House Prices: Advanced Regression Techniques

Stephane Mendez, Erick Mendoza, Omar Hernández & Luis Enrique Martínez

26/2/2019

Introducción

La base de datos *Ames Housing dataset* contiene 79 variables que se pretenden utilizar para predecir el precio de las casas. La métrica de error para evaluar el desempeño de los modelos será error cuadrático medio (RMSE) y las técnicas de modelo serán de regresión dado que el precio es una variable continua.

Análisis exploratorio y tratamiento de datos faltantes

Realizando el análisis exploratorio de las variables nos dimos cuenta que las siguientes variables deben ser factor y no numéricas:

- **MSSubclass**: Estilo del hogar en venta
- **MoSold**: Mes en que se realizó la venta
- **YrSold**: Año en que se realizó la venta

```r
train.data$MSSubClass<-as.factor(train.data$MSSubClass)
train.data$MoSold<-as.factor(train.data$MoSold)
train.data$YrSold<-as.factor(train.data$YrSold)
##Lo mismo en test
test.data$MSSubClass<-as.factor(test.data$MSSubClass)
test.data$MoSold<-as.factor(test.data$MoSold)
test.data$YrSold<-as.factor(test.data$YrSold)
##Validamos--->están ok
sapply(train.data,class)
```

```
##            Id    MSSubClass      MSZoning   LotFrontage       LotArea
##     "integer"      "factor"      "factor"     "integer"     "integer"
##        Street         Alley      LotShape   LandContour     Utilities
##      "factor"      "factor"      "factor"      "factor"      "factor"
##     LotConfig     LandSlope  Neighborhood    Condition1    Condition2
##      "factor"      "factor"      "factor"      "factor"      "factor"
##      BldgType    HouseStyle   OverallQual   OverallCond     YearBuilt
##      "factor"      "factor"     "integer"     "integer"     "integer"
##  YearRemodAdd     RoofStyle      RoofMatl   Exterior1st   Exterior2nd
##     "integer"      "factor"      "factor"      "factor"      "factor"
##    MasVnrType    MasVnrArea     ExterQual     ExterCond    Foundation
##      "factor"     "integer"      "factor"      "factor"      "factor"
##      BsmtQual      BsmtCond  BsmtExposure  BsmtFinType1    BsmtFinSF1
##      "factor"      "factor"      "factor"      "factor"     "integer"
##  BsmtFinType2    BsmtFinSF2     BsmtUnfSF   TotalBsmtSF       Heating
##      "factor"     "integer"     "integer"     "integer"      "factor"
##     HeatingQC    CentralAir    Electrical     X1stFlrSF     X2ndFlrSF
##      "factor"      "factor"      "factor"     "integer"     "integer"
##  LowQualFinSF     GrLivArea  BsmtFullBath  BsmtHalfBath      FullBath
##     "integer"     "integer"     "integer"     "integer"     "integer"
##      HalfBath  BedroomAbvGr  KitchenAbvGr   KitchenQual  TotRmsAbvGrd
##     "integer"     "integer"     "integer"      "factor"     "integer"
```

```
##    Functional    Fireplaces   FireplaceQu    GarageType   GarageYrBlt
##      "factor"     "integer"      "factor"      "factor"     "integer"
##  GarageFinish    GarageCars    GarageArea    GarageQual    GarageCond
##      "factor"     "integer"     "integer"      "factor"      "factor"
##    PavedDrive    WoodDeckSF   OpenPorchSF EnclosedPorch    X3SsnPorch
##      "factor"     "integer"     "integer"     "integer"     "integer"
##   ScreenPorch      PoolArea        PoolQC         Fence   MiscFeature
##     "integer"     "integer"      "factor"      "factor"      "factor"
##       MiscVal        MoSold        YrSold      SaleType SaleCondition
##     "integer"      "factor"      "factor"      "factor"      "factor"
##     SalePrice
##     "integer"
```

