Máster Big Data y Business Analytics

UNED 2018/2019

Competición – Kaggle

House Prices: Advanced Regression Techniques

con CARET

Juan Carlos Santiago Culebras
DNI: 51418593T
Fecha: 30 de septiembre de 2019

Contenido

Introducción

Objetivo

El objetivo que busco en este trabajo es intentar integrar el máximo del temario realizado a lo largo del curso, aunque centrándome en el uso de técnicas propias de la ciencia de datos y más en concreto del Machine Learning.

Además, he intentado que la implementación sirva como un banco de pruebas para probar distintas técnicas de forma sencilla en las distintas fases, lo que me permitirá avanzar en el conocimiento tanto de las funciones disponibles como del lenguaje.

Selección del conjunto a estudio

Para empezar, quería que el conjunto seleccionado para el TFM fuera de la plataforma Kaggle, ya que para mí es importante introducirme en dicha plataforma.

He intentado buscar un conjunto de datos que me diera el máximo juego posible, después de buscar mucho me he decidido por la competición recomendada de kaggle "House Prices: Advanced Regression Techniques", ya que:

- Contiene una gran cantidad de variables donde se debe aplicar las distintas técnicas de preprocesamiento, ingeniería y selección de características.
- Me permite evaluar el avance mediante una variable objetivo, aplicando distintas técnicas de regresión.
- Los inconvenientes de este conjunto de datos son:
    - El número de observaciones es bastante reducido
    - No permite enriquecer los datos mediante otros dataset

Descripción de la competición

"House Prices: Advanced Regression Techniques": Es una competición de la plataforma Kaggle para principiantes, donde el objetivo es predecir el precio de casas de la ciudad de Ames, Iowa (Estados Unidos). Para ello, se parte de dos conjuntos de datos con 79 variables, en uno de ellos se incluye la variable objetivo SalePrice y en el otro no (conjunto de test), se trate de obtener el precio más aproximado posible para este conjunto.

La evaluación de la competición se basa en obtener el menor RMSE (raíz del error cuadrático medio), entre el logaritmo del valor predicho y el logaritmo del precio de venta observado.

Metodología de trabajo

La metodología implementada en el desarrollo del modelo de predicción se basa en un proceso iterativo en fases para ir explorando distintas alternativas posibles e ir refinando los resultados. Las fases definidas son las siguientes:

- Fase 1 – Análisis y Limpieza inicial
- Fase 2 – Ingeniería de características
- Fase 3 – Selección de predictores con caret
- Fase 4 – Creación de modelos predictivos con caret
- Fase 5 – Selección del modelo definitivo y presentación de resultados

Cada etapa tomará la salida de la etapa anterior y generará una salida para la siguiente etapa, en base a realizar distintas iteraciones se obtendrán distintos modelos entrenados con distintos predictores, que a su vez utilizarán una ingeniería de características distinta.

Lenguaje y estructura del proyecto

Para la realización del proyecto he elegido el lenguaje r, ya que me resultaba bastante más sencillo realizar el análisis inicial del conjunto de datos gracias al entorno RStudio, donde se pueden consultar fácilmente el valor de las variables.

Dada la extensión del trabajo he decidido dividirlo en distintos ficheros rMarkdown, he generado un proyecto de R que contiene, tanto los ficheros de código como los resultados obtenidos. Cada fichero de código se nombrará con F, el número de la fase a la que pertenece y un nombre descriptivo de su contenido:

- Ejemplo "F01 TFM Preprocesamiento - Estudio y Limpieza inicial.Rmd"

La fase 2 se ha subdividido en dos ficheros con el objetivo de realizar implementaciones distintas de la solución.

El proyecto contiene un directorio por cada fase, donde se guardarán los conjuntos de datos y los modelos obtenidos.

Gestión de los dataset - Pruebas de Test

Como ya se ha comentado, se parte de dos conjuntos de datos, test y train, con 80 variables descriptivas y 1 variable objetivo (SalePrice) en el conjunto de train. En el procesamiento de estos ficheros se procederá de la siguiente manera:

- Para el preprocesamiento se juntarán ambos conjuntos en uno solo (dsDataAll), añadiendo una variable indTrain que nos permita saber si el dato pertenece originalmente al conjunto de entrenamiento o test.
- Las transformaciones y/o eliminaciones se realizarán sobre este conjunto.
- Para la selección de predictores y el entrenamiento de modelos, se procederá a separar el fichero modificado de nuevo en test y train, a su vez el conjunto de entrenamiento se dividirá en 2:
    - dsTrain.training (70%): se utilizará en la selección predictores y en él entrenamiento de modelos, además en estos procesos se utilizará validación cruzada y/o Bootstrap, que nos permitirá generalizar los resultados.
    - dsTrain.CV (30%): No se utilizará en la selección o entrenamiento de modelos, solo se utilizará para evaluar el RMSE de Test.

  En los conjuntos de entrenamiento se eliminan las columnas Id y el indicador de entrenamiento.
- Para cada modelo entrenado obtendremos la medida RMSE para el conjunto de entrenamiento y para el de test.

Fase 1 - Análisis y Limpieza inicial del conjunto de datos

El objetivo de esta etapa es realizar un estudio de los datos y realizar una primera limpieza, ha sido una fase muy costosa ya que implica entender el conjunto de datos tanto en su estructura como en su contenido.

Lo primero ha sido entender el dataset, para ello he utilizado la documentación existente [De Cock 2011], a partir de ella se ha elaborado un documento Excel con los siguientes campos:

- Campo: nombre de la variable en el dataset
- Descripción de la variable
- Tipo (Continua, Discreta, Nominal, Ordinal)
- Valores: Posibles valores que pueden tener las variables ordinales y nominales
- Clasificación: indica a que se refiere la variable
    - Segmento: Edificio, Propiedad, Ubicación, Venta u Otras Caracteristicas
    - Subsegmento: Matiza el campo anterior (Sotano, Porche, Techo, superficie…)

Este documento se ha incluido como Anexo 1 y los datos que contiene se cargaran en el proyecto desde dos archivos csv (campos y campos_valor), ya que lo utilizaré para el preprocesamiento del dataset.

Entrada fase 1

Datos originales train.csv y test.csv

Análisis descriptivo

El primer paso para realizar el análisis es juntar los dos conjuntos “train” y “test”, en un nuevo conjunto “dsDataAll”:

- Añadimos SalePrice al conjunto de Test con valor NA
- Marcamos datos de entrenamiento y test

Realizamos un análisis inicial, que costa de varias subetapas:

- Se estudian las variables existen (dimensión, nombres, descripción, tipos)
    - dim(dsDataAll): 2919 Observaciones 82 Variables
    - names(dsDataAll): Corrijo en el Excel y/o en el dataset los nombres que no cuadran con la documentación.
    - str(dsDataAll): Estudio los tipos de datos y los comparo con la documentación.

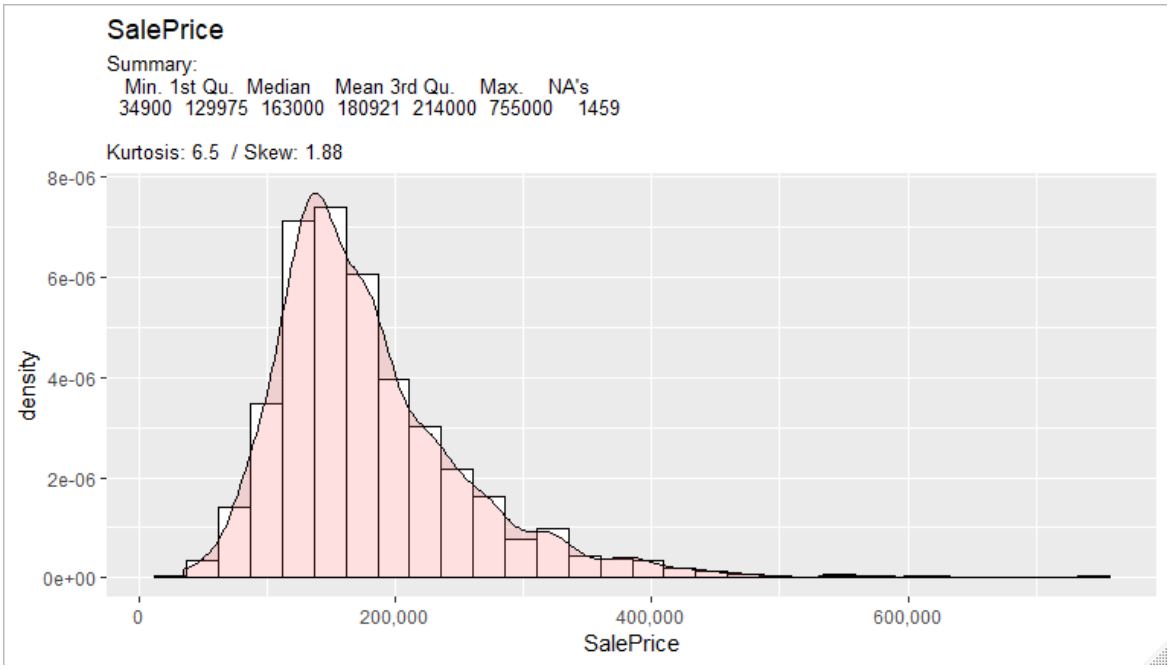
- Se calculan medidas que describen las características más importantes de los datos

- summary(dsDataAll): Examino la distribución de los datos dentro de cada variable.

