Universidad Autónoma de Yucatán

Maestría en Ciencias de la Computación

Métodos Estadísticos de Machine Learning

Proyecto Final

Autor: Mario Herrera Almira

27 de mayo del 2023

Índice

Enunciado

Este proyecto tiene un valor de 20 puntos sobre la calificación final del curso, se debe entregar a más tardar el lunes 29 de mayo de 2023, utilizando la plataforma Enlinea2 para subir el reporte (en formato pdf), y enviar por email (jbatun@correo.uad.mx) el script r utilizado.

En la plataforma Enlinea 2 se encuentran los archivos:

- **Datos**, que tiene la información sobre la cual se aplicaran técnicas de construcción de modelos de regresión
- **data_description**, que tiene una descripción de las variables que conforman la base de datos anterior.

Se debe escribir un reporte que incluya la información necesaria y que explique en forma clara como se realizaron las siguientes actividades, así como presentar la información que se pide en alguna de esas actividades.

1. Un análisis descriptivo de los datos, que incluya:
    a. Identificación de la variable dependiente y la independiente
    b. Como se procedió para la limpieza de la base de datos
    c. Determinar si es necesario hacer una normalización de la base de datos, y en caso de aplicarla, mencionar que tipo de normalización se realizó
2. (Enfoque train-test validation) Particiona la muestra en forma 80-20 de manera aleatoria. El 80% corresponde a la muestra de entrenamiento. Utilizando la función regsubsets determina el mejor modelo bajo el criterio del MSE. Presenta la ecuación del modelo, y comenta acerca de si cada variable es significativa en presencia de las restantes. **(5 puntos)**
3. Utilizando el enfoque Cross Validation y la función regsubsets, determina el mejor modelo bajo el criterio del MSE. **(5 puntos)**
4. (Opcional, 10 puntos para alguno de los proyectos 1 y 2, recuperación) Utilizando el enfoque Cross Validation y la función regsubsets, determina el mejor modelo bajo un criterio diferente al MSE.
5. Utilizando Ridge Regression y Lasso, en combinación con la validación cruzada, determina dos modelos de regresión. Debes explicar con mucho detalle el criterio para la elección del número de folds y el valor del parámetro de penalización **(10 puntos)**
6. Si el script r está muy bien documentado, completo y bien organizado, se pueden agregar 5 puntos de recuperación, para alguno de los proyectos anteriores.

Respuesta

La base de datos describe las características de viviendas que fueron vendidas a lo largo de un periodo de 5 años y el precio al que fueron vendidas. El objetivo de este trabajo es limpiar la base de datos y utilizar varias técnicas para poder crear modelos que permitan predecir el precio a que se va a vender una vivienda basado en sus características.

1. Análisis descriptivo de los datos

1a. Identificación de la variable dependiente y las independientes.

La base de datos cuenta inicialmente con 81 variables de las cuales la última es la variable que se desea predecir (variable dependiente) y las otras 80 son variables independientes que describen características de las viviendas.

Variable dependiente: SalePrice (precio en el que se vende la vivienda).

Variables independientes: Las otras 80 variables.

1b. Cómo se procedió para la limpieza de la base de datos.

Tratando los valores NA:

La base de datos contiene inicialmente 34 columnas con valores NA, se pueden observar los nombres de las columnas y la cantidad de valores NA que contiene cada una en la siguiente imagen:

```
      PoolQC  MiscFeature        Alley        Fence  FireplaceQu
        2909         2814         2721         2348         1420
 LotFrontage  GarageYrBlt GarageFinish   GarageQual   GarageCond
         486          159          159          159          159
  GarageType     BsmtCond BsmtExposure     BsmtQual BsmtFinType2
         157           82           82           81           80
BsmtFinType1   MasVnrType   MasVnrArea     MSZoning    Utilities
          79           24           23            4            2
BsmtFullBath BsmtHalfBath   Functional  Exterior1st  Exterior2nd
           2            2            2            1            1
  BsmtFinSF1   BsmtFinSF2    BsmtUnfSF  TotalBsmtSF   Electrical
           1            1            1            1            1
 KitchenQual   GarageCars   GarageArea     SaleType
           1            1            1            1
> cat('Hay', length(NAcol), 'columnas con valores NA')
Hay 34 columnas con valores NA>
```

Pero esta gran cantidad de valores NA se deben a que en muchos casos un valor NA no significa la ausencia del dato sino solamente que la vivienda no tiene esa característica. Por ejemplo, los valores NA de la columna PoolQC significan que la vivienda no tiene piscina y no que no se conozca el dato. En todos los casos donde sucede algo similar simplemente se cambiaron los NA por otro valor, en su mayoría por “None” para las variables categóricas y cero para las variables numéricas. En la siguiente imagen se pueden observar todas las variables cuyos valores NA fueron sustituidos por otro valor:

```
all$Alley[is.na(all$Alley)] <- 'None'
all$BsmtQual[is.na(all$BsmtQual)] <- 'None'
all$BsmtCond[is.na(all$BsmtCond)] <- 'None'
all$BsmtExposure[is.na(all$BsmtExposure)] <- 'None'
all$BsmtFinType1[is.na(all$BsmtFinType1)] <- 'None'
all$BsmtFinType2[is.na(all$BsmtFinType2)] <- 'None'
all$FireplaceQu[is.na(all$FireplaceQu)] <- 'None'
all$GarageType[is.na(all$GarageType)] <- 'No Garage'
all$GarageFinish[is.na(all$GarageFinish)] <- 'None'
all$GarageQual[is.na(all$GarageQual)] <- 'None'
all$GarageCond[is.na(all$GarageCond)] <- 'None'
all$PoolQC[is.na(all$PoolQC)] <- 'None'
all$Fence[is.na(all$Fence)] <- 'None'
all$MiscFeature[is.na(all$MiscFeature)] <- 'None'
all$BsmtFullBath[is.na(all$BsmtFullBath)] <-0
all$BsmtHalfBath[is.na(all$BsmtHalfBath)] <-0
all$BsmtFinSF1[is.na(all$BsmtFinSF1)] <-0
all$BsmtFinSF2[is.na(all$BsmtFinSF2)] <-0
all$BsmtUnfSF[is.na(all$BsmtUnfSF)] <-0
all$TotalBsmtSF[is.na(all$TotalBsmtSF)] <-0
all$MasVnrArea[is.na(all$MasVnrArea)] <-0
```

Luego de realizar todos estos cambios la cantidad de NA disminuye considerablemente como se puede observar en la siguiente imagen:

```
LotFrontage GarageYrBlt  MasVnrType    MSZoning   Utilities
        486         159          24           4           2
 Functional Exterior1st Exterior2nd  Electrical KitchenQual
          2           1           1           1           1
 GarageCars  GarageArea    SaleType
          1           1           1
> cat('Hay', length(NAcol), 'columnas con valores NA')
Hay 13 columnas con valores NA>
```