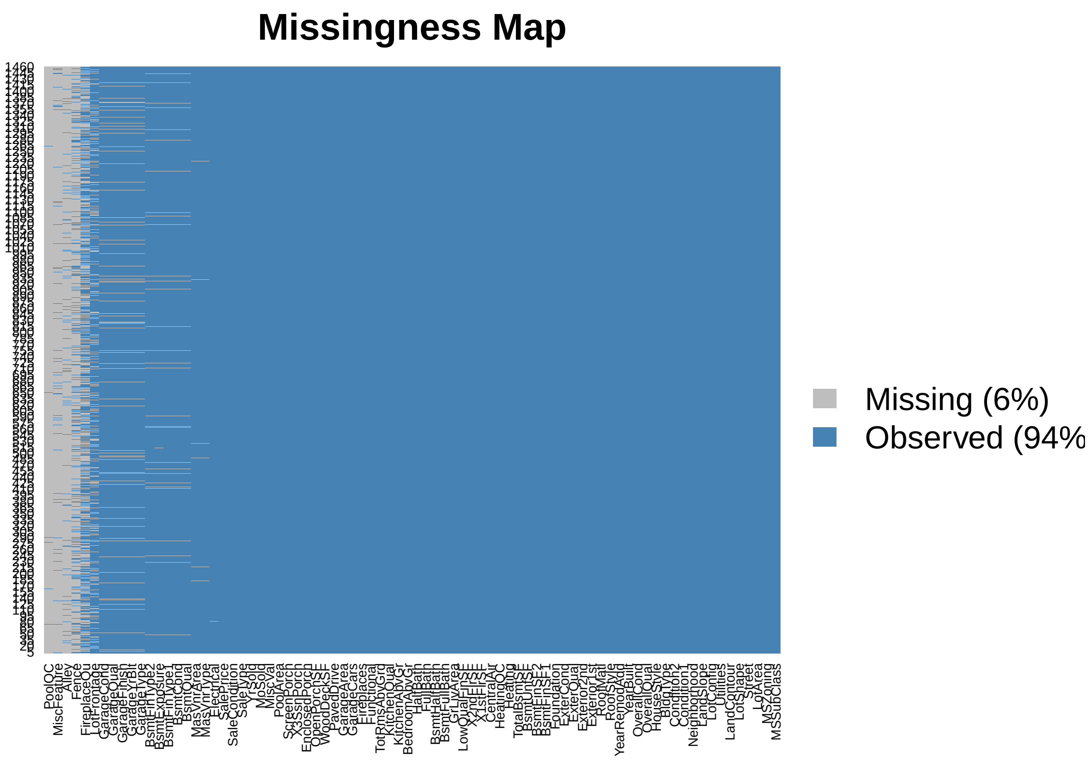
```r
sapply(test.data,class)
```

```
##            Id    MSSubClass      MSZoning   LotFrontage       LotArea
##     "integer"      "factor"      "factor"     "integer"     "integer"
##        Street         Alley      LotShape   LandContour     Utilities
##      "factor"      "factor"      "factor"      "factor"      "factor"
##     LotConfig     LandSlope  Neighborhood    Condition1    Condition2
##      "factor"      "factor"      "factor"      "factor"      "factor"
##      BldgType    HouseStyle   OverallQual   OverallCond     YearBuilt
##      "factor"      "factor"     "integer"     "integer"     "integer"
##  YearRemodAdd     RoofStyle      RoofMatl   Exterior1st   Exterior2nd
##     "integer"      "factor"      "factor"      "factor"      "factor"
##    MasVnrType    MasVnrArea     ExterQual     ExterCond    Foundation
##      "factor"     "integer"      "factor"      "factor"      "factor"
##      BsmtQual      BsmtCond  BsmtExposure  BsmtFinType1    BsmtFinSF1
##      "factor"      "factor"      "factor"      "factor"     "integer"
##  BsmtFinType2    BsmtFinSF2     BsmtUnfSF   TotalBsmtSF       Heating
##      "factor"     "integer"     "integer"     "integer"      "factor"
##     HeatingQC    CentralAir    Electrical     X1stFlrSF     X2ndFlrSF
##      "factor"      "factor"      "factor"     "integer"     "integer"
##  LowQualFinSF     GrLivArea  BsmtFullBath  BsmtHalfBath      FullBath
##     "integer"     "integer"     "integer"     "integer"     "integer"
##      HalfBath  BedroomAbvGr  KitchenAbvGr   KitchenQual  TotRmsAbvGrd
##     "integer"     "integer"     "integer"      "factor"     "integer"
##    Functional    Fireplaces   FireplaceQu    GarageType   GarageYrBlt
##      "factor"     "integer"      "factor"      "factor"     "integer"
##  GarageFinish    GarageCars    GarageArea    GarageQual    GarageCond
##      "factor"     "integer"     "integer"      "factor"      "factor"
##    PavedDrive    WoodDeckSF   OpenPorchSF EnclosedPorch    X3SsnPorch
##      "factor"     "integer"     "integer"     "integer"     "integer"
##   ScreenPorch      PoolArea        PoolQC         Fence   MiscFeature
##     "integer"     "integer"      "factor"      "factor"      "factor"
##       MiscVal        MoSold        YrSold      SaleType SaleCondition
##     "integer"      "factor"      "factor"      "factor"      "factor"
```

