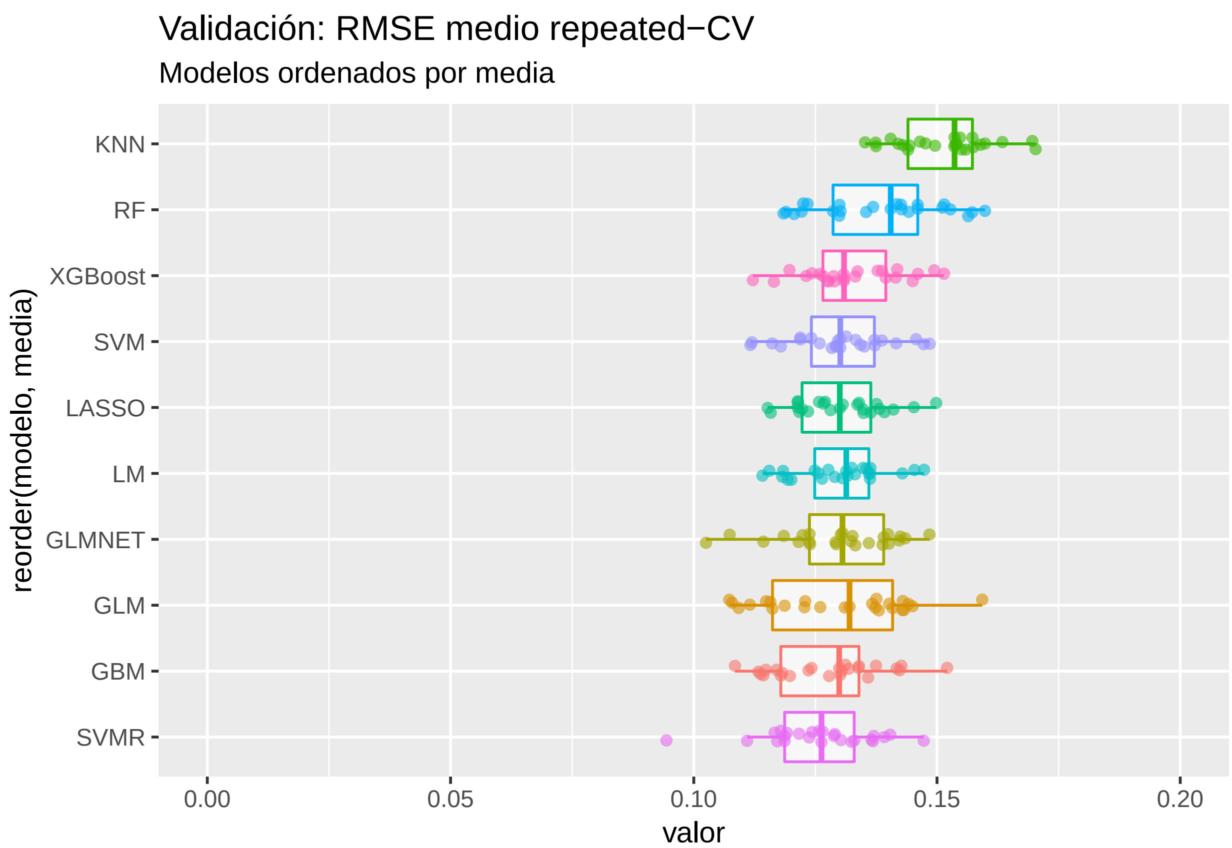
```
## # A tibble: 10 x 4
## # Groups:   modelo [10]
##    modelo     MAE  RMSE Rsquared
##    <chr>    <dbl> <dbl>    <dbl>
##  1 SVMR    0.0871 0.126    0.904
##  2 GBM     0.0900 0.127    0.902
##  3 GLM     0.0932 0.130    0.898
##  4 GLMNET  0.0931 0.130    0.898
##  5 LM      0.0931 0.130    0.898
##  6 LASSO   0.0932 0.130    0.897
##  7 SVM     0.0929 0.130    0.897
##  8 XGBoost 0.0932 0.132    0.894
##  9 RF      0.0948 0.138    0.889
## 10 KNN     0.109  0.151    0.866
```

Comparativa gráfica

```r
dg <- metricas_resamples %>%
  filter(metrica == "RMSE") %>%
  group_by(modelo) %>%
  summarise(media = mean(valor))

ggplot(dg, aes(x = reorder(modelo, media), y = media, label = round(media, 4))) +
    geom_segment(aes(x = reorder(modelo, media), y = 0,
                     xend = modelo, yend = media),
                     color = "grey50") +
    geom_point(size = 15, color = "blue") +
    geom_text(color = "white", size = 3) +
    scale_y_continuous(limits = c(0, 0.2)) +
    labs(title = "Validación: RMSE medio repeated-CV",
         subtitle = "Modelos ordenados por media",
         x = "modelo") +
    coord_flip() +
    theme_bw()
```




```r
dg <-metricas_resamples %>%
  filter(metrica == "RMSE") %>%
  group_by(modelo) %>%
  mutate(media = mean(valor)) %>%
  ungroup()

ggplot(dg, aes(x = reorder(modelo, media), y = valor, color = modelo)) +
    geom_boxplot(alpha = 0.6, outlier.shape = NA) +
    geom_jitter(width = 0.1, alpha = 0.6) +
    scale_y_continuous(limits = c(0, 0.2)) +
    labs(title = "Validación: RMSE medio repeated-CV",
         subtitle = "Modelos ordenados por media") +
    coord_flip() +
    theme(legend.position = "none")
```



Comparativas

La función diff() hace comparaciones por pares aplicando un t-test pareado con correcciones por comparaciones múltiples.

```
difs <- diff(resultados_resamples)

difs
```

```
##
## Call:
## diff.resamples(x = resultados_resamples)
##
## Models: GBM, GLM, LM, KNN, RF, SVM, SVMR, LASSO, XGBoost, GLMNET
## Metrics: MAE, RMSE, Rsquared
## Number of differences: 45
## p-value adjustment: bonferroni
```

```
summary(difs)
```

```
##
## Call:
## summary.diff.resamples(object = difs)
##
```

```