Para el resto de los valores NA en la base de datos que sí significan ausencia de datos se procedió a eliminar todos los registros que tuvieran datos faltantes con la función $na.omit()$ de R. Finalmente la base de datos quedó libre de valores NA como se puede observar en la siguiente imagen:

```
> cat('Hay', length(NAcol), 'columnas con valores NA')
Hay 0 columnas con valores NA>
```

Con este último procedimiento la base de datos pasó de tener 2919 registros a tener 2261 registros:

Antes

```
> dim(all)
[1] 2919   81
```

Después

```
> dim(all)
[1] 2261   81
```

Tratando las variables tipo factor, ordinales y numéricas:

En un inicio todas las variables eran numéricas o de tipo carácter, pero muchas de ellas podían ser transformadas a tipo factor en caso de que una de sus categorías no tuviera más importancia que otra, o tipo ordinal en caso de que sí. Todas las variables de la imagen siguiente fueron convertidas a tipo factor por ser variables categóricas cuyo orden no importa:

```
###Convirtiendo a factor las variables categóricas.
all$MSZoning<- as.factor(all$MSZoning)
all$Alley<- as.factor(all$Alley)
all$LandContour<- as.factor(all$LandContour)
all$Utilities<- as.factor(all$Utilities)
all$LotConfig<- as.factor(all$LotConfig)
all$Neighborhood<- as.factor(all$Neighborhood)
all$Condition1<- as.factor(all$Condition1)
all$Condition2<- as.factor(all$Condition2)
all$BldgType<- as.factor(all$BldgType)
all$HouseStyle<- as.factor(all$HouseStyle)
all$RoofStyle<- as.factor(all$RoofStyle)
all$RoofMatl<- as.factor(all$RoofMatl)
all$Exterior1st<- as.factor(all$Exterior1st)
all$Exterior2nd<- as.factor(all$Exterior2nd)
all$Foundation<- as.factor(all$Foundation)
all$Heating<- as.factor(all$Heating)
all$Electrical<- as.factor(all$Electrical)
all$GarageType<- as.factor(all$GarageType)
all$Fence<- as.factor(all$Fence)
all$MiscFeature<- as.factor(all$MiscFeature)
all$SaleType<- as.factor(all$SaleType)
all$SaleCondition<- as.factor(all$SaleCondition)
```

Hay dos casos en los que las variables eran numéricas originalmente, pero en realidad deberían ser categóricas. Este es el caso de MoSold y MSSubClass las cuales fueron convertidas a factor también:

```
all$MoSold <- as.factor(all$MoSold)
all$MSSubClass <- as.factor(all$MSSubClass)
```

Para hacer más fácil el trabajo de convertir las variables categóricas a ordinales donde sí importa el orden se crearon varias categorías que podían ser reutilizadas en varios casos. Para los casos donde las categorías eran particulares de una sola variable pues no se crearon categorías, simplemente se hacía directo sobre la variable correspondiente. La imagen siguiente muestra las categorías creadas:

```
Qualities <- c('None' = 0, 'Po' = 1, 'Fa' = 2, 'TA' = 3, 'Gd' = 4, 'Ex' = 5)
Finish <- c('None'=0, 'Unf'=1, 'RFn'=2, 'Fin'=3)
Exposure <- c('None'=0, 'No'=1, 'Mn'=2, 'Av'=3, 'Gd'=4)
FinType <- c('None'=0, 'Unf'=1, 'LwQ'=2, 'Rec'=3, 'BLQ'=4, 'ALQ'=5, 'GLQ'=6)
Masonry <- c('None'=0, 'BrkCmn'=0, 'BrkFace'=1, 'Stone'=2)
```

Y en la siguiente imagen se muestran todas las variables que fueron convertidas a ordinales:

```
all$PoolQC<-as.integer(revalue(all$PoolQC, Qualities))
all$FireplaceQu<-as.integer(revalue(all$FireplaceQu, Qualities))
all$LotShape<-as.integer(revalue(all$LotShape, c('IR3'=0, 'IR2'=1, 'IR1'=2, 'Reg'=3)))
all$GarageFinish<-as.integer(revalue(all$GarageFinish, Finish))
all$GarageQual<-as.integer(revalue(all$GarageQual, Qualities))
all$GarageCond<-as.integer(revalue(all$GarageCond, Qualities))
all$BsmtQual<-as.integer(revalue(all$BsmtQual, Qualities))
all$BsmtCond<-as.integer(revalue(all$BsmtCond, Qualities))
all$BsmtExposure<-as.integer(revalue(all$BsmtExposure, Exposure))
all$BsmtFinType1<-as.integer(revalue(all$BsmtFinType1, FinType))
all$BsmtFinType2<-as.integer(revalue(all$BsmtFinType2, FinType))
all$MasVnrType<-as.integer(revalue(all$MasVnrType, Masonry))
all$KitchenQual<-as.integer(revalue(all$KitchenQual, Qualities))
all$Functional <- as.integer(revalue(all$Functional, c('Sal'=0, 'Sev'=1, 'Maj2'=2,
                                     'Maj1'=3, 'Mod'=4, 'Min2'=5, 'Min1'=6, 'Typ'=7)))
all$ExterQual<-as.integer(revalue(all$ExterQual, Qualities))
all$ExterCond<-as.integer(revalue(all$ExterCond, Qualities))
all$HeatingQC<-as.integer(revalue(all$HeatingQC, Qualities))
all$CentralAir<-as.integer(revalue(all$CentralAir, c('N'=0, 'Y'=1)))
all$LandSlope<-as.integer(revalue(all$LandSlope, c('Sev'=0, 'Mod'=1, 'Gtl'=2)))
all$Street<-as.integer(revalue(all$Street, c('Grvl'=0, 'Pave'=1)))
all$PavedDrive<-as.integer(revalue(all$PavedDrive, c('N'=0, 'P'=1, 'Y'=2)))
```

Agrupando variables en variables nuevas más generales:

Hay algunas variables que pueden ser agrupadas para conformar nuevas variables que de cierto modo contienen la información de las agrupadas un ejemplo es las variables que indican la cantidad de baños que posee la vivienda. En lugar de tener varias variables que indican cuantos baños hay de cada tipo se pueden unir para conformar una variable que indique el total de los baños que tiene la vivienda. A continuación, se muestran todas las variables que fueron agrupadas con una pequeña explicación de lo que significa cada variable:

```
###TotBathrooms: contiene la suma de los baños de la vivienda.
all$TotBathrooms <- all$FullBath + (all$HalfBath*0.5) + all$BsmtFullBath + (all$BsmtHalfBath*0.5)

###Remod: contiene información sobre si una vivienda fue remodelda o no.
all$Remod <- ifelse(all$YearBuilt==all$YearRemodAdd, 0, 1) #0=No Remodelada, 1=Remodelada

###Age: contiene información sobre los años que pasaron desde que se remodeló la vivienda
###hasta que fue vendida.
all$Age <- as.numeric(all$YrSold)-all$YearRemodAdd

###IsNew: contiene información sobre si la vivienda se vendió en el mismo año que se construyó.
all$IsNew <- ifelse(all$YrSold==all$YearBuilt, 1, 0)

###TotalSqFeet: contiene la suma de los pies cuadrados de la vivienda.
all$TotalSqFeet <- all$GrLivArea + all$TotalBsmtSF

###TotalSqFeet: contiene la suma de todos los pies cuadrados del portal.
all$TotalPorchSF <- all$OpenPorchSF + all$EnclosedPorch + all$X3SsnPorch + all$ScreenPorch

#NeighRich: agrupa los diferentes vecindarios en tan solo tres niveles (pobre=0, medio=1, rico=2)
all$NeighRich[all$Neighborhood %in% c('StoneBr', 'NridgHt', 'NoRidge')] <- 2
all$NeighRich[!all$Neighborhood %in% c('MeadowV', 'IDOTRR', 'BrDale',
                                       'StoneBr', 'NridgHt', 'NoRidge')] <- 1
all$NeighRich[all$Neighborhood %in% c('MeadowV', 'IDOTRR', 'BrDale')] <- 0
```