- Se realizan representación gráfica de las variables.

Se ha creado una función que permiten presentar gráficamente las variables y sus estadísticas de resumen, tendencia central, dispersión y forma.

Examino la variable objetivo:



Para el resto de las variables continuas y discretas se llama a la misma función mediante apply.

Limpieza y preparación de los datos

En general se ha intentado no eliminar observaciones, ya que el conjunto de datos es muy reducido. Se han realizado las siguientes tareas:

Verificación de contenido de datos campos Nominales y Ordinales

En primer lugar, convertimos variables factor a texto, para poder buscar y corregir valores. Seguidamente cruzo los valores existentes en la documentación con los datos de los ficheros:

```
dsCamposValorOriginales <- select(dsDataAll, c("Id",c(dsCamposOrdinalesNominal$Campo))) %>%
  gather("Campo","Valor",c(dsCamposOrdinalesNominal$Campo)) %>%
  na.omit() %>%
  arrange(Id)

# Busco valores que no concuerdan con especificaciones
dsCamposValorOriginales %>%
  anti_join(dsCamposValor,  by = c("Campo","Valor"))
```

En general los datos son correctos, corrijo algunos valores que discrepan de la documentación.

- Normalizar valores para los campos Exterior1st / Exterior2nd -> cambio valor en Excel y corregir valores en el dataSet, elimino espacios en blanco en los valores que pueden dar problemas al convertir variables en dummy
- MSZoning “C (all)” -> Cambio valor a C

Valores faltantes - Missing Data

Muchos algoritmos no aceptan observaciones con valores no definidos (NA), por lo que, es necesario encontrarlos y darles una solución, se puede:

- Eliminar observaciones que estén incompletas: dado que existen pocos datos esta opción no se ha usado.
- Eliminar variables que contengan valores ausentes.
- Estimar los valores ausentes empleando el resto de información disponible (imputación).

Identifico valores faltantes, con el porcentaje que suponen frente al total de observaciones.

```
missingData <- dsDataAll %>%
  summarise_all(funs(sum(is.na(.)))) %>%
  gather("column") %>%
  rename(NumNAs = value) %>%
  mutate(PrcNAs = NumNAs/nrow(dsDataAll)) %>%
  filter(NumNAs!=0) %>%
  arrange(desc(PrcNAs))
```

He eliminado aquellas variables que contienen un número de valores faltantes muy alto, en nuestro caso se eliminan si superan un 80% de los datos. Las variables eliminadas son:

- PoolQC (99,65% de valores faltantes) - Calidad de la piscina
- MiscFeature (96,40% de valores faltantes) - Características varias no cubiertas en otras categorías
- Alley (93,21% de valores faltantes) - tipo de acceso al callejón
- Fence (80,43% de valores faltantes) - calidad de la cerca
- Utilities - tipo de utilidades disponibles: contenía 2 valores ausentes, pero el resto de los valores menos 1 tenía el mismo valor.

Imputación:

- Valor fijo: En nuestro caso se han asignado None o 0 a variables que estaban vacías, pero indican valores referentes a una característica ausente, por ejemplo, datos del garaje si la casa no tiene garaje. Los datos modificados así son todos los que se refieren a garaje, chimenea, sótano y a las variables MasVnrType tipo de chapa de albañilería y MasVnrAre área de revestimiento de mampostería en pies cuadrados
- Media: esta opción se ha usado para variables continuas, en concreto con LotFrontage - pies lineales de calle conectados a la propiedad
- Moda para variables nominales (MSZoning, Functional, Exterior1st, Exterior2nd, Electrical, KitchenQual, SaleType)

Verificación del tipo de datos

Se verifican que los tipos de datos de las variables coinciden con las especificaciones, cruzando con el dataset de campos.

Las variables ordinales se convierten a numéricas basándonos en el orden establecido en la documentación, se añaden todos los posibles valores como factores aunque no existan datos en el dataset. Esta parte es posible que se pueda mejorar estudiando caso a caso, ya que el orden no tiene por qué indicar un valor directo, pero en general si parece plausible para este conjunto de datos.

Todas las variables nominales se convierten en tipo factor, de estas las variables con dos valores se convierten directamente a numéricas indicando 0 ausencia y 1 presencia del valor, CentralAir y Street que pasa a llamarse StreetPave.

Búsqueda de valores atípicos y eliminación outliers

Se revisan todas las variables continuas y discretas en busca de posibles valores atípicos, los posibles tratamientos sobre estos valores son:

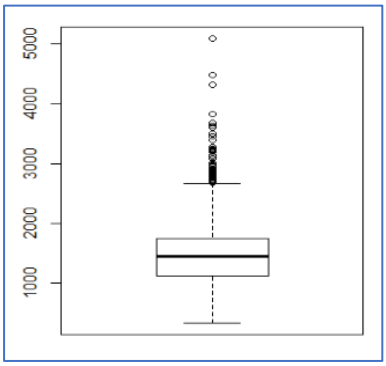
- Retirar la fila
- Asignar el siguiente valor más cercano a la mediana en lugar del valor atípico

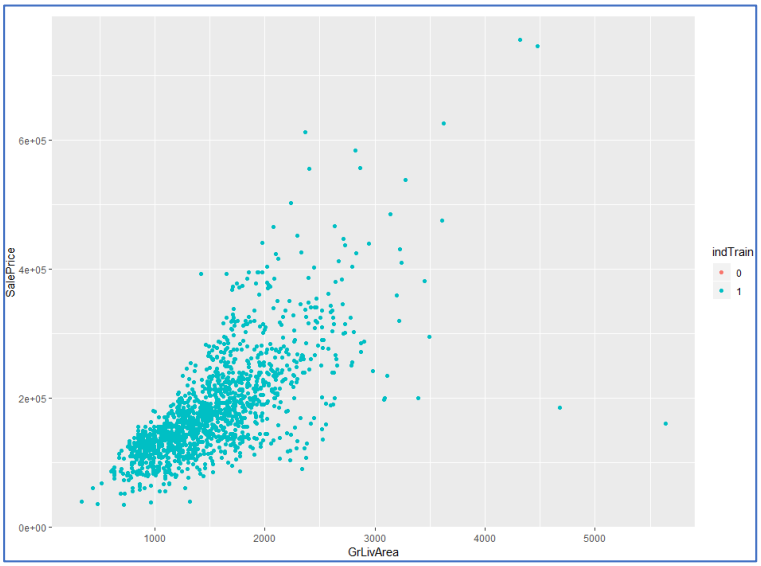
Para la búsqueda de valores atípicos, se realiza un estudio gráfico mediante:

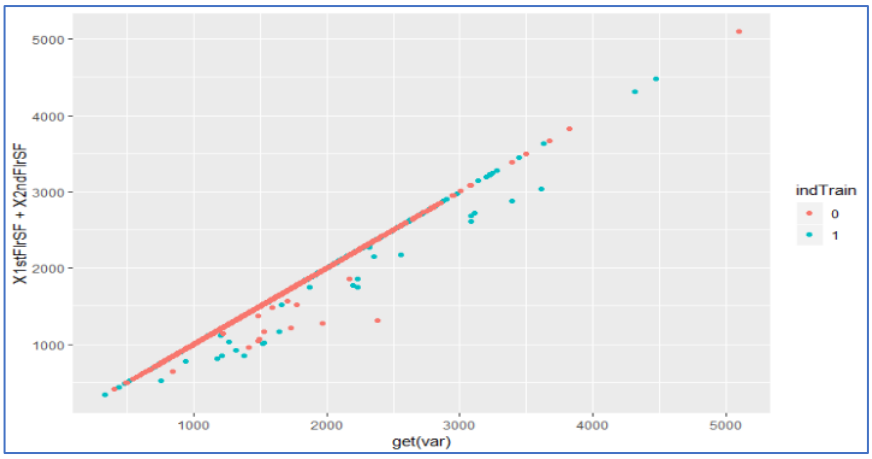
- Un BoxPlot sobre la variable a estudio

- Un gráfico de puntos, presentando la relación con la variable objetivo
- Gráficos de puntos presentando la relación con otras variables que puedan darnos información sobre los valores atípicos.