Tratamiento de datos faltantes

De la base de entrenamiento analizamos el % de missings por variable.

```r
missmap(train.data[-1], col=c('grey', 'steelblue'), y.cex=0.4, x.cex=0.4)
```



Se observa que el 94% de la base está poblada adecuadamente y las variables con datos faltantes en más de la mitad de base son:

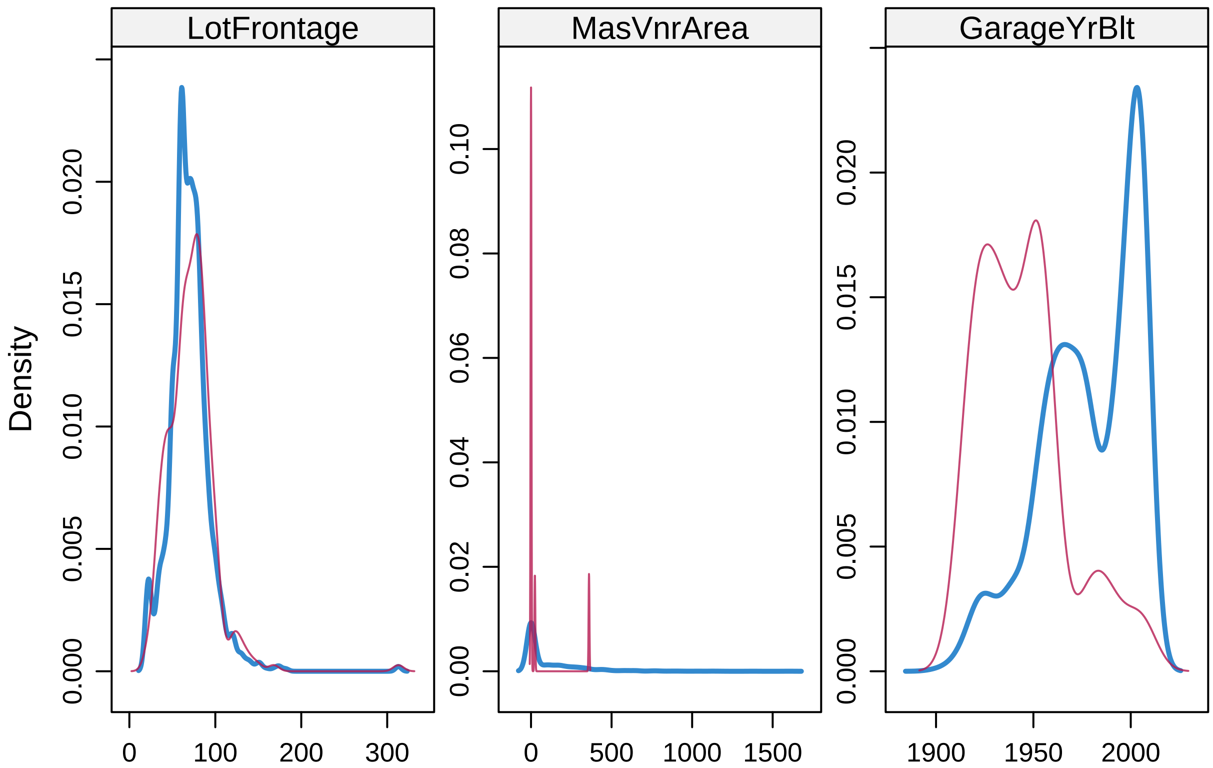
- PoolQc
- MiscFeature
- Alley
- Fence

Por lo tanto, no es adecuado considerarlos como input del modelo. Para el resto de las variables se utiliza árboles de clasificación y regresión (CART) en el manejo de datos faltantes.

```
##Excluimos las variables que tienen muchos missings
exclude <- c('PoolQC', 'MiscFeature', 'Alley', 'Fence')
include <- setdiff(names(train.data), exclude)
train.data <- train.data[include]