Luego de agrupar las variables se pueden eliminar algunas de las variables utilizadas para crear las nuevas y de esta forma se evita tener información repetida en la base de datos. La siguiente imagen muestra aquellas variables que fueron eliminadas:

```
all$FullBath <- NULL
all$HalfBath <- NULL
all$BsmtFullBath <- NULL
all$BsmtHalfBath <- NULL
all$GrLivArea <- NULL
all$TotalBsmtSF<- NULL
all$OpenPorchSF <- NULL
all$EnclosedPorch<- NULL
all$X3SsnPorch <- NULL
all$ScreenPorch <- NULL
all$Neighborhood<- NULL
```

Hay dos variables en la base de datos que también pueden ser eliminadas ya que no aportan información relevante para la creación de los modelos, estas variables son “Id” y “Utilities”. La variable “Id” es tan solo el identificador de las variables que no aporta información sobre si el precio de una vivienda puede subir o bajar. La variable “Utilities” tiene el mismo valor en todas sus filas por lo que tampoco aporta información útil y puede ser eliminada. Por lo tanto:

```
all$Id<- NULL
all$Utilities<- NULL
```

Por último, para terminar de refactorizar las variables, la variable YrSold que en un inicio es numérica puede ser convertida a factor porque ya se tiene su información contenida en parte dentro de las variables IsNew y Age que se crearon anteriormente. Así que el valor numérico de YrSold ya no es necesario:

```
all$YrSold <- as.factor(all$YrSold)
```

Listo todas las variables ya se encuentran en el formato correcto y se puede pasar a verificar si es necesario o no normalizar los datos.

1c. Determinar si es necesario hacer una normalización de la base de datos.

En la base de datos hay valores numéricos expresados en cantidades pequeñas como la cantidad de baños que tiene la vivienda, que pueden ser 1 o 2 o valores así pequeños, y hay otros valores como la cantidad de pies cuadrados que pueden estar expresados en cientos o incluso miles. Lo correcto en este caso sería llevar a cabo una normalización de los datos para que los resultados fueran más precisos. Para llevar a cabo la normalización es necesario separa la base de datos en valores numéricos y valores tipo factor lo que se hizo de la siguiente manera:

```
> numericVars <- which(sapply(all, is.numeric))
> factorVars <- which(sapply(all, is.factor))
> cat('Hay', length(numericVars), 'variables numéricas, y',
Hay 52 variables numéricas, y 23 variables categóricas>
```

Antes de aplicar la normalización se eliminan las variables que presentan mucha correlación entre ellas. Para lograrlo se encontraron parejas de variables que tuvieran una correlación mayor a 0.7 y se eliminó la que menos correlación tuviera con la variable de resultado (SalePrice). El método $corSelect()$ se encarga de realizar esta selección:

```
excludedVars <- corSelect(all, sp.cols = 68 ,numericVarNames [c(-45)] , cor.thresh = 0.7)
```

Las variables eliminadas con este método fueron las siguientes:

```
[1] "ExterQual"   "BsmtFinSF2"  "FireplaceQu" "GarageYrBlt" "GarageArea"
[6] "PoolQC"      "Age"         "TotalSqFeet"
```

Como un paso adicional las variables categóricas se separaron en variables “dummies” para que fuera más sencillo para los métodos de construcción de modelos analizar estas variables. El procedimiento se llevó a cabo de la siguiente manera:

```
###Creando variables dummies para todas las variables de tipo factor.
DFdummies <- as.data.frame(model.matrix(~., factorVarsData))

###Eliminando las variables dummies que tengan menos de 10 valores iguales a 1 en sus columnas
###porque se considera que tiene muy pocas apariciones en la base de datos y por tanto no aportan
###muchas información.
fewOnes <- which(colSums(DFdummies[1:nrow(DFdummies),])<10)
DFdummies <- DFdummies[,-fewOnes]
```

Como se puede ver en la imagen también se eliminaron de estas variables “dummies” aquellas que tenían menos de 10 apariciones, o sea las que el total de unos en sus columnas es menor que 10, debido a que al tener tan pocas apariciones se considera que no aportan información realmente relevante para la construcción del modelo.

Finalmente se lleva a cabo la normalización de las variables numéricas, para ellos se aplica el logaritmo más 1 de todas aquellas variables cuyo valor absoluto del coeficiente de asimetría sea mayor a 0.8, luego se aplica una técnica de centrado y escalado de los valores para que finalmente queden normalizados de la siguiente manera:

```
for(i in 1:ncol(allNumeric)){
        if (abs(skew(allNumeric[,i]))>0.8){
                allNumeric[,i] <- log(allNumeric[,i] +1)
        }
}

###Centrando y escalando todos los valores.
PreNum <- preProcess(allNumeric, method=c("center", "scale"))

###Normalizando los datos.
DFnorm <- predict(PreNum, allNumeric)
```

En la siguiente imagen se puede observar un ejemplo de cómo estaban las variables numéricas antes y cómo quedan luego de ser normalizadas:

Antes:

```
> head(selectedNumericVars)
  LotFrontage LotArea Street LotShape LandSlope OverallQual OverallCond
1          65    8450      1        3         2           7           5
2          80    9600      1        3         2           6           8
3          68   11250      1        2         2           7           5
4          60    9550      1        2         2           7           5
5          84   14260      1        2         2           8           5
6          85   14115      1        2         2           5           5
```