A modo de ejemplo se presenta el estudio sobre la variable GrLivArea:







Realizando este estudio sobre todas las variables continuas y discretas se han identificado los siguientes valores atípicos:

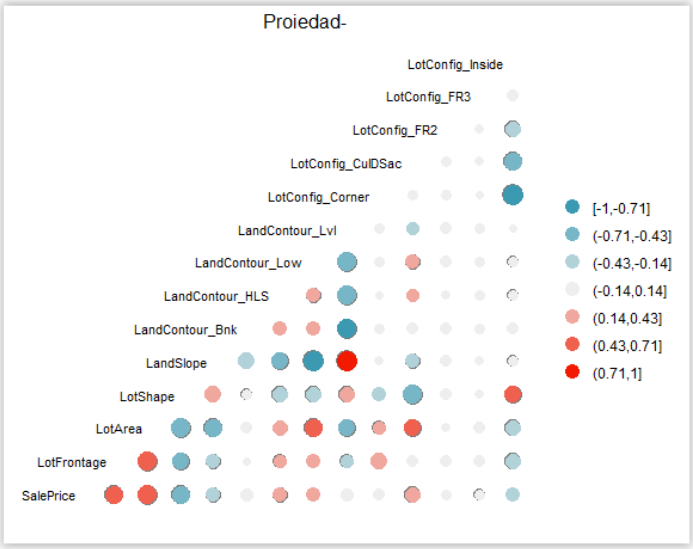
- GrLivArea: Existen 2 valores atípicos son muy altos para el precio que tienen en el conjunto de entrenamiento, estas filas se eliminarán al ser esta una variable principal para el proceso de predicción
- LotArea: Existen 4 valores claramente fuera de rango, creo variable nueva actualizándolos con los valores con la mediana según el tipo de construcción
- LowQualFinSF: Parece que existe un par de valores extraños, creo variable nueva y actualizo a la mediana de todos los valores no cero
- MasVnrArea: Identifico un valor extraño y se asigna la mediana
- WoodDeckSF: Parece que existe un valor extraño, sin embargo, existe la posibilidad de que sea una casa completamente de madera, pero como el valor está en el conjunto de test no se puede usar para entrenar y el modelo resultante no podrá calcular precios para casas solo de madera, por lo que actualizo el valor a la mediana según la superficie
- OpenPorchSF: Identifico un par de valores extraños Uno en el conjunto de entrenamiento, con un porche muy grande y un precio bajo y Otro en el conjunto de test, con una superficie muy grande
- EnclosedPorch: Parece que existen un valor extraño en el conjunto de test, con una superficie muy grande
- YearRemodAdd: Parece que a las casas que se construyeron antes de 1950 se les puso una fecha de remodelación 1950 Modifico la fecha de remodelación para casas anteriores a 1950 asignándoles la fecha de construcción
- GarageYrBlt: Los datos de esta variable parecen incorrectos por lo que es eliminada

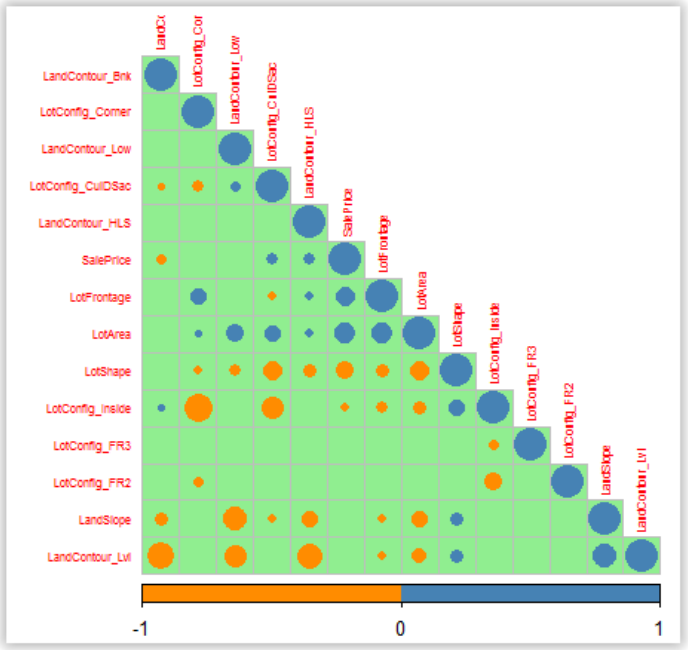
Estudio de correlaciones

Para estudiar las correlaciones, todas las variables deben ser numéricas, por lo que previamente al estudio he convertido las variables nominales a binarias.

El número tan elevado de variable, 73 después de la primera limpieza y 210 con las variables dummy, impide realizar una sola matriz que nos muestre las correlaciones, por lo que he optado por separar en varias matrices. He dividido las variables en grupos utilizando la clasificación obtenida de la documentación y grabada en el csv campos.

Para cada grupo realizo una representación gráfica mediante ggcorr y corrplot:





Además, he calculado tablas que presentan:

- las variables más correlacionadas con SalePrice
- las variables con más correlación entre ellas

Salida Fase 1

Contiene todos los datos (train/test) limpios (sin valores perdidos ni outliers), solo se han eliminado características con problemas en los datos.

- F01_dsDataAll (2917 Observaciones, 76 Variables)

Fase 2 - Ingeniería de características

En esta fase se profundiza en el análisis descriptivo del conjunto de datos, y se intentará modificar el conjunto de características para aumentar su eficacia predictiva.

La idea es generar variaciones sobre cómo realizar las transformaciones, tanto en el tipo de transformaciones a aplicar como en el orden en el que se realizan y pasar las diferentes salidas a las siguientes fases de tal forma que se puedan identificar que opciones son las mejores.

Esta fase se ha divido en dos subfases y cada una de ellas genera salidas distintas:

- 01 Ingeniería de características básica y PCA
- 02 Ingeniería de características mediante paquete recipes y selección manual

Entrada fase 2

- F01_dsDataAll: Todos los datos (train/test) (sin valores perdidos ni outliers)

01 Ingeniería de características básica

Normalización de variables

En el análisis inicial se detectó que la mayoría de las variables continuas están sesgadas por lo que he decido aplicar la función log a todas ellas. Para ello se crea una función que convierte una variable del dataset, pasada como parámetro aplicándole la función log, esta función se llama mediante apply sobre todos los campos continuos.

```
funcMutateLog <- function(var){
  dsDataAll[,var] <- log(dsDataAll[,var])
  assign('dsDataAll',dsDataAll,envir=.GlobalEnv) # Pendiente buscar solución más elegante
}

apply(dsCamposContinua, MARGIN=1, funcMutateLog)
```

Tratamiento de variables nominales - dummy

Todas las variables nominales se convertirán a numéricas binarias, para ello cada variable generara nuevas variables una por cada valor existente en el conjunto de datos, indicando como valor 0 o 1, ausencia o presencia del valor.

Existen multitud de formas en r de realizar esta transformación, yo he decidido usar dummy_cols del paquete fastDummies.

Eliminación de variables con varianza próxima a cero

Si una variable tiene casi todas las observaciones con un mismo valor, su varianza será próxima a cero. Estas variables pueden añadir más ruido que información, también dan problemas cuando se seleccionan los conjuntos de entrenamiento, ya que si en el conjunto de entrenamiento solo queda un valor puede producir que el entrenamiento sea erróneo.

He utilizado la función nearZeroVar() del paquete caret para seleccionar estas variables y se han eliminado del dataset general.