##Mediante árboles de clasificación y regresión imputamos los datos faltantes
imp.train.data <- mice(train.data, m=1, method='cart', printFlag=FALSE)
```

```
## Warning: Number of logged events: 75
```

```
densityplot(imp.train.data )
```




```r
train.complete <- mice::complete(imp.train.data)
#Confirmamos que no haya missings
sum(sapply(train.complete, function(x) { sum(is.na(x)) }))
```

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

```r
sum(sapply(train.data, function(x) { sum(is.na(x)) }))
```

```
## [1] 1558
```

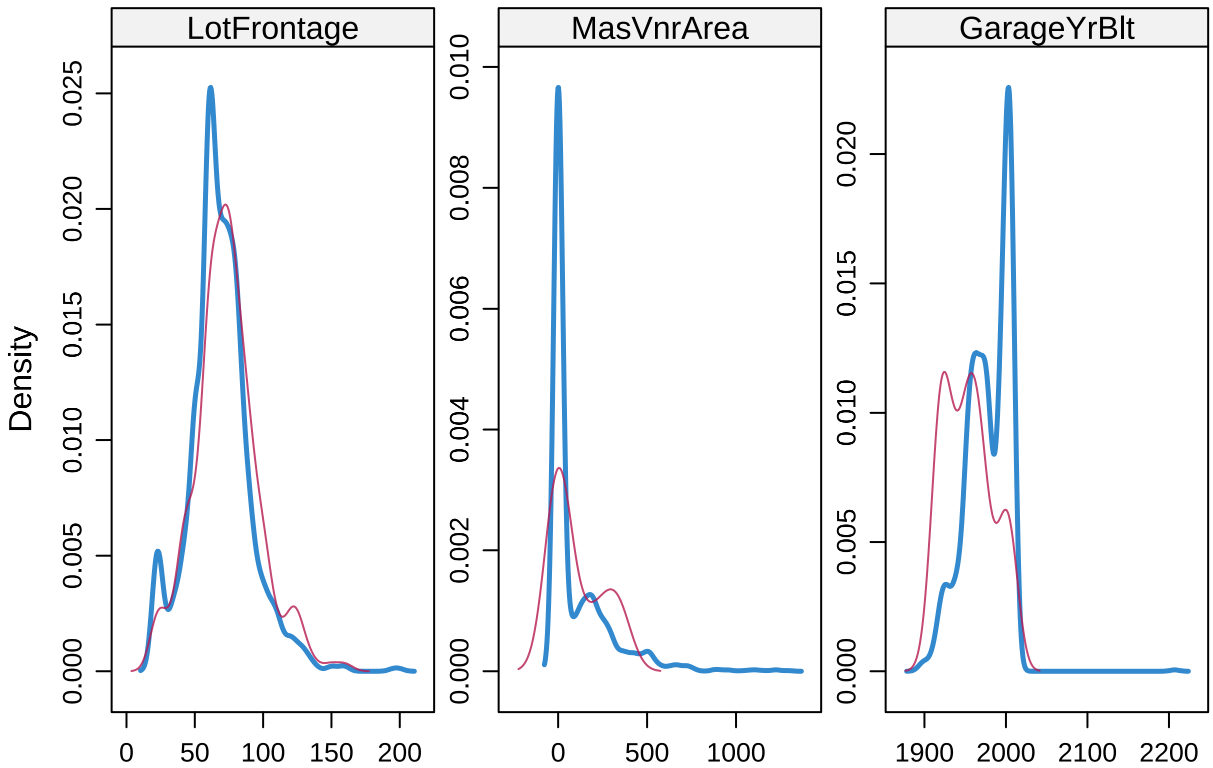
Se observa que pasamos de tener 1558 registros faltantes a 0. Adicional se valida que en las variables continuas la imputación siga la misma distribución que los datos reales.