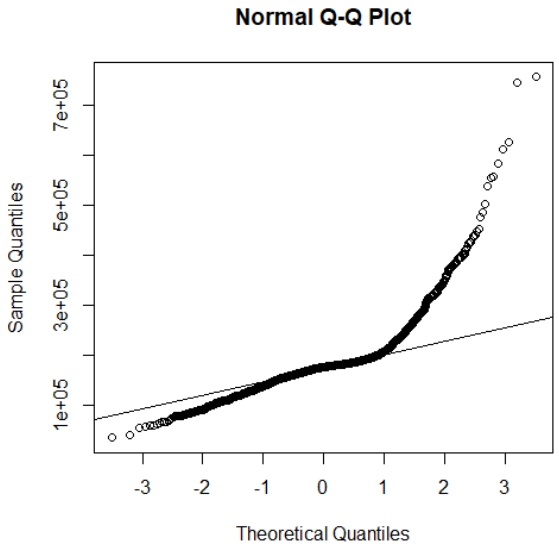
Después:

```
> head(DFnorm)
  LotFrontage     LotArea    Street   LotShape LandSlope OverallQual
1 -0.04244176 -0.07324924 0.0557155  0.5637598 0.2024045   0.5788049
2  0.54866202  0.18757789 0.0557155  0.5637598 0.2024045  -0.1301214
3  0.08586059  0.51179567 0.0557155 -1.0611895 0.2024045   0.5788049
4 -0.26982918  0.17690336 0.0557155 -1.0611895 0.2024045   0.5788049
5  0.68778908  0.99646006 0.0557155 -1.0611895 0.2024045   1.2877312
6  0.72154768  0.97556724 0.0557155 -1.0611895 0.2024045  -0.8390477
```

Por último, se analiza la variable de respuesta (SalePrice) utilizando nuevamente el valor del coeficiente de asimetría y luego el logaritmo:

```
###Viendo el coeficiente de asimetría de las variable de respuesta.
SalePrice <- all$SalePrice
skew(SalePrice)
qqnorm(SalePrice)
qqline(SalePrice)
```

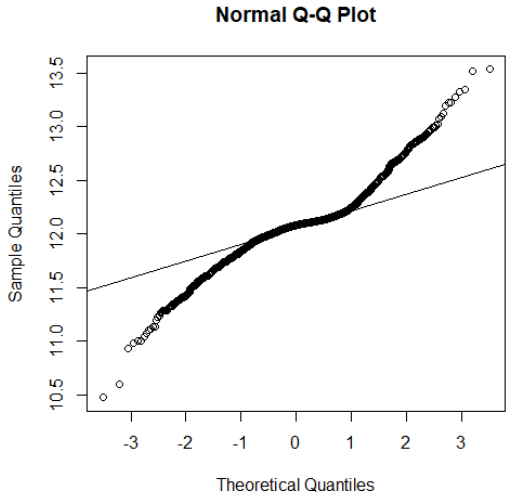
```
> skew(SalePrice)
[1] 2.762849
```



El valor del coeficiente de asimetría está un poco alto y se puedo observar que una asimetría hacia la derecha en el grafico por lo que se aplica el logaritmo a todos los valores de SalePrice:

```
SalePrice  <- log(SalePrice)
skew(SalePrice)
qqnorm(SalePrice)
qqline(SalePrice)
```

```
> skew(SalePrice)
[1] 0.2394391
```



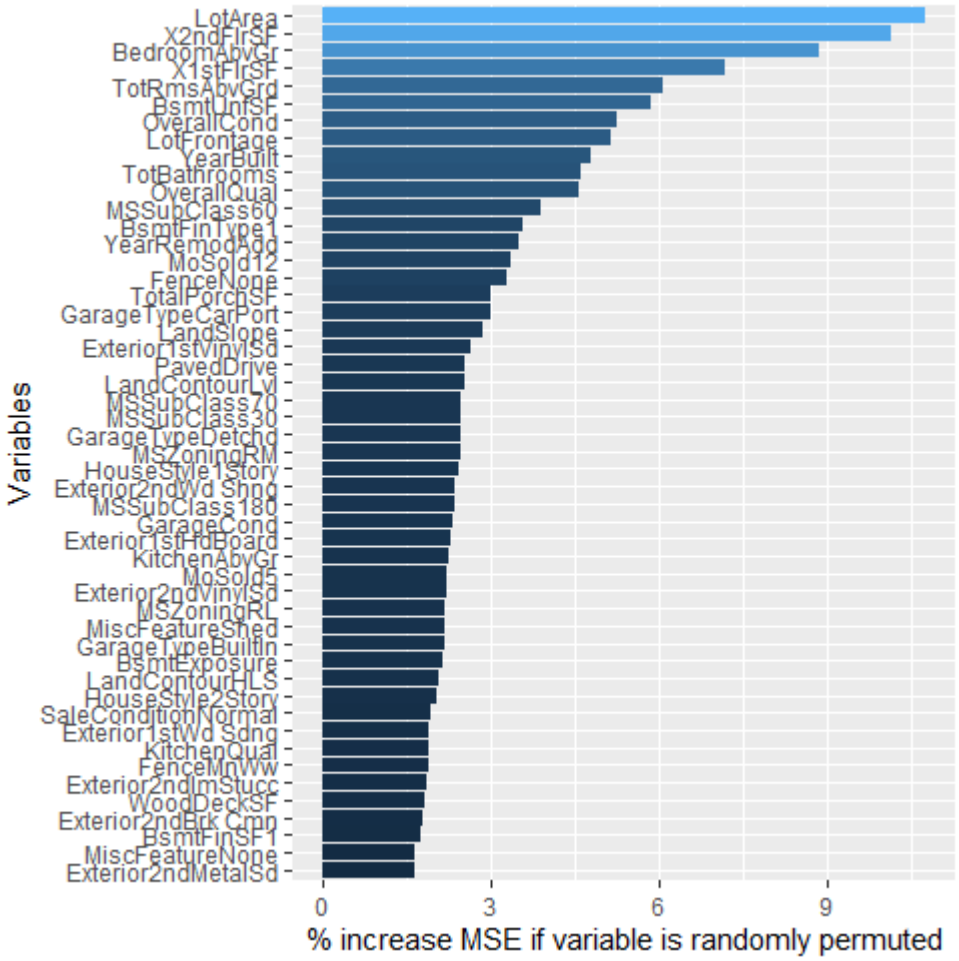
Ya se puede observar que la gráfica queda mejor balanceada. Con este último procedimiento ya quedan normalizados todos los valores numéricos de la base de datos y ya se puede pasar a seleccionar las variables más importantes para utilizarlas en la

construcción de los modelos. Pero primero su unos nuevamente las variables numéricas ya normalizadas con las variables “dummies” que se crearon anteriormente:

```
combined <- cbind(DFnorm, DFdummies)
all <- cbind(combined, SalePrice)
```

Para seleccionar las variables que mayor importancia tienen en la base de datos, o sea las que mayor impacto tienen sobre el comportamiento de la variable de respuesta (SalePrice) se realizó un $randomForest()$ y luego se graficó la importancia de las variables ordenadas de mayor a menor de la siguiente manera:

```
quickRF <- randomForest(x=all[,-155], y=all$SalePrice, ntree=100,importance=TRUE)
impRF <- importance(quickRF)
impRF <- data.frame(Variables = row.names(impRF), MSE = impRF [,1])
impRF <- impRF [order(impRF$MSE, decreasing = TRUE),]

ggplot(impRF[1:50,], aes(x=reorder(Variables, MSE), y=MSE, fill=MSE))
            + geom_bar(stat = 'identity') + labs(x = 'Variables',
            y= '% increase MSE if variable is randomly permuted')
            + coord_flip() + theme(legend.position="none")
```



Analizando el gráfico de la importancia de las variables se puede concluir que con las primeras 30 variables ya se explica la mayor parte del comportamiento de la variable de respuesta por lo que se seleccionan estas variables para continuar con todo el proceso.