La eliminación de estas variables se debería realizar antes de la normalización, ya que después ya no tendrán la varianza cero, pero en este caso como solo se han normalizado las variables continuas, se ha decidido hacer posteriormente y se ha verificado que el resultado no empeoraba en la siguiente fase.

Salida Fase 02 01

El resultado final es un dataset con 92 variables, después de haber creado las variables dummy y eliminado los valores con varianza próxima a cero.

La salida de esta fase (F02_01_dsDataAll.RData) se ha utilizado para realizar una primera prueba completa del proyecto pasando a la fase 3 de selección de predictores, pero además se ha utilizados para una subfase posterior, donde se ha intentado reducir la dimensionalidad del problema mediante PCA (Análisis de componentes principales). con lo que se obtiene otra salida (F02_02_dsDataAll_PCA.RData).

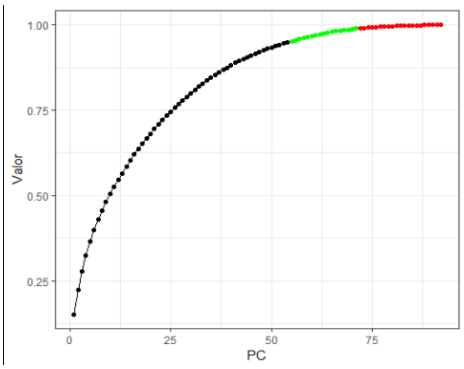
Reducción de dimensionalidad - Análisis de Componentes Principales

El objetivo del PCA es buscar una representación alternativa y reducida de las tuplas originales formadas por n atributos. Se buscan los k vectores ortogonales (k < n) que mejor representan los n atributos originales.

Para realizar el PCA es conveniente tener dos tareas previas:

- Escalado de las variables: las variables tienen que estar escaladas y normalizadas (media 0 y desviación estándar 1) ya que si alguna variable tiene una escala mayor puede dominar al resto. Utilizo scale para escalar todas las variables.
- Influencia de outliers: PCA es altamente sensible a outliers (se han tratado de identificar en la fase 1)

Una vez aplicado PCA se obtiene tantos componentes como variables originales, al ser tantos componentes y variables no se aprecia la representación con ggbiplot. Sin embargo, he utilizado summary(pca)$importance y en concreto la variable 'Cumulative Proportion' proporción acumulada de la variabilidad explicada por el componente principal y anteriores, para buscar el número de componentes a seleccionar.



Seleccionamos como número de componentes principales los que explican el 95%, 55.

02 Ingeniería de características mediante paquete recipes

En esta subfase se utilizará el paquete “recipes”, que facilita las transformaciones de las variables para realizar la ingeniería de características.

Para su utilización:

- Se crea un objeto récipe con la variable respuesta y los predictores, esto se realiza con el conjunto de entrenamiento.
- Sobre dicho objeto se definen los pasos a realizar para transformar los datos. En nuestro caso:
  - Eliminación de variables con varianza próxima a cero
  - Estandarización y escalado, sobre las variables numéricas
  - Binarización de variables nominales (dummy)
  - Eliminación de variables con varianza próxima a cero para variables dummy
- Se realiza un entrenamiento, donde se aprenden los parámetros necesarios para realizar las transformaciones.
- Se aplican las transformaciones a los conjuntos deseados, en nuestro caso se realizará sobre todos los datos, ya que será la entrada de la siguiente fase.

Salida Fase 02 02

El resultado final es un dataset con 86, la diferencia con el conjunto anterior 95 variables se explica por el orden de los pasos seguidos, al eliminar primero las variables con varianza cero y luego realizar normalizar se eliminan algunas variables más, aunque posteriormente es necesario volver a eliminar variables con varianza próxima a cero ya que al realizar entrenamiento algunos algoritmos fallan.

Selección manual de variables

He decidido realizar una prueba de selección manual de características desde el conocimiento del problema y apoyado con el estudio de correlaciones realizado en la fase anterior. Para ello parto de las variables modificadas con recipe.

Se han recorrido todas las categorías creadas en los campos y por cada una de ellas se han seleccionado las variables más correlacionadas con el precio, aunque se han excluido algunas que tenía una clara relación con otra variable ya seleccionada, por ejemplo, LotFrontage se eliminó por estar muy relacionado con LotArea. Además, se han añadido algunas variables que, aunque no dan una gran correlación con la variable objetivo son consideradas importantes para el problema.

Salida fase 2

En esta fase se han generado 4 salidas que se utilizarán como entrada en la fase 3, con lo que se podrá probar distintas formas de realiza la ingeniería de características.

- F02_01_dsDataAll.RData
- F02_02_dsDataAll_PCA.RData
- F02_03_dsDataAll_Recipe.RData
- F02_04_dsDataAll_SelManual.RData

Todos estos ficheros contienen un dataset llamado dsDataAll, con 3 campos iniciales SalePrice (que será el precio aplicada la función log), indTrain y Id.

Observaciones sobre la fase 2

Existen transformaciones que se deberían estudiar variable a variable, como el escalado o lo normalización. Desafortunadamente no me ha dado tiempo a realizar análisis más detallados para poder aplicar estas técnicas de forma más eficaz.

Fase 3 - Selección de predictores con caret

El objetivo es estudiar las distintas formas de reducir el volumen de características que ofrece caret, de tal forma que únicamente los predictores que están relacionados con la variable respuestas se incluyan en el modelo.

Los métodos de selección de predictores se pueden clasificar como:

- Métodos wrapper: evalúan múltiples modelos utilizando procedimientos que agregan y/o eliminan predictores para encontrar la combinación óptima que maximice el rendimiento del modelo, son algoritmos de búsqueda donde los predictores son las entradas y el modelo a optimizar es la salida.
    - Eliminación de características recursivas
    - Algoritmos genéticos
    - Simulated annealing
- Métodos de filtro: Analizan la relación que tiene cada predictor con la variable respuesta, evaluando la relevancia de los predictores fuera de los modelos y seleccionando los que pasan algún criterio.

Además, cada uno de estos métodos puede utilizar distintos algoritmos (regresión lineal, naive bayes, random forest) y métodos de entrenamiento (Validación cruzada o bootstrapping).

- He seleccionado como método de evaluación, la validación cruzada con 5 particiones.

Se implementarán y analizarán los resultados de varios de estos modelos con el objetivo de realizar la selección más adecuada de predictores.

Caret permite simplificar la utilización de distintos modelos realizando llamadas a funciones con una sintaxis similar y cambiando ligeramente algunos parámetros, básicamente solo es necesario realizar las siguientes acciones:

- Crear objeto controlador del modelo: donde se indica la función a utilizar (regresión lineal, naive bayes, random forest) y el método de entrenamiento (Validación cruzada o bootstrapping)

- Ejecutar la función de selección de predictores donde se pasan los datos indicando la variable objetivo y el control definido en el paso anterior, en ocasiones también es necesario añadir algunos parámetros.

Toda esta fase trabajará exclusivamente con el conjunto dsTrain.training ver Gestión de los dataset - Pruebas de Test.

Entrada fase 3

Los conjuntos de entrada serán las salidas de la fase anterior, salvo PCA donde no será necesario realizar la selección de predictores, por lo que esta fase se ejecutara 3 veces, el conjunto es el dataset llamado dsDataAll que incluye todos los datos (train/test), los campos SalePrice, inTrain e Id y el resto de información que dependiendo del origen son campos distintos.

Eliminación de características recursivas

RFE (Recursive feature elimination) de Caret ofrece multitud de posibilidades para ejecutar estas funciones, yo he implementado varias de ellas:

- Regresión lineal y validación cruzada
- Randon Forest y validación cruzada

El parámetro "size" permite determinar sobre que tamaños de conjuntos de variables se desea que busque el algoritmo. En este caso y después de realizar varias pruebas he optado por buscar en conjuntos con tamaños:

- (5 10 12 14 16 18 20 25 30 35 40 45 50 55 60)

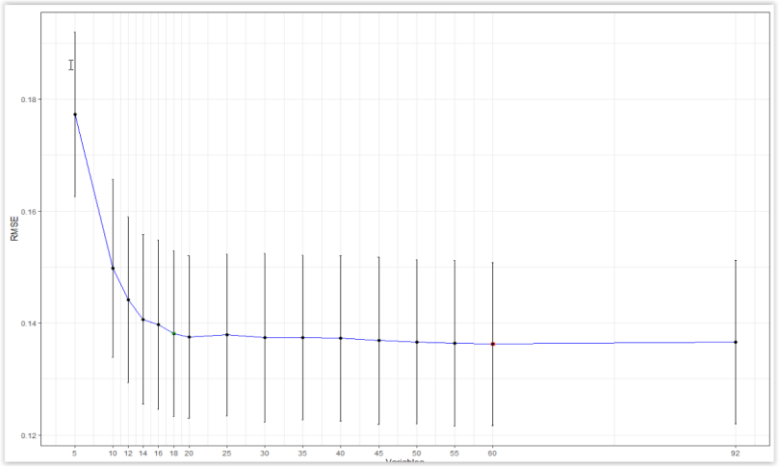
Esto me permite dibujar gráficas para ver la evolución de RMSE con los distintos subconjuntos, caret además añade un cálculo con todas las posibles variables. En algunos casos ha sido necesario refinar la búsqueda, modificando estos conjuntos.