Realizamos la misma metodología de tratamiento de datos faltantes para la base de prueba.

```r
##Excluimos las variables que tienen muchos missings
exclude <- c('PoolQC', 'MiscFeature', 'Alley', 'Fence')
include <- setdiff(names(test.data), exclude)
test.data <- test.data[include]
##Mediante árboles de clasificación y regresión imputamos los datos faltantes
imp.test.data <- mice(test.data, m=1, method='cart', printFlag=FALSE)
```

```
## Warning: Number of logged events: 141
```

```r
densityplot(imp.test.data )
```




```
test.complete <- mice::complete(imp.test.data)
```

Confirmamos que en la base de prueba tampoco se mueve la distribución en los datos imputados.

Ingeniería de datos

Intentamos mejorar el rendimiento del modelo teniendo en cuenta la interacción bidireccional entre variables continuas. Además, describimos dicha interacción con la ayuda del suavizado B-spline.

```
##Funciones de ayuda
MMContVars <- function(x, y, x.test, y.test){
  ##Calculamos el producto tensor de los coeficientes de B-spline de x y y
  bc.x <- BSplineCoeff(x, x.lim, K1)
  bc.y <- BSplineCoeff(y, y.lim, K2)
  bc.x.test <- BSplineCoeff(x.test, x.lim, K1)
  bc.y.test <- BSplineCoeff(y.test, y.lim, K2)

  int.matrix <- cbind(data.matrix(bc.x), data.matrix(bc.y))
  int.matrix.test <- cbind(data.matrix(bc.x.test), data.matrix(bc.y.test))
  model.matrix <- t(apply(int.matrix, 1, MatrixForModel, kx=K1, ky=K2))
  model.matrix.test <- t(apply(int.matrix.test, 1, MatrixForModel, kx=K1, ky=K2))

  return(list(model.matrix, model.matrix.test))
}

TransofrmVariables <- function(x, x.test){
  #Transformamos x para que su dominio sea [0,1],
  #Penalizamos los valores grandes de x
  m <- median(x[x!=0])
```

```
  lambda <- (-1 / m) * (log(0.5))
  x.test <- (1 - exp(-lambda*x.test))
  x <- (1 - exp(-lambda*x))
  return(list(x, x.test))
}

BSplineCoeff <- function(z, x, control_points_num){
  ## B-spline coeficientes del vector z en el intervalo min(x) a max(x).
  num_kps <- control_points_num - spline.power
  lenx <- max(x) - min(x)
  cstep <- lenx / (num_kps + 1)
  kps <- seq((min(x) + cstep), (max(x) - cstep), by = cstep)
  a <- bs(as.matrix(z), knots = kps, Boundary.knots = c(min(x), max(x)),
          degree = spline.power)
  a <- as.matrix(a[ , 2 : control_points_num])
  a <- cbind(data.frame(1 - rowSums(a)), a)
  return(a)
}

MatrixForModel <- function(x, kx, ky){
  ## "2D tensor producto de x and y
  x <- data.frame(t(x))
  len1 <- kx
  len2 <- kx+ky
  a <- x[, 1 : len1]
  b <- x[, (len1+1): len2]
  c <- as.vector(t(as.matrix(a)) %*% as.matrix(b))
  return(c)
}

##Toda la data
all_data <- rbind(select(train.complete, MSSubClass:SaleCondition),
                  select(test.complete, MSSubClass:SaleCondition))

# transformamos el precio en logaritmo
train.complete$SalePrice <- log(train.complete$SalePrice + 1)

# features numéricos logaritmos
feature_classes <- sapply(names(all_data),function(x){class(all_data[[x]])})
numeric_feats <- names(feature_classes[feature_classes != "factor"])

# sesgo de esos features numéricos
skewed_feats <- sapply(numeric_feats,function(x){skewness(all_data[[x]], na.rm=TRUE)})