2. Enfoque train-test validation con regsubsets y MSE

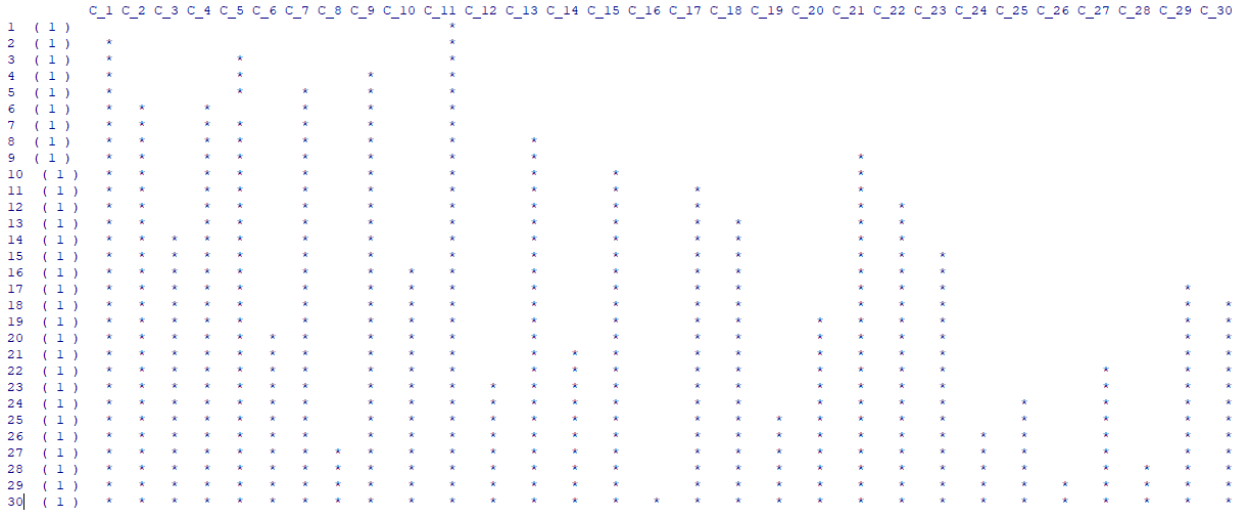
La base de datos se dividió en entrenamiento (80%) y prueba (20%) de manera aleatoria con el método $createDataPartition()$:

```
index <- createDataPartition(all$SalePrice, times = 1, p = 0.8, list = FALSE)
train <- all[index ,]
test <- all[-index,]
```

Luego se utilizó el método $regsubsets()$ para probar exhaustivamente todas las posibles combinaciones diferentes de variables desde 1 hasta 30 para poder seleccionar el modelo de menor MSE. La siguiente imagen muestra cómo se empleó este método:

```
set.seed(1)
regfitBest <- regsubsets(train$SalePrice ~ ., data = train, nvmax = 30)
```

Para poder ver cuáles fueron los mejores modelos con cada cantidad de variables se muestra la tabla $outmat$ que devuelve el método $regsubsets()$ y se modificaron los nombres de las columnas de esta tabla para que se pudiera visualizar de manera correcta:



Como se puede observar en la imagen hay variables que están presentes en casi todos los modelos por lo que se podría decir que tienen una importancia alta para el modelo, algunas de estas variables son:

C_1: LotArea.

C_2: X2ndFlrSF.

C_4: X1stFlrSF.

C_5: TotRmsAbvGrd.

C_7: OverallCond.

C_9: YearBuilt.

C_11: OverallQual.

Solo por mencionar algunas.

Por el otro lado tenemos algunas variables que prácticamente no aparecen en casi ningún modelo solo en aquellos en los que se prueba con un alto número de variables. Por lo que se puede decir que estas variables no tienen una importancia muy alta para el modelo. Algunos ejemplos de estas variables son:

C_8: LotFrontage

C_16: FenceNone

C_24: MSSubClass30

C_26: MSZoningRM

C_28: Exterior2ndWd Shng

Luego para determinar cuál es el mejor modelo se utiliza el criterio del error cuadrático medio o MSE por sus siglas en inglés. Se calcula el MSE para cada uno de los modelos y se escoge el que menos error tenga:

```
### Calculando el MSE para cada uno modelos.
testMatrix <- model.matrix(test$SalePrice~ ., data = test)
val.errors <- rep(NA, 30)
for (i in 1:30) {
 coefi <- coef(regfitBest, id = i)
 pred <- testMatrix[, names(coefi)] %*% coefi
 val.errors[i] <- mean((exp(test$SalePrice) - exp(pred))^2)
}

### Seleccionando el modelo con el menor MSE.
val.errors
which.min(val.errors)
```

Al realizar estos procedimientos obtenemos un valor de MSE para cada uno de los modelos y el menor de todos es el que ocupa la posición 16, por lo que seleccionamos el modelo que tiene 16 variables como el mejor. Es necesario aclarar que se aplicó el exponencial de las predicciones y de la variable SalePrice del conjunto de prueba para poder apreciar los MSEs sin el efecto de la normalización que se hizo aplicando el logaritmo en la etapa de preparación de los datos:

```
> val.errors
 [1] 1966907917 1791438705 1738513351 1705586815 1712590326 1750838013
 [7] 1727220713 1719822229 1721195511 1714544045 1710516358 1687863546
[13] 1687588145 1670439295 1664667763 1659845079 1660764971 1660315216
[19] 1669275840 1667212856 1666999076 1663275220 1665611661 1667334057
[25] 1669957505 1670185327 1669305622 1669382072 1669866938 1669826598
> which.min(val.errors)
[1] 16
```

Luego utilizamos los coeficientes de este modelo para construir la ecuación del modelo:

```
> coef(regfitBest, which.min(val.errors))
      (Intercept)           LotArea         X2ndFlrSF
     12.040828732       0.053372614       0.051275255
     BedroomAbvGr          X1stFlrSF      TotRmsAbvGrd
      0.010224312       0.063505027       0.031232941
      OverallCond         YearBuilt      TotBathrooms
      0.042778400       0.075184352       0.009012964
      OverallQual       BsmtFinType1          MoSold12
      0.056321845       0.019169139       0.055953279
     TotalPorchSF GarageTypeCarPort         PavedDrive
      0.010191349      -0.103625094      -0.013618079
    LandContourLvl       MSSubClass70
      0.026557261       0.033617768
```

$$Ecuación = 12.04 + (0.05 * LotArea) + (0.05 * X2ndFlrSF) + (0.01 * BedroomAbvGr) + (0.06 * X1stFlrSF) + (0.03 * TotRmsAbvGrd) + (0.04 * OverallCond) + (0.075 * YearBuilt) + (0.009 * TotBathrooms) + (0.056 * OverallQual) + (0.019 * BsmtFinType1) + (0.056 * MoSold12) + (0.010 * TotalPorchSF) + (-0.134 * GarageTypeCarPort) + (-0.013 * PavedDrive) + (0.027 * LandContourLvl) + (0.034 * MSSubClass70)$$