## p-value adjustment: bonferroni
## Upper diagonal: estimates of the difference
## Lower diagonal: p-value for H0: difference = 0
##
## MAE
##         GBM       GLM        LM         KNN        RF         SVM
## GBM               -3.182e-03 -3.102e-03 -1.944e-02 -4.791e-03 -2.876e-03
## GLM     1.000000              7.935e-05 -1.626e-02 -1.609e-03  3.060e-04
## LM      1.000000  1.000000              -1.634e-02 -1.688e-03  2.267e-04
## KNN     1.649e-09 2.242e-08  1.859e-10              1.465e-02  1.657e-02
## RF      0.181958  1.000000   1.000000   7.428e-07              1.915e-03
## SVM     1.000000  1.000000   1.000000   1.115e-09  1.000000
## SVMR    1.000000  0.008262   0.008830   8.198e-12  0.018688   0.007091
## LASSO   0.431041  1.000000   1.000000   5.467e-11  1.000000   1.000000
## XGBoost 1.000000  1.000000   1.000000   4.202e-08  1.000000   1.000000
## GLMNET  1.000000  1.000000   1.000000   1.649e-07  1.000000   1.000000
##         SVMR       LASSO      XGBoost    GLMNET
## GBM      2.863e-03 -3.178e-03 -3.194e-03 -3.085e-03
## GLM      6.044e-03  3.829e-06 -1.246e-05  9.674e-05
## LM       5.965e-03 -7.552e-05 -9.181e-05  1.739e-05
## KNN      2.230e-02  1.626e-02  1.625e-02  1.636e-02
## RF       7.653e-03  1.613e-03  1.596e-03  1.706e-03
## SVM      5.738e-03 -3.022e-04 -3.185e-04 -2.093e-04
## SVMR               -6.041e-03 -6.057e-03 -5.948e-03
## LASSO   0.028198              -1.629e-05  9.291e-05
## XGBoost 0.064571   1.000000               1.092e-04
## GLMNET  0.072358   1.000000   1.000000
##
## RMSE
##         GBM       GLM        LM         KNN        RF         SVM
## GBM               -0.0023093 -0.0024976 -0.0239476 -0.0105305 -0.0028760
## GLM     1.00000              -0.0001883 -0.0216383 -0.0082212 -0.0005668
## LM      1.00000   1.00000               -0.0214500 -0.0080329 -0.0003784
## KNN     1.249e-05 3.445e-05  2.264e-07              0.0134171  0.0210715
## RF      0.20725   1.00000    0.94431    0.01487               0.0076544
## SVM     1.00000   1.00000    1.00000    3.941e-06  1.00000
## SVMR    1.00000   1.00000    1.00000    3.156e-07  0.30975    1.00000
## LASSO   1.00000   1.00000    1.00000    2.270e-07  0.60154    1.00000
## XGBoost 1.00000   1.00000    1.00000    1.739e-05  1.00000    1.00000
## GLMNET  1.00000   1.00000    1.00000    2.529e-05  0.72712    1.00000
##         SVMR       LASSO      XGBoost    GLMNET
## GBM      0.0013622 -0.0027633 -0.0050120 -0.0023810
## GLM      0.0036715 -0.0004540 -0.0027027 -0.0000717
## LM       0.0038598 -0.0002657 -0.0025144  0.0001166
## KNN      0.0253098  0.0211843  0.0189355  0.0215666
## RF       0.0118927  0.0077672  0.0055184  0.0081495
## SVM      0.0042383  0.0001128 -0.0021360  0.0004951
## SVMR               -0.0041255 -0.0063742 -0.0037432
## LASSO   1.00000               -0.0022487  0.0003823
## XGBoost 1.00000    1.00000                0.0026310
## GLMNET  1.00000    1.00000    1.00000
##
## Rsquared
##         GBM       GLM        LM         KNN        RF         SVM
```

```
## GBM                3.448e-03  3.971e-03  3.536e-02  1.234e-02  4.235e-03
## GLM     1.0000000             5.233e-04  3.191e-02  8.892e-03  7.870e-04
## LM      1.0000000 1.0000000              3.139e-02  8.369e-03  2.637e-04
## KNN     9.090e-06 1.162e-05  2.087e-05            -2.302e-02 -3.112e-02
## RF      1.0000000 1.0000000  1.0000000  0.0038673            -8.105e-03
## SVM     1.0000000 1.0000000  1.0000000  8.237e-06  1.0000000
## SVMR    1.0000000 1.0000000  1.0000000  3.473e-08  0.5364576  1.0000000
## LASSO   1.0000000 1.0000000  1.0000000  2.333e-06  1.0000000  1.0000000
## XGBoost 1.0000000 1.0000000  1.0000000  0.0001246  1.0000000  1.0000000
## GLMNET  1.0000000 1.0000000  1.0000000  0.0006465  1.0000000  1.0000000
##         SVMR       LASSO      XGBoost    GLMNET
## GBM     -2.711e-03  4.379e-03  7.397e-03  4.066e-03
## GLM     -6.159e-03  9.314e-04  3.949e-03  6.176e-04
## LM      -6.682e-03  4.081e-04  3.426e-03  9.428e-05
## KNN     -3.807e-02 -3.098e-02 -2.796e-02 -3.129e-02
## RF      -1.505e-02 -7.961e-03 -4.943e-03 -8.274e-03
## SVM     -6.946e-03  1.444e-04  3.162e-03 -1.694e-04
## SVMR                7.090e-03  1.011e-02  6.776e-03
## LASSO   1.0000000              3.018e-03 -3.138e-04
## XGBoost 1.0000000  1.0000000             -3.332e-03
## GLMNET  1.0000000  1.0000000  1.0000000
```

```r
compare_models(modelo_svmRadial, modelo_glmnet)
```

```
##
##  One Sample t-test
##
## data:  x
## t = -1.0082, df = 24, p-value = 0.3234
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  -0.011405871  0.003919511
## sample estimates:
##   mean of x
## -0.00374318
```

Error de test

Utilizamos extractPrediction() para obtener las predicciones de una lista de modelos, que devuelve tanto para las observaciones de entrenamiento como para las de test.