# keep only features that exceed a threshold for skewness
skewed_feats <- skewed_feats[skewed_feats > 0.75]

# get names of categorical features
categorical_feats <- names(feature_classes[feature_classes == "factor"])

# use caret dummyVars function for hot one encoding for categorical features
dummies <- dummyVars(~., all_data[categorical_feats])
categorical_1_hot <- predict(dummies, all_data[categorical_feats])
```

```
categorical_1_hot[is.na(categorical_1_hot)] <- 0  #for any level that was NA, set to zero

# for any missing values in numeric features, impute mean of that feature
numeric_df <- all_data[numeric_feats]

for (x in numeric_feats) {
  mean_value <- mean(train.complete[[x]], na.rm = TRUE)
  all_data[[x]][is.na(all_data[[x]])] <- mean_value
}

# reconstruct all_data with pre-processed data
all_data <- cbind(all_data[numeric_feats], categorical_1_hot)

# create data for training and test
X_train_m1 <- all_data[1:nrow(train.complete),]
X_test_m1 <- all_data[(nrow(train.complete)+1):nrow(all_data),]

# transform excessively skewed features with log(x + 1)
for(x in names(skewed_feats)) {
  X_train_m1[,x] <- log(X_train_m1[,x] + 1)
  X_test_m1[,x] <- log(X_test_m1[,x] + 1)
}

## create data for training and test
X_train_m2 <- all_data[1:nrow(train.complete),]
X_test_m2 <- all_data[(nrow(train.complete)+1):nrow(all_data),]

## Number of knots for 2D B-spline
K1 <- K2 <- 4

## Spline power.
spline.power <- 3

## B-spline domain.
x.lim <- c(0, 1)
y.lim <- c(0, 1)

## Define variable names for two-way interaction.
new_vars <- data.frame(matrix(0, 5, 2))
new_vars[1,] <- c("LotArea", "BsmtFinSF1")
new_vars[2,] <- c("LotArea", "TotalBsmtSF")
new_vars[3,] <- c("MasVnrArea", "BsmtFinSF1")
new_vars[4,] <- c("MasVnrArea", "X2ndFlrSF")
new_vars[5,] <- c("X1stFlrSF", "GrLivArea")

## Create and add new variables to our dataset.
for (i in 1:nrow(new_vars)){

  ## Transform variables.
  bf <- TransofrmVariables(all_data[1:nrow(train.complete), new_vars[i, 1]], all_data[(nrow(train.compl
  x <- bf[[1]]
  x_test <- bf[[2]]
  bf <-TransofrmVariables(all_data[1:nrow(train.complete), new_vars[i, 2]], all_data[(nrow(train.comple
```

```r
  y <- bf[[1]]
  y_test <- bf[[2]]

  ## Create new variable which represented by 16 (K1*K2) 2D B-spline coefficients.
  SD <- MMContVars(x, y, x_test, y_test)

  nm <- c()
  for (j in 1:(K1*K2)) {nm <- c(nm, paste(new_vars[i, 1], "&", new_vars[i, 2], "_", j, sep=""))}
  d1 <- data.frame(SD[[1]])
  d2 <- data.frame(SD[[2]])
  names(d1) <- names(d2) <- nm

  ## Add new new variable to dataset.
  X_train_m2 <- cbind(X_train_m2, d1)
  X_test_m2 <- cbind(X_test_m2, d2)
}

# transform excessively skewed features with log(x + 1)
for(x in names(skewed_feats)) {
  X_train_m2[,x] <- log(X_train_m2[,x] + 1)
  X_test_m2[,x] <- log(X_test_m2[,x] + 1)
}

## Delete additive interaction of variables which were used for new features.
new_vars_names <- unique(c(new_vars[,1], new_vars[,2]))
x_train_var_names <- names(X_train_m2)
x_train_var_names <- x_train_var_names[!(x_train_var_names %in% new_vars_names)]

X_train_m2 <- X_train_m2[x_train_var_names]
X_test_m2 <- X_test_m2[x_train_var_names]
y<-train.complete$SalePrice
X_train_m2<-cbind(X_train_m2,y)
```