3. Enfoque CrossValidation con regsubsets y MSE

Para realizar la validación cruzada se estableció un número de folds igual a 10 y se utilizó el método $regsubsets()$ para buscar de manera exhaustiva el mejor modelo comprobando con todas las combinaciones posibles de variables desde 1 hasta 30. La diferencia es que ahora para cada modelo se hace también una validación cruzada por lo que se tendrían 10 valores de MSE para cada modelo y luego se calcula la media para tener resultados más

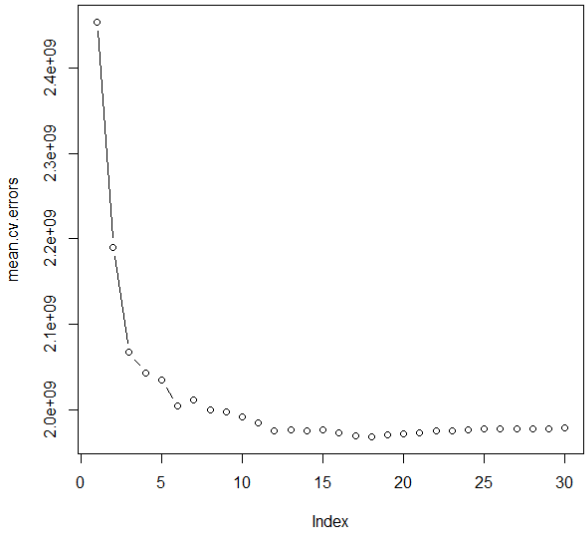
precisos. En la siguiente imagen se muestra el método utilizado para encontrar los modelos y calcular los errores:

```
set.seed(1)
folds <- sample(rep(1:k, length = n))
cv.errors <- matrix(NA, k, 30, dimnames = list(NULL, paste(1:30)))
for (j in 1:k) {
  best.fit <- regsubsets(SalePrice ~ .,
       data = all[folds != j,],
       nvmax = 30)
  for (i in 1:30) {
    pred <- predict(best.fit, all[folds == j, ], id = i)
    cv.errors[j, i] <- mean((exp(all$SalePrice[folds == j]) - exp(pred))^2)

   }
}
```

Luego se calcula la media de los errores y se muestra cual de todos los modelos tiene el error más bajo:

```
###Calculando la media del MSE obtenido en los folds de cada modelo.
mean.cv.errors <- apply(cv.errors, 2, mean)

###Escogiendo el mejor modelo según el MSE
which.min(mean.cv.errors)
par(mfrow = c(1, 1))
plot(mean.cv.errors, type = "b")
```

```
> which.min(mean.cv.errors)
18
```



En las imágenes anteriores se puede observar que el modelo que tiene el menor MSE es el que tiene 18 variables por lo que se selecciona este modelo y se utilizan sus coeficientes para construir la ecuación del modelo:

```
> coef(reg.best, which.min(mean.cv.errors))
      (Intercept)           LotArea         X2ndFlrSF
      12.02049217        0.05235413        0.05798644
     BedroomAbvGr         X1stFlrSF      TotRmsAbvGrd
       0.01633539        0.05874467        0.02493525
       BsmtUnfSF       OverallCond         YearBuilt
       0.00503490        0.03978672        0.07277189
     TotBathrooms       OverallQual       BsmtFinType1
       0.01360730        0.05527337        0.01838004
         MoSold12      TotalPorchSF GarageTypeCarPort
       0.06039910        0.01124562       -0.08540684
       PavedDrive    LandContourLvl      MSSubClass70
      -0.01366516        0.03220218        0.04183523
 HouseStyle1Story
       0.02848037
```

$$Ecuación = 12.02 + (0.052 * LotArea) + (0.058 * X2ndFlrSF) + (0.016 * BedroomAbvGr) + (0.059 * X1stFlrSF) + (0.025 * TotRmsAbvGrd) + (0.005 * BsmtUnfSF) + (0.040 * OverallCond) + (0.073 * YearBuilt) + (0.014 * TotBathrooms) + (0.055 * OverallQual) + (0.018 * BsmtFinType1) + (0.060 * MoSold12) + (0.011 * TotalPorchSF) + (-0.085 * GarageTypeCarPort) + (-0.014 * PavedDrive) + (0.032 * LandContourLvl) + (0.042 * MSSubClass70) + (0.028 * HouseStyle1Story)$$

4. Enfoque CrossValidation con regsubsets y R cuadrada ajustada (Puntos Extras)

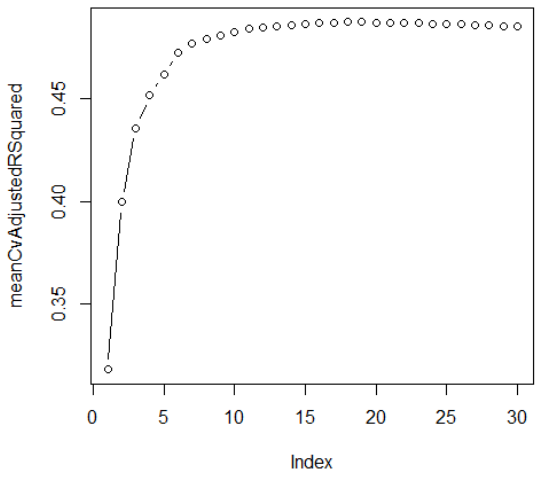
Para realizar la validación cruzada se estableció un número de folds igual a 10 y se utilizó el método $regsubsets()$ para buscar de manera exhaustiva el mejor modelo comprobando con todas las combinaciones posibles de variables desde 1 hasta 30. La diferencia es que ahora para cada modelo se hace también una validación cruzada por lo que se tendrían 10 valores de R cuadrada ajustada para cada modelo y luego se calcula la media para tener resultados más precisos. En la siguiente imagen se muestra el método utilizado para encontrar los modelos y almacenar las R cuadras ajustadas de cada uno:

```
set.seed(1)
folds <- sample(rep(1:k, length = n))
cvAdjustedRSquared <- matrix(NA, k, 30, dimnames = list(NULL, paste(1:30)))
for (j in 1:k) {
  best.fit <- regsubsets(SalePrice ~ .,
       data = all[folds != j,],
       nvmax = 30)
  summaryBestFit <- summary(best.fit)
  cvAdjustedRSquared[j,] <- summaryModel$adjr2
}
```

Luego se calcula la media de las R cuadradas ajustadas y se muestra cuál de todos los modelos tiene el mayor valor de R cuadrada ajustada:

```
###Calculando la media de la R cuadrada ajustada obtenida en
###los folds de cada modelo.
meanCvAdjustedRSquared  <- apply(cvAdjustedRSquared, 2, mean)

###Escogiendo el mejor modelo según la R cuadrada ajustada
which.max(meanCvAdjustedRSquared )
par(mfrow = c(1, 1))
plot(meanCvAdjustedRSquared , type = "b")
```

```
> which.max(meanCvAdjustedRSquared )
18
```