Resultados

He implementado un estudio de los resultados en los que se muestra el resultado y una gráfica que muestra la evolución de RMSE según crece el número de predictores.

Como ejemplo se muestra la salida de la eliminación recursiva con randon forest y validación cruzada sobre el conjunto inicial de prueba con 92 posibles predictores.

```
Recursive feature selection

Outer resampling method: Cross-Validated (5 fold, repeated 5 times)

Resampling performance over subset size:

 Variables   RMSE Rsquared      MAE  RMSESD RsquaredSD    MAESD Selected
         5 0.1773   0.8147 0.12900 0.01475     0.02732 0.007566
        10 0.1498   0.8670 0.10412 0.01595     0.02780 0.008373
        12 0.1441   0.8772 0.09935 0.01482     0.02476 0.007010
        14 0.1407   0.8841 0.09664 0.01514     0.02381 0.007493
        16 0.1397   0.8858 0.09590 0.01510     0.02338 0.007621
        18 0.1381   0.8886 0.09506 0.01479     0.02166 0.007244
        20 0.1374   0.8903 0.09432 0.01451     0.02089 0.007003
        25 0.1379   0.8902 0.09458 0.01443     0.02057 0.007113
        30 0.1374   0.8910 0.09414 0.01505     0.02165 0.007477
        35 0.1374   0.8911 0.09393 0.01470     0.02113 0.007060
        40 0.1373   0.8916 0.09390 0.01476     0.02105 0.007049
        45 0.1369   0.8922 0.09376 0.01496     0.02126 0.007298
        50 0.1366   0.8929 0.09355 0.01465     0.02069 0.007237
        55 0.1364   0.8932 0.09341 0.01484     0.02097 0.007166
        60 0.1362   0.8936 0.09327 0.01458     0.02048 0.007250        *
        92 0.1366   0.8935 0.09342 0.01461     0.02046 0.007241

The top 5 variables (out of 60):
   GrLivArea, OverallQual, TotalBsmtSF, X1stFlrSF, BsmtFinSF1

  media_RMSE media_Rsquared
1  0.1412275       0.8835616
```



En esta salida se puede observar como la capacidad de predicción del modelo mejora de forma significativa con 15 predictores y después se estanca con mejoras muy pequeñas según se van añadiendo nuevos predictores, aunque el menor RMSE se logra con todos ellos.

Por el principio de parsimonia que nos indica que el modelo más simple es posiblemente el mejor seleccionaríamos un conjunto de 15 predictores, sin embargo, he realizado pruebas seleccionando 15, 25, 30 y 40 predictores y entrenando distintos modelos con ellas y he logrado los mejores resultados los he logrado con 25 y 30.

Selección de características usando algoritmos genéticos

Los algoritmos genéticos (GA) imitan la selección natural para encontrar valores óptimos de alguna función, se basa en:

- Generar un conjunto de soluciones (población)
- Seleccionar las soluciones (individuos) dentro de esta población con mejor resultado.
- Combinar aleatoriamente los individuos seleccionados
- Añadir mutaciones aleatorias
- Iterar multitud de veces

Los parámetros básicos por determinar son:

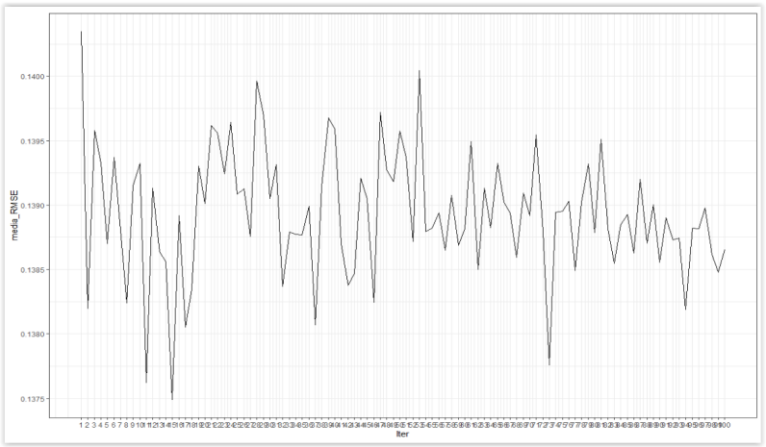
- popSize: número de subconjunto en cada iteración (número de individuos de la población)
- iters: es el número de iteraciones, yo he realizado una prueba con 20 y ampliado posteriormente a 100, ya que los resultados eran bastante pobres.

Por otro lado, es necesario determinar que algoritmo y parámetros de entrenamiento se utilizan para generar las soluciones, en cada iteración para cada individuo, he seleccionado randon forest y validación cruzada, pero sin repetición ya que el algoritmo es muy pesado al tener que realizar los cálculos para el número de individuos en cada iteración.

Resultados

En este caso se puede ver la evolución de las distintas iteraciones mostrando la media de RMSE:

Iter <int>	media_RMSE <dbl>	media_Rsquared <dbl>
15	0.1374921	0.8938398
11	0.1376238	0.8936377
73	0.1377606	0.8937565
17	0.1380531	0.8930309
37	0.1380711	0.8928991
94	0.1381884	0.8926521
2	0.1381967	0.8928602
8	0.1382404	0.8925305
46	0.1382428	0.8923882
18	0.1383470	0.8921662
32	0.1383690	0.8925707
42	0.1383773	0.8923376
43	0.1384702	0.8921089
99	0.1384804	0.8918744
77	0.1384920	0.8922651
62	0.1385003	0.8920539
83	0.1385467	0.8923472
90	0.1385548	0.8921570
14	0.1385583	0.8917722
68	0.1385938	0.8918559
98	0.1386176	0.8914195
86	0.1386290	0.8918743



El resultado es la selección de un conjunto de 46 predictores como mejores características, en la iteración 15, que, aunque reduce la dimensionalidad sigue siendo un número muy alto.

He realizado pruebas de entrenamiento de distintos modelos con el conjunto de predictores sugerido por el algoritmo genético, pero los resultados son inferiores a los obtenidos con los seleccionados por eliminación de características recursivas, en este problema concreto.

El tiempo de ejecución del algoritmo, que ha llegado a más de 6 horas, y los resultados que no son muy interesantes, no se observa en la tendencia una mejora según avanzan las iteraciones, me ha llevado a descartar este algoritmo (solo se ha usado en el primer conjunto de datos).

Selección de características utilizando filtros univariados

SBF (selection by filter) de Caret al igual rfe ofrece multitud de posibilidades para ejecutar la selección de características, yo he implementado varias de ellas:

- Regresión lineal y validación cruzada
- Randon Forest y validación cruzada

En este caso no se especifica el número de predictores que se desea.

Resultados

Como resultado se obtiene la mejor selección de variables, un ejemplo de salida es:

```
Selection By Filter

Outer resampling method: Cross-Validated (5 fold, repeated 5 times)

Resampling performance:

   RMSE Rsquared     MAE   RMSESD RsquaredSD    MAESD
 0.1319   0.8951 0.09441 0.007469    0.01357 0.004087

Using the training set, 84 variables were selected:
   LotFrontage, LotArea, LotShape, OverallQual, YearBuilt...

During resampling, the top 5 selected variables (out of a possible 85):
   BedroomAbvGr (100%), BldgType_1Fam (100%), BsmtExposure (100%), BsmtFinSF1 (100%), BsmtFinType1
(100%)

On average, 81 variables were selected (min = 79, max = 83)
```

Después de ejecutar la selección de predictores para el conjunto inicial (92 predictores), los resultados han sido:

- Regresión lineal 84 variables RMSE 0.1319
- Randon Forest 79 variables RMSE 0.1389

Con respecto al conjunto de características seleccionas con recipe (86 variables) los resultados fueron:

- Regresión lineal 80 variables RMSE 0.1316
- Randon Forest 76 variables RMSE 0.1382

Como puede apreciarse para ambos casos el número de características seleccionadas es prácticamente todo el conjunto por lo que no se utilizarán para entrenar los modelos.