```r
predicciones <- extractPrediction(
                  models = modelos,
                  testX = dsTrain.CV[, -1],
                  testY = dsTrain.CV$SalePrice
                  )

metricas_tipo <- predicciones %>%
                 group_by(object, dataType) %>%
                 summarise(RMSE = RMSE(pred, obs)) %>%
                 rename(modelo = object)

metricas_tipo
```

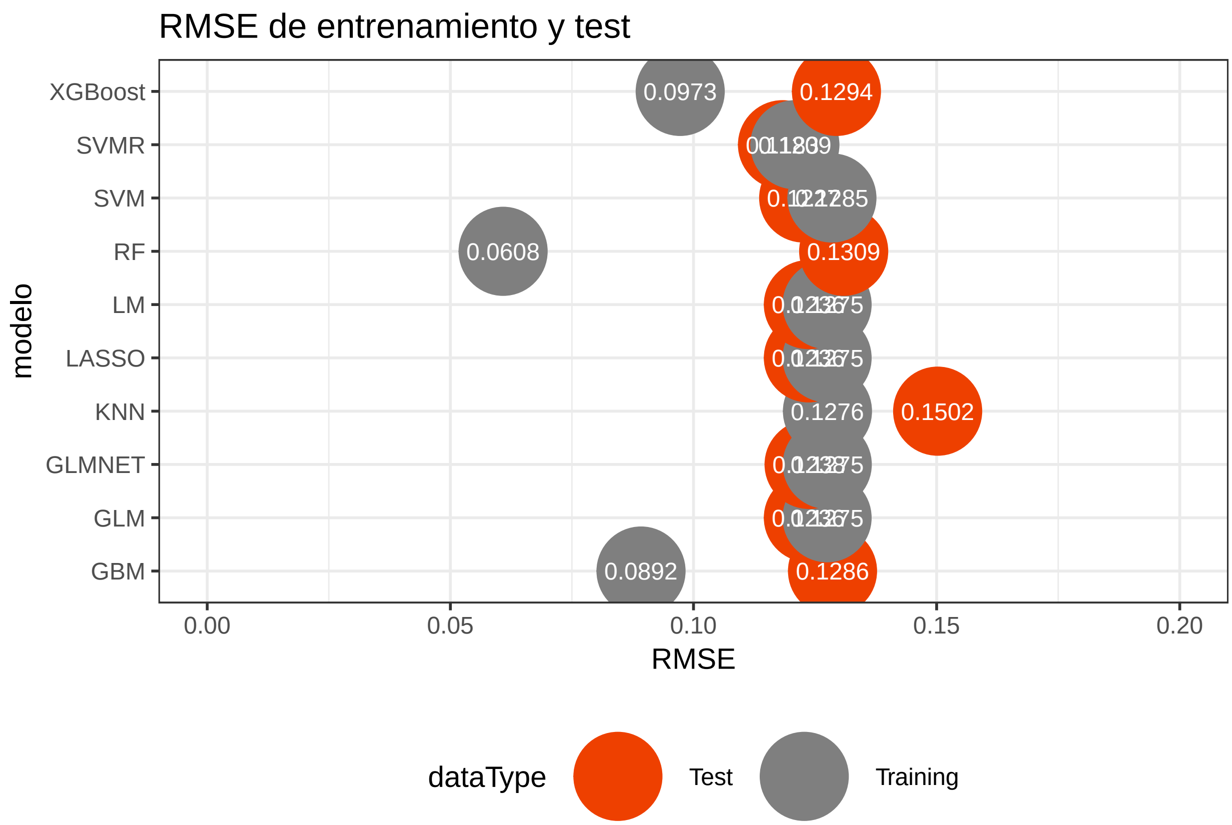
```
## # A tibble: 20 x 3
## # Groups:   modelo [10]
##    modelo  dataType   RMSE
##    <fct>   <fct>     <dbl>
##  1 GBM     Test     0.129
##  2 GBM     Training 0.0892
##  3 GLM     Test     0.124
##  4 GLM     Training 0.127
##  5 GLMNET  Test     0.124
##  6 GLMNET  Training 0.128
##  7 KNN     Test     0.150
##  8 KNN     Training 0.128
##  9 LASSO   Test     0.124
## 10 LASSO   Training 0.127
## 11 LM      Test     0.124
## 12 LM      Training 0.127
## 13 RF      Test     0.131
## 14 RF      Training 0.0608
## 15 SVM     Test     0.123
## 16 SVM     Training 0.128
## 17 SVMR    Test     0.118
## 18 SVMR    Training 0.121
## 19 XGBoost Test     0.129
## 20 XGBoost Training 0.0973
```

```
metricas <- metricas_tipo %>%
  spread(key = dataType, RMSE) %>%
  arrange(Test)

metricas
```

```
## # A tibble: 10 x 3
## # Groups:   modelo [10]
##    modelo   Test Training
##    <fct>   <dbl>    <dbl>
##  1 SVMR    0.118   0.121
##  2 SVM     0.123   0.128
##  3 LASSO   0.124   0.127
##  4 GLM     0.124   0.127
##  5 LM      0.124   0.127
##  6 GLMNET  0.124   0.128
##  7 GBM     0.129   0.0892
##  8 XGBoost 0.129   0.0973
##  9 RF      0.131   0.0608
## 10 KNN     0.150   0.128
```

```
ggplot(data = metricas_tipo,
       aes(x = modelo, y = RMSE,
           color = dataType, label = round(RMSE, 4))) +
  geom_point(size = 15) +
  scale_color_manual(values = c("orangered2", "gray50")) +
  geom_text(color = "white", size = 3) +
  scale_y_continuous(limits = c(0, 0.2)) +
```

```r
  coord_flip() +
  labs(title = "RMSE de entrenamiento y test",
       x = "modelo") +
  theme_bw() +
  theme(legend.position = "bottom")
```



Métricas globales

Guardamos resultados juntos con los ya existentes

```r
metricas
```

```