En las imágenes anteriores se puede observar que el modelo que tiene la mayor R cuadrada ajustada es el que tiene 18 variables por lo que se selecciona este modelo y se utilizan sus coeficientes para construir la ecuación del modelo:

```
> coef(reg.best, which.max(meanCvAdjustedRSquared))
      (Intercept)           LotArea         X2ndFlrSF
      12.02049217        0.05235413        0.05798644
     BedroomAbvGr         X1stFlrSF      TotRmsAbvGrd
       0.01633539        0.05874467        0.02493525
       BsmtUnfSF       OverallCond         YearBuilt
       0.00503490        0.03978672        0.07277189
     TotBathrooms       OverallQual      BsmtFinType1
       0.01360730        0.05527337        0.01838004
         MoSold12      TotalPorchSF GarageTypeCarPort
       0.06039910        0.01124562       -0.08540684
       PavedDrive    LandContourLvl      MSSubClass70
      -0.01366516        0.03220218        0.04183523
HouseStyle1Story
       0.02848037
```

$$Ecuación = 12.02 + (0.052 * LotArea) + (0.058 * X2ndFlrSF) + (0.016 * BedroomAbvGr) + (0.059 * X1stFlrSF) + (0.025 * TotRmsAbvGrd) + (0.005 * BsmtUnfSF) + (0.040 * OverallCond) + (0.073 * YearBuilt) + (0.014 * TotBathrooms) + (0.055 * OverallQual) + (0.018 * BsmtFinType1) + (0.060 * MoSold12) + (0.011 * TotalPorchSF) + (-0.085 * GarageTypeCarPort) + (-0.014 * PavedDrive) + (0.032 * LandContourLvl) + (0.042 * MSSubClass70) + (0.028 * HouseStyle1Story)$$

Para este experimento se obtiene que el mejor modelo es el que tiene 18 variables tanto para el MSE como se observó en la sección anterior (Sección 3) como para la R cuadrada ajustada como se observó en esta sección. Al haber realizado el mismo procedimiento, pero teniendo en cuenta métricas diferentes y obtener el mismo resultado se puede estar aún más seguros de que los resultados son correctos.

5. Enfoque CrossValidation con Ridge Regression y Lasso

La base de datos se dividió en entrenamiento (80%) y prueba (20%) de manera aleatoria con el método $createDataPartition()$:

```
index <- createDataPartition(all$SalePrice, times = 1, p = 0.8, list = FALSE)
train <- all[index ,]
test <- all[-index,]
```

En este caso para mayor comodidad los conjuntos de entrenamiento y prueba se separaron para tener la variable de respuesta en variables aparte de las predictoras:

```
xTrain <- model.matrix(SalePrice ~ ., train)[, -1]
yTrain <- train$SalePrice
xTest <- model.matrix(SalePrice ~ ., test)[, -1]
yTest <- test$SalePrice
```

5.1 Ridge Regression

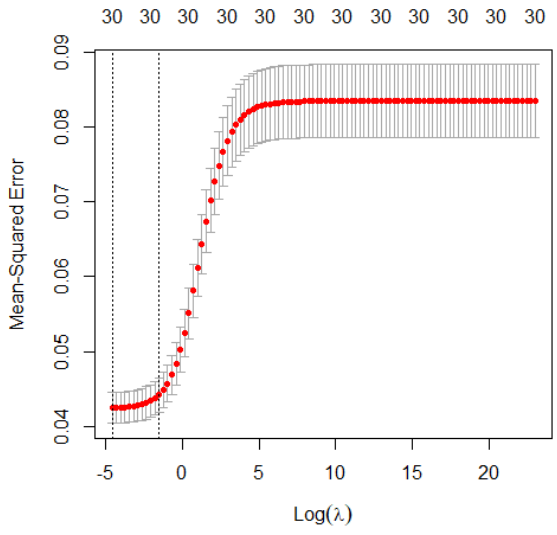
Para encontrar un modelo con el método de Ridge primero es necesario crear un vector con muchos valores diferentes de lambda para poder probarlos uno a uno hasta encontrar el valor óptimo. Es necesario aclarar que el valor de lambda está directamente relacionado con la penalización que se les aplica a los coeficientes del modelo, una mayor lambda resulta en una mayor penalización. Por este motivo, el proceso de selección del valor óptimo de lambda es muy importante. Para crear el vector que contiene diferentes valores de lambda se utiliza el siguiente método:

```
grid <- 10^seq(10, -2, length = 100)
```

Luego se procede a aplicar el método de Ridge con validación cruzada y se plantea un valor de umbral ("thresh") para probar con todos los valores de lambda hasta que el cambio en los coeficientes sea menor al valor especificado en thresh, en este caso se utilizó un valor de $1e-12$. Para seleccionar el número de folds no existe un valor óptimo que funcione siempre mejor para todas las situaciones, va a depender mucho del tamaño de los datos y de la capacidad de cómputo. En este trabajo de probó con nfolds igual a 5, 10, 15 y 20, en los cuatro casos los coeficientes del modelo resultante fueron exactamente iguales por lo que se decidió dejar el valor por defecto que es 10. La imagen siguiente muestra el método que se utilizó para aplicar la regresión Ridge con validación cruzada:

```
set.seed(1)
ridgeMod <- cv.glmnet(xTrain, yTrain, alpha = 0, lambda = grid, thresh = 1e-12, nfolds = 10)
```

A continuación, se muestra un gráfico donde se puede apreciar el comportamiento del error cuadrático medio a medida que cambia el valor de lambda:



Luego se selecciona el valor óptimo de lambda y se vuelve a aplicar el método de Ridge, pero esta vez solo utilizando el valor óptimo de lambda para encontrar el mejor modelo:

```
###Encontrando el valor óptimo de lambda.
bestLambdaRidge <- ridgeMod$lambda.min

###Encontrando el modelo de Ridge utilizando el valor óptimo de lambda.
finalModelRidge <- glmnet(xTrain, yTrain, alpha = 0, lambda = bestLambdaRidge)
```

Después se calcula el MSE del modelo encontrado:

```
###Calculando el valor de MSE para el modelo de Ridge.
ridgePred <- predict(finalModelRidge , s = bestLambdaRidge, newx = xTest)
mean((ridgePred - yTest)^2)
```

```
> mean((exp(ridgePred) - exp(yTest))^2)
[1] 1860434340
```

Hay que aclarar que igualmente en este caso se aplica la función $exp()$ para revertir la normalización que se hizo con el logaritmo y poder apreciar el MSE con su valor original. Luego se utilizan los coeficientes del modelo para construir la ecuación:

(Intercept)	12.040162486
LotArea	0.051322899
X2ndFlrSF	0.044875235
BedroomAbvGr	0.017853873
X1stFlrSF	0.055953151
TotRmsAbvGrd	0.026599870
BsmtUnfSF	0.003287668
OverallCond	0.039434117
LotFrontage	-0.005027864
YearBuilt	0.062420634
TotBathrooms	0.015572166
OverallQual	0.056687664
MSSubClass60	0.009485342
BsmtFinType1	0.016123543
YearRemodAdd	0.007173213
MoSold12	0.054879969
FenceNone	0.002444998
TotalPorchSF	0.011497604
GarageTypeCarPort	-0.024387391
LandSlope	-0.000893870
Exterior1stVinylSd	-0.002382226
PavedDrive	-0.016640197
LandContourLvl	0.023600130
MSSubClass70	0.042002617
MSSubClass30	-0.002554523
GarageTypeDetchd	-0.014815717
MSZoningRM	-0.008412545
HouseStyle1Story	0.009190677
`Exterior2ndWd Shng`	0.016253816
MSSubClass180	-0.018339385
GarageCond	0.006018840

$$Ecuación = 12.04 + (0.051 * LotArea) + (0.045 * X2ndFlrSF) + (0.018 * BedroomAbvGr) + (0.056 * X1stFlrSF) + (0.027 * TotRmsAbvGrd) + (0.003 * BsmtUnfSF) + (0.039 * OverallCond) + (-0.005 * LotFrontage) + (0.062 * YearBuilt) + (0.016 * TotBathrooms) + (0.057 * OverallQual) + (0.009 * MSSubClass60) + (0.016 * BsmtFinType1) + (0.007 * YearRemodAdd) + (0.055 * MoSold12) + (0.002 * FenceNone) + (0.011 * TotalPorchSF) + (-0.024 * GarageTypeCarPort) + (-0.001 * LandSlope) + (-0.002 * Exterior1stVinylSd) + (-0.017 * PavedDrive) + (0.024 * LandContourLv1) + (0.042 * MSSubClass70) + (-0.003 * MSSubClass30) + (-0.015 * GarageTypeDetchd) + (-0.008 * MSZoningRM) + (0.009 * HouseStyle1Story) + (0.016 * Exterior2ndWd Shng) + (-0.018 * MSSubClass180) + (0.006 * GarageCond)$$

5.2 Lasso

Para encontrar un modelo con el método de Ridge primero es necesario crear un vector con muchos valores diferentes de lambda para poder probarlos uno a uno hasta encontrar el valor óptimo. Es necesario aclarar que el valor de lambda está directamente relacionado con la penalización que se les aplica a los coeficientes del modelo, una mayor lambda resulta en una mayor penalización. Por este motivo, el proceso de selección del valor óptimo de

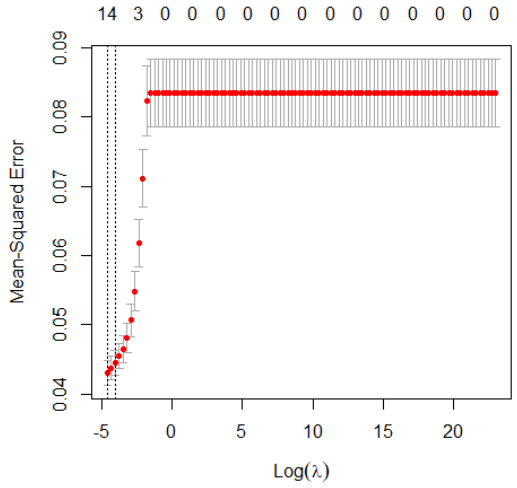
lambda es muy importante. Para crear el vector que contiene diferentes valores de lambda se utiliza el siguiente método:

```
grid <- 10^seq(10, -2, length = 100)
```

Luego se procede a aplicar el método de Lasso con validación cruzada y se plantea un valor de umbral ("thresh") para probar con todos los valores de lambda hasta que el cambio en los coeficientes sea menor al valor especificado en thresh, en este caso se utilizó un valor de $1e-12$. Para seleccionar el número de folds no existe un valor óptimo que funcione siempre mejor para todas las situaciones, va a depender mucho del tamaño de los datos y de la capacidad de cómputo. En este trabajo de probó con nfolds igual a 5, 10, 15 y 20, en los cuatro casos los coeficientes del modelo resultante fueron exactamente iguales por lo que se decidió dejar el valor por defecto que es 10. La imagen siguiente muestra el método que se utilizó para aplicar la regresión Ridge con validación cruzada:

```
set.seed(1)
lassoMod <- cv.glmnet(xTrain, yTrain, alpha = 1, lambda = grid, thresh = 1e-12, nfolds = 10)
```

A continuación, se muestra un gráfico donde se puede apreciar el comportamiento del error cuadrático medio a medida que cambia el valor de lambda:



Luego se selecciona el valor óptimo de lambda y se vuelve a aplicar el método de Lasso, pero esta vez solo utilizando el valor óptimo de lambda para encontrar el mejor modelo:

```
###Encontrando el valor óptimo de lambda.
bestLambdaLasso <- lassoMod$lambda.min

###Encontrando el modelo de Lasso utilizando el valor óptimo de lambda.
finalModel <- glmnet(xTrain, yTrain, alpha = 0, lambda = bestLambdaLasso)
```

Después se calcula el MSE del modelo:

```
###Calculando el valor de MSE para el modelo de Lasso.
lassoPred <- predict(lassoMod, s = bestLambdaLasso, newx = xTest)
mean((lassoPred - yTest)^2)
```

```
> mean((exp(lassoPred) - exp(yTest))^2)
[1] 1852210495
```

Hay que aclarar que igualmente en este caso se aplica la función $exp()$ para revertir la normalización que se hizo con el logaritmo y poder apreciar el MSE con su valor original. Luego se utilizan los coeficientes del modelo para construir la ecuación:

(Intercept)	12.066613703
LotArea	0.045412765
X2ndFlrSF	0.016727277
BedroomAbvGr	0.013560672
X1stFlrSF	0.034245087
TotRmsAbvGrd	0.033557306
BsmtUnfSF	.
OverallCond	0.021698260
LotFrontage	.
YearBuilt	0.034425869
TotBathrooms	0.025610426
OverallQual	0.072219121
MSSubClass60	0.008001758
BsmtFinType1	0.003612213
YearRemodAdd	0.011605981
MoSold12	.
FenceNone	.
TotalPorchSF	0.004975124
GarageTypeCarPort	.
LandSlope	.
Exterior1stVinylSd	.
PavedDrive	.
LandContourLvl	.
MSSubClass70	.
MSSubClass30	.
GarageTypeDetchd	-0.002059804
MSZoningRM	.
HouseStyle1Story	.
`Exterior2ndWd Shng`	.
MSSubClass180	.
GarageCond	.

$$Ecuación = 12.067 + (0.045 * LotArea) + (0.017 * X2ndFlrSF) + (0.014 * BedroomAbvGr) + (0.034 * X1stFlrSF) + (0.034 * TotRmsAbvGrd) + (0.022 * OverallCond) + (0.034 * YearBuilt) + (0.027 * TotBathrooms) + (0.072 * OverallQual) + (0.008 * MSSubClass60) + (0.004 * BsmtFinType1) + (0.012 * YearRemodAdd) + (0.005 * TotalPorchSF) + (-0.002 * GarageTypeDetchd)$$