Salida fase 3

Como resultado del proceso anterior tenemos varios conjuntos de posibles candidatos a mejores predictores. Para seleccionar los predictores definitivos nos hemos fijado en el RMSE de entrenamiento que generan los distintos algoritmos, intentando minimizar el número de predictores.

- RFE, buscaremos dos subconjuntos de características una con el mejor rendimiento y la mejor de forma absoluta (menor RMSE).
  - F03_11_dsDataSelVar_rfe_MejorRendimiento_topN.RData
  - F03_12_dsDataSelVar_rfe_MenorRMSE_topN.RData
- Algoritmos genéticos se realizará una sola prueba con todas las características seleccionadas.
  - F03_13_dsDataSelVar_ga_100_N.RData
- Mezcla intentaremos determinar un subconjunto de 30 características teniendo en cuenta todos los modelos (30 es el número de características que mejor me ha funcionado en pruebas iniciales para algunos modelos de ejemplo).
  - F03_14_dsDataSelVar_mezcla_30.RData
- Todos los predictores
  - F03_15_dsDataSelVar_Completo.RData

Los ficheros de salida guardarán un dataSet llamado dsDataSelVar, con 3 campos iniciales SalePrice (que será el precio original y aplicada la función log), indTrain e Id y el resto de las características seleccionadas.

Las salidas se almacenarán en un directorio con el nombre del fichero de entrada en cada ejecución y se utilizarán como entrada en la fase 3, con lo que se probarán las distintas formas de realizar la selección de características.

Observaciones sobre la fase 3

El tiempo para generar los predictores con los distintos métodos ha sido muy grande, además he realizado bastantes pruebas antes de llegar a la configuración definitiva, por lo que esta fase me ha llevado más tiempo del previsto inicialmente.

El número de posibilidades para seleccionar predictores y sus resultados heterogéneos, me llevo a realizar una selección muy grande de conjuntos para pasar a la siguiente fase.

Fase 4 - Creación de modelos predictivos con caret

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

Entrada fase 4

Los conjuntos de entrada serán las salidas de la fase anterior. Por lo tanto, esta fase se ejecutará múltiples veces. El conjunto de entrada es el dataset llamado dsDataAllVarSel que incluye todos los datos (train/test) y los campos SalePrice aplicado log, indTrain, Id y los predictores seleccionados.

Creación y entrenamiento de modelos

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

Para crear y entrenar un nuevo modelo solo será necesario realizar las siguientes acciones por cada modelo:

- Crear control de entrenamiento
- Definir hiperparámetros
  - Se puede definir distintas posibilidades para los parámetros y el entrenamiento evaluará todos ellos y seleccionará los que mejor resultados den.
  - También se puede indicar que sea caret quien asigne y evalué parámetros.
- Ejecutar la función train pasando:
  - Datos (variable objetivo y resto de datos)
  - Método (algoritmo a utilizar)
  - Hiperparametros
  - Métrica para selección, en nuestro caso RMSE
  - Control de entrenamiento (el control creado en el paso 1)

He seleccionado como método de evaluación, la validación cruzada con 5 particiones y 5 repeticiones y se utilizará para todos los modelos, por lo que se crea una sola vez.

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

Para realizar el entrenamiento de los modelos se utilizará el conjunto dsTrain.training, con la variable SalePrice como variable objetivo y el resto como predictores, el dataSet viene de la etapa anterior con los predictores ya seleccionados.

Por lo tanto, la llamada para crear y entrenar un modelo será de la siguiente forma:

```
modelo <- train(SalePrice ~ .
                , data = dsTrain.training
                , method = modelo
                , tuneGrid = hiperparámetros
                , metric = "RMSE"
                , trControl = fitControl)
```

Donde solo será necesario configurar los hiperparámetros y el algoritmo a utilizar (método).

Además, caret ofrece una salida con funciones estandarizadas para todos los modelos, esto me ha permitido generar una función para presentar el resultado para todos los modelos, en la cual se pasa como parámetro el modelo entrenado y si tienen hiperparámetros, y la función presenta la siguiente información:

- Modelo final: Resumen del modelo seleccionado en el entrenamiento y cuáles son los mejores parámetros.
- Estudio del RMSE obtenido en la validación

Para ello se presentan dos gráficas:

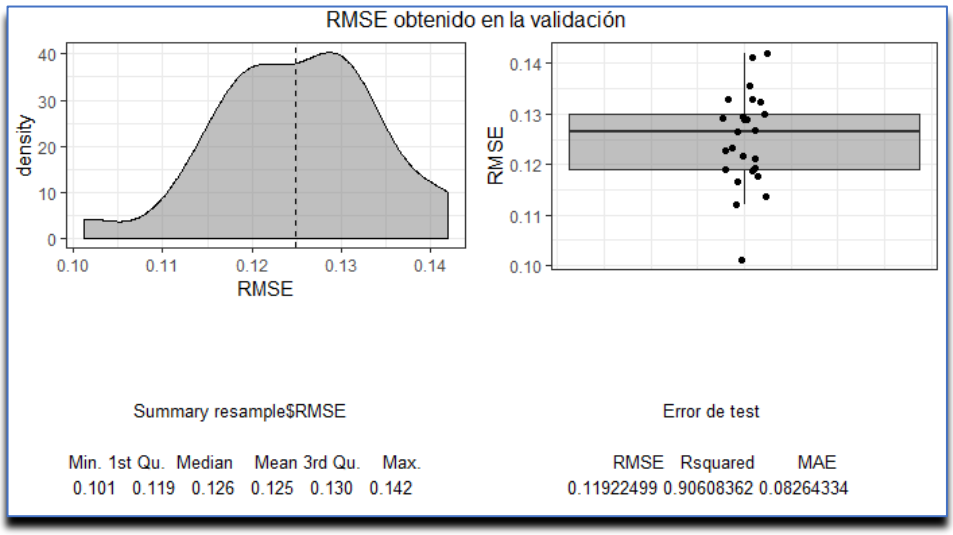
- Gráfica de densidad de RMSE de los distintos cálculos realizados en el entrenamiento.
- Boxplot de RMSE de los distintos cálculos

El resumen del RMSE obtenido para el entrenamiento

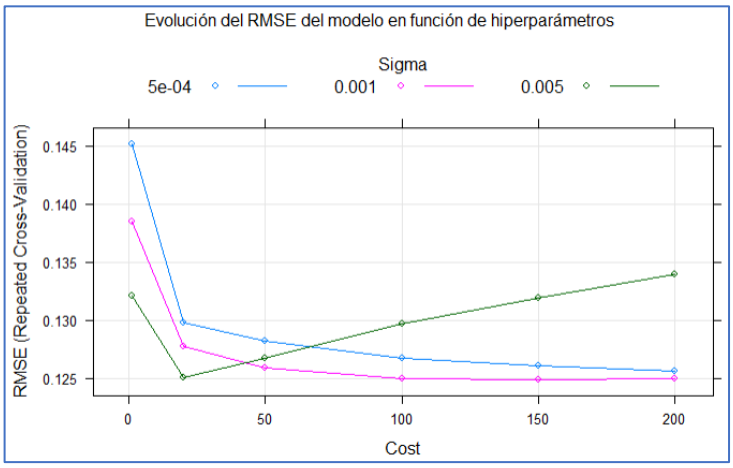
- Summary de resample$RMSE

Y por último el error de test

- Para ello la función llama a predict sobre el modelo con el conjunto dsTrain.CV ver Gestión de los dataset - Pruebas de Test)



- Si tiene hiperparámetros presenta Evolución del RMSE del modelo en función de hiperparámetros (función plot)



Caret permite usar paralelismo, yo realice algunas pruebas subiendo el número de cores pero el rendimiento en mi ordenador era prácticamente el mismo por lo que no se ha incluido en la implementación.