Con esta ingeniería de datos pasamos de tener 79 variables explicativas a tener 377 que presuponen un mayor poder predictivo.

Desarrollo de modelos

Con el fin de obtener la mejor predicción del precio de las casas probamos 3 diferentes modelos y seleccionaremos el que tenga el menor error cuadrático medio.

Para evaluar el desempeñode los modelos, la base de entrenamiento la dividiremos en 70% para desarrollo de modelo y 30% para la validación y cálculo del RMSE.

```r
set.seed(123)

index_train <- sample(1:1460, size = 1022, replace = F)

Devdata <- X_train_m2[index_train,]
Itvdata <- X_train_m2[-index_train,]
```

Regresión Lasso

```r
Control_train <- trainControl(method="repeatedcv",
                              number=5,
                              repeats=5,
                              verboseIter=FALSE)

## entrenamos modelo
set.seed(123)

model_lasso <- train(x=Devdata, y=Devdata$y,
                method="glmnet",
                metric="RMSE",
                maximize=FALSE,
                trControl=Control_train,
                tuneGrid=expand.grid(alpha=1,  # Lasso regression
                                    lambda=c(0.3, 0.1,0.05,0.01,seq(0.009,0.001, -0.001),
                                             0.00075,0.0005,0.0001)))


predict_lasso<-predict(model_lasso, Itvdata)
head(predict_lasso)
```

```
##        2        5        7       21       22       25
## 12.10660 12.41747 12.61687 12.67308 11.85039 11.94709
```

```r
##RMSE en test
mean(sum((predict_lasso - Itvdata$y)^2))
```

```
## [1] 0.05918723
```

El error cuadrático medio de la regresión es: 0.05918723

GBM

```r
set.seed(1)
Control_traingbm <- trainControl(## 10-fold CV
  method = "repeatedcv",
  number = 10,
  ## repeated ten times
  repeats = 10)
model_gbm<- train(y~., data= Devdata,method="gbm", trControl= Control_traingbm, verbose=FALSE)
summary(model_gbm)

predict_gbm<-predict(model_gbm, Itvdata)

#RMSE Test
mean(sum((predict_gbm - Itvdata$y)^2))
```

El error cuadrático medio de GBM: 0.04317343

Dado que nuestra métrica de selección de modelo es el error cuadrático medio, consideramos que el modelo de GBM es el adecuado para predecir el precio de las casas.