## # A tibble: 10 x 3
## # Groups:   modelo [10]
##    modelo   Test Training
##    <fct>   <dbl>    <dbl>
##  1 SVMR    0.118   0.121
##  2 SVM     0.123   0.128
##  3 LASSO   0.124   0.127
##  4 GLM     0.124   0.127
##  5 LM      0.124   0.127
##  6 GLMNET  0.124   0.128
##  7 GBM     0.129   0.0892
```

```
##   8 XGBoost 0.129   0.0973
##   9 RF      0.131   0.0608
## 10 KNN     0.150   0.128
```

```r
# Cargamos metricas anteriores
if (file.exists('./F04_Modelos/F04_200_metricas.RData')){
  load('./F04_Modelos/F04_200_metricas.RData')
}

metricas <- mutate(metricas
                   ,OrigenF2 = strOrigenF2
                   ,OrigenF3 = strOrigenF3
                   ,fch = Sys.Date())

if (file.exists('./F04_Modelos/F04_200_metricas.RData')){
  metricasGuardadas <- union_all(metricasGuardadas,metricas)
} else{
  metricasGuardadas <- metricas
}
metricasGuardadas <- as.data.frame(metricasGuardadas)
save(metricasGuardadas, file = './F04_Modelos/F04_200_metricas.RData')

# Top 10
head(arrange(metricasGuardadas,Test),10)
```

```
##     modelo      Test   Training                OrigenF2
## 1     SVMR 0.1183000 0.12085139 F02_03_dsDataAll_Recipe
## 2      SVM 0.1226602 0.12846428 F02_03_dsDataAll_Recipe
## 3    LASSO 0.1236241 0.12747086 F02_03_dsDataAll_Recipe
## 4      GLM 0.1236241 0.12747086 F02_03_dsDataAll_Recipe
## 5       LM 0.1236241 0.12747086 F02_03_dsDataAll_Recipe
## 6   GLMNET 0.1237693 0.12750407 F02_03_dsDataAll_Recipe
## 7      GBM 0.1285882 0.08920917 F02_03_dsDataAll_Recipe
## 8  XGBoost 0.1293966 0.09726788 F02_03_dsDataAll_Recipe
## 9       RF 0.1308879 0.06084568 F02_03_dsDataAll_Recipe
## 10     KNN 0.1502120 0.12755013 F02_03_dsDataAll_Recipe
##                                       OrigenF3        fch
## 1  F03_11_dsDataSelVar_rfe_MejorRendimiento_top18 2019-09-22
## 2  F03_11_dsDataSelVar_rfe_MejorRendimiento_top18 2019-09-22
## 3  F03_11_dsDataSelVar_rfe_MejorRendimiento_top18 2019-09-22
## 4  F03_11_dsDataSelVar_rfe_MejorRendimiento_top18 2019-09-22
## 5  F03_11_dsDataSelVar_rfe_MejorRendimiento_top18 2019-09-22
## 6  F03_11_dsDataSelVar_rfe_MejorRendimiento_top18 2019-09-22
## 7  F03_11_dsDataSelVar_rfe_MejorRendimiento_top18 2019-09-22
## 8  F03_11_dsDataSelVar_rfe_MejorRendimiento_top18 2019-09-22
## 9  F03_11_dsDataSelVar_rfe_MejorRendimiento_top18 2019-09-22
## 10 F03_11_dsDataSelVar_rfe_MejorRendimiento_top18 2019-09-22
```