Ajuste de hiperparámetros

Para ajustar los hiperparámetros he partido:

- de una asignación inicial, si la he encontrado en algún ejemplo
- o he utilizado tuneLength si no sabía que valores podían ser los adecuados.

Esta asignación es una matriz con múltiples posibilidades para cada hiperparámetro.

```
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))
```

Después de una primera ejecución he estudiado la salida que genera el gráfico de evolución de los distintos hiperparámetros, con ello he modificado los rangos posibles de los parámetros para refinar la búsqueda entrando en un proceso iterativo hasta encontrar los hiperparámetros que obtienen el menor RMSE.

Modelos

En este apartado se clasificarán y describirán los modelos que he probado.

- 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 (method = "lm"): 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.

  - Generalized Linear Model (method = "glm"): es una generalización flexible de la regresión lineal, permite que el modelo lineal se relacione con la variable de respuesta a través de una función de enlace. Este modelo parece más apto para nuestro problema ya que permite que la variable respuesta tenga una distribución arbitraria, nuestra variable es solo positiva y varia gran escala, es una distribución sesgada.

- 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. [Carmona 2013]

  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 (method = "svmLinear"): Permite solo seleccionar líneas (o hiperplanos)
  - Support Vector Machines with Radial Basis Function Kernel (method = "svmRadial"): Permiten seleccionar círculos (o hiperesferas)

  Estudiando los resultados los mejores resultados se obtienen con el kernel Radial, por lo que podemos suponer que el precio no se puede separar con hiperplanos.

- 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 bienpara 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:

  - Gradient Boosting – XGBoost (Extra Gradient boosting) (method = "xgbTree")

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. Yo he realizado unos pequeños ajustes en los hiperparámetros, pero no he logrado un rendimiento alto en test, parece que en mi implementación el modelo está realizando un sobreajuste. Otra desventaja es el tiempo de entrenamiento considerablemente mas alto que en otros modelos.

- Random Forest (method = "ranger")
  Al igual que en el caso anterior Random Forest utiliza una combinación de árboles, en este caso cada árbol depende de los valores de un vector aleatorio. [Wikipedia 1]

  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.

- Stochastic Gradient Boosting (method = "gbm")
  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.

- k-Nearest Neighbors (method = "knn")
  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.

  Los resultados obtenidos en el TFM, son realmente malos comparados con los otros modelos. Además, después de ejecutarlo unas cuantas veces con los mismos conjuntos de entrenamiento, vi que el parámetro k varia en cada ejecución de forma significativa.

- LASSO (method = "lasso")
  0perador de mínima contracción y selección absoluta. (least absolute shrinkage and selection operator) se utiliza para modelos de sistemas no lineales y trata de minimizar la Suma de Cuadrados Residuales (RSS).

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

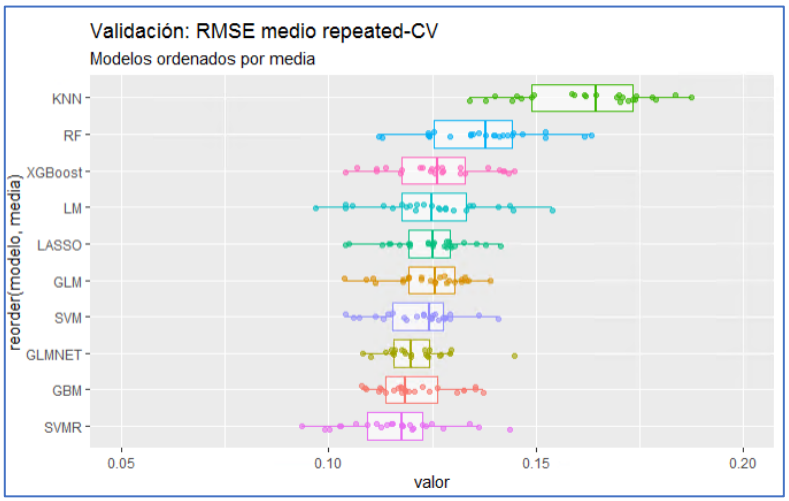
- Elasticnet (method = "glmnet")
  Es una combinación de LASSO y Ridge regression, donde predictores altamente correlacionados presentan coeficientes estimados similares.

Cada modelo entrenado se guarda en el directorio correspondiente a la ejecución de la fase, que se nombrara por el conjunto de datos y el proceso de selección utilizados, para poderlos diferenciar posteriormente.

Comparación de modelos

Para comparar el resultado de los modelos usaremos las métricas de validación calculadas en el entrenamiento, que se pueden obtener para una lista de modelos mediante la función resamples.

Presentamos los resultados comparando los distintos modelos mediante una gráfica para los distintos valores RMSE durante el entrenamiento y la media y un listado ordenado por la media obtenida.

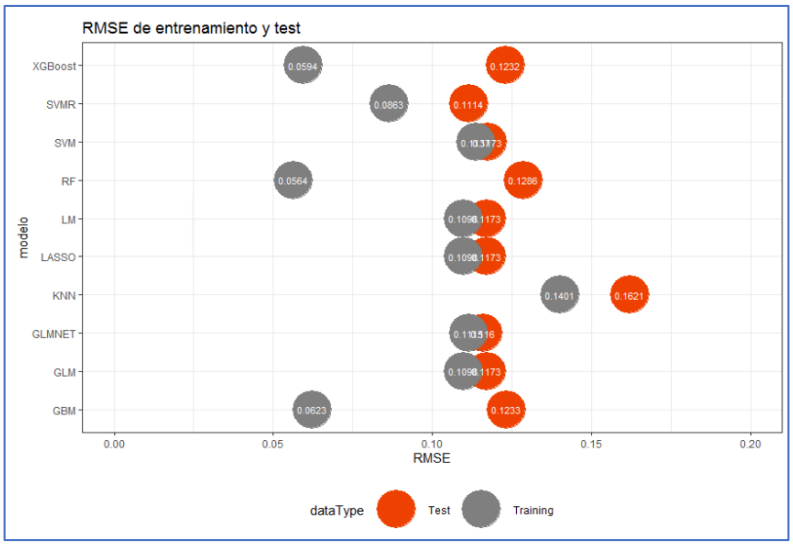


modelo	MAE	RMSE	Rsquared
SVMR	0.08276858	0.1196109	0.9138887
GBM	0.08468081	0.1210356	0.9115041
GLMNET	0.08824486	0.1234036	0.9077945
GLM	0.09000497	0.1261941	0.9037491
XGBoost	0.08934867	0.1264903	0.9036735
LM	0.08964391	0.1266483	0.9028908
LASSO	0.09039037	0.1272500	0.9012155
SVM	0.09100315	0.1278660	0.9013204
RF	0.09392209	0.1370811	0.8929737
KNN	0.17046745	0.2350589	0.6652512

Calcularemos también, las métricas de error de test, que se obtendrán aplicando extractPrediction a la misma lista y pasando como conjunto de test dsTrain.CV. Con esto obtengo una lista de los modelos ordenados por su error de test

object	Test	Training
SVMR	0.1114359	0.08632997
GLMNET	0.1159645	0.11149980
GLM	0.1172617	0.10981097
LM	0.1172617	0.10981097
LASSO	0.1172617	0.10981097
SVM	0.1172823	0.11370909
XGBoost	0.1231579	0.05938251
GBM	0.1232789	0.06232819
RF	0.1285580	0.05639253
KNN	0.1621204	0.14007632

Vemos que el orden no es el mismo si consideramos el error de test como criterio en vez del error de entrenamiento. Si presentamos los dos valores juntos en una solo gráfica, podemos ver como ciertos modelos tienen un RMSE de entrenamiento muy bajo pero el de test es muy alto, esto puede indicar que en estos modelos existe sobreentrenamiento.



Métricas globales

Una vez calculadas las métricas estas se guardarán en un dataset en el cual se especificará tanto el origen de los datos, ingeniería de características realizada como el proceso de selección de predictores utilizado, acumulando de esta forma toda la información de las distintas ejecuciones de esta fase.

Este conjunto nos permitirá estudiar no solo que modelos tienen las mejores métricas sino también el tipo de selección de predictores y la ingeniería de características usada.

Salida fase 4

Como resultado tenemos una lista con las métricas de todos los entrenamientos y modelos utilizados.

Observaciones sobre la fase 4

Es destable la facilidad que ofrece Caret para entrenar modelos, prácticamente solo hay que centrarse en la selección del algoritmo y la selección del conjunto de hiperparámetros. Esto hace muy sencillo agregar nuevos modelos a las pruebas.

Sin embargo, la multitud de tipos de algoritmos y dentro de estos la variedad de implementaciones, hace muy difícil decidir cuál puede ser adecuada para el problema. Además, el ajuste de parámetros en ciertos modelos lleva bastante tiempo, entrando en un bucle de prueba y error.

Fase 5 - Selección del modelo definitivo y presentación de resultados

Selección del mejor modelo

En total se han estudiado 11 modelos y se han realizado más de 100 entrenamientos distintos. Seguidamente presento una gráfica con el error de Test obtenido en las distintas ejecuciones realizadas, excluyendo el modelo KNN que claramente está muy alejado del resultado de los demás modelos:

También presento las medias de RMSE por tipo de ingeniería de características, selección de predictores y tipo de modelo:

modelo <fctr>	media <dbl>
SVMR	0.1163018
GLMNET	0.1226111
LASSO	0.1231027
GLM	0.1231027
LM	0.1231027
SVM	0.1234297
GBM	0.1261320
XGBoost	0.1275490
RF	0.1315715

9 rows

OrigenF2 <fctr>	media <dbl>
F02_03_dsDataAll_Recipe	0.1220452
F02_01_dsDataAll	0.1239695
F02_04_dsDataAll_SelMar	0.1280397
F02_02_dsDataAll_PCA	0.1290355

4 rows

OrigenF3 <fctr>	media <dbl>
F03_12_dsDataSelVar_rfe_MenorRMSE_top55	0.1220882
F03_15_dsDataSelVar_Completo	0.1227458
F03_13_dsDataSelVar_ga_100_46	0.1230884
F03_12_dsDataSelVar_rfe_MenorRMSE_top60	0.1236795
F03_14_dsDataSelVar_mezcla_31	0.1237774
F03_11_dsDataSelVar_rfe_MejorRendimiento_top18	0.1257100
F03_1_dsDataAllVarSel_PCA	0.1290355

Las conclusiones que obtenemos de estos datos son:

- La selección de predictores con mejores resultados ha sido la realizada con Recipe

- El proceso de selección de características no ha conseguido reducir el número, el mejor conjunto en casi todos los modelos ha sido el formado por todas las características de la selección Recipe, se podría usar como alternativa la selección de los 55 mejores predictores realizada con rfe, pero parece que esto nos aleja del objetivo.
- Sin duda con los entrenamientos realizados el modelo SVMR es el que mejor se comporta.

Por lo tanto, he seleccionado el modelo Support Vector Machines with Radial Basis Function Kernel como mejor modelo y como conjunto de predictores todos los generados con Recipe.

Presentación de resultados

Una vez seleccionado el modelo, lo ejecutamos contra el conjunto de test, para verificar el resultado, también he verificado la conversión a dólares.

Para realizar la subida de las predicciones a la competición Kaggle he realizado las siguientes acciones:

- Generamos las predicciones para el conjunto de test original
- Aplicamos la función exp a la predicción para calcular la predicción en dólares
- Generamos el fichero con las predicciones

En la entrega he obtenido un Score de 0.12401 que es un 0.01258 más de lo que había obtenido en pruebas de test. La posición obtenida en la competición ha sido 1454 de 4548.

Mejora de modelos

Se ha intentado mejorar la predicción de dos de los modelos mediante una búsqueda más exhaustiva de los hiperparámetros.

- Support Vector Machines with Radial Basis Function Kernel
- XGBoost

Aunque se logrado mejorar con los datos de entrenamiento los resultados finales no han sido mejorado respecto al modelo generado previamente.

Entrega y conclusiones

Conclusiones

La realización del proyecto fin de máster sobre la competición de Kaggle, me ha permitido enfrentarme con muchas de las etapas necesarias para generar un modelo de predicción. La metodología seguida, dividiendo el proyecto en fases e iterando sobre ellas, creo que ha sido acertada ya que de esta forma he podido ir probando distintos métodos y refinando los resultados, pero lo que es más importante, aprendiendo nuevas técnicas.

El nivel de automatización en algunas de las tareas como la ingeniería de características con récipe o la selección de predictores con caret, hacen muy cómodo estos procesos, pero se pierde el conocimiento de porque se realizan ciertas tareas. En este sentido haber realizado previamente estas tareas de forma “manual” me ha servido para comprender estos procesos.

En la selección de predictores no he logrado realizar una reducción significativa del número de estos, aunque le he dedicado grandes esfuerzos, creo que esto es debido al tipo de problema y al escaso número de observaciones.

Con respecto al entrenamiento y selección de los modelos, como acabo de comentar es extremadamente sencillo entrenar modelos, pero bastante complicado la selección del mejor de ellos y sus parámetros, sobre todo cuando no tienes experiencia previa.

Uno de los grandes problemas que he tenido es mi falta de conocimientos estadísticos, aunque en el master ha habido un módulo dedicado a ello, 2 semanas se quedan cortas para comprender por qué se realizan ciertas acciones o como seleccionar un modelo de forma consistente.

Visibilidad del trabajo

Todo el proyecto esta subido a la plataforma Github donde se puede consultar tanto el código como los datos generados en las distintas fases:

- https://github.com/JuanCarlosSantiagoCulebras/TFM_UNED_Curso1819_JCSC

Dada la extensión de la primera fase del proyecto y la documentación tan extensa que genera, he decidido realizar un markdown de resumen en un solo fichero con el trabajo definitivo, por cada fase se ha incluido:

- Fase 1 – Análisis y Limpieza inicial: No se incluirá el análisis del dataset solo las transformaciones realizadas
- Fase 2 – Ingeniería de características mediante paquete recipes
- Fase 3 – Selección de predictores con caret, solo se incluye un ejemplo Eliminación de características recursivas con Randon Forest y validación cruzada
- Fase 4 – Creación de modelos predictivos con caret, se incluirán 5 modelos.
- Fase 5 – Selección del modelo definitivo y presentación de resultados

Este resumen se ha subido a RPubs:

- http://rpubs.com/Soka/HousePricesCaret

También se ha subido a la plataforma Kaggle como un NoteBook en una versión en inglés:

- https://www.kaggle.com/jcsantiago/house-prices-model-selection-with-caret

Referencias

Gran parte de este proyecto se ha realizado sobre la guía de RPubs “Machine Learning con R y caret” de Joaquín Amat Rodrigo.

De Cock, Dean (2011) - *Ames, Iowa: Alternative to the Boston Housing Data as an End of Semester Regression Project* dese

Carmona, Enrique J. (2013) *Tutorial sobre Máquinas de Vectores Soporte (SVM)* Dpto. de Inteligencia Articial, ETS de Ingeniería Informática, Universidad Nacional de Educación a Distancia (UNED)

http://jse.amstat.org/v19n3/decock.pdf
http://topepo.github.io/caret
https://rpubs.com/Joaquin_AR/383283
https://www.machinelearningplus.com/machine-learning/caret-package/
https://www.datacamp.com/community/tutorials/pca-analysis-r
[wikipedia 1] https://es.wikipedia.org/wiki/Random_forest
http://www.diegocalvo.es/xgboost/
https://www.aprendemachinelearning.com/clasificar-con-k-nearest-neighbor-ejemplo-en-python/
https://www.kaggle.com/pelkoja/visual-xgboost-tuning-with-caret
https://medium.com/@aravanshad/gradient-boosting-versus-random-forest-cfa3fa8f0d80
https://educationalresearchtechniques.com/2017/04/14/elastic-net-regression-in-r/
https://www.kaggle.com/benumeh/advanced-prediction-of-house-prices-top-10

Anexos

He generado desde los ficheros markdown del proyecto la documentación complementaria a esta presentación con todo el código y estudios gráficos:

- Anexo 1 -Estudio de campos.xlsx
- Anexo 2 - F01_TFM_Preprocesamiento_Estudio_y_Limpieza_inicial.pdf
- Anexo 3 - F02_1_TFM_Ingenieria_de_caracteristicas.pdf
- Anexo 4 - F02_2_TFM_Ingenieria_de_caracteristicas_Recipe.pdf
- Anexo 5 - F03_TFM_Seleccion_de_predictores.pdf
- Anexo 6 - F04_TFM_Seleccion_de_modelo.pdf
- Anexo 7 - F05_TFM_Modelo_definitivo.pdf
- Anexo 8 - F10_TFM_Resumen.pdf