Inteligencia de Negocio (2017-2018)
Grado en Ingeniería Informática
Universidad de Granada

Práctica 3

Juan José Sierra González
jjsierra103@gmail.com

9 de enero de 2018

Índice

Índice de figuras

Índice de tablas

1. Introducción

La última práctica de la asignatura de Inteligencia de Negocio consiste en participar en una competición dentro de la conocida plataforma de ciencia de datos Kaggle. Esta competición se realiza a nivel mundial pero los alumnos de la UGR competimos entre nosotros para ver quién consigue la mejor solución, es decir, obtener un error cuadrático medio menor sobre las predicciones de la variable respuesta. Como la competición es de la categoría "Getting started" dentro de la plataforma está indicado que es una competición orientada al aprendizaje, y se pueden encontrar muchos kernels y manuales de ayuda para facilitar la comprensión y el estudio del problema.

El problema al que nos enfrentamos es tratar de predecir el precio de aproximadamente 1500 viviendas de Ames, Iowa, a partir de otras 1500 viviendas de las que conocemos 79 variables descriptivas y una variable respuesta (el precio de venta de la casa) que será la que tendremos que predecir.

2. Tabla de resultados

En esta competición he realizado un total de **18 subidas a Kaggle**. Mi posición final fue la 456, con un error cuadrático medio de 0.11692. A continuación se mostrará una tabla con los resultados obtenidos en cada una de las subidas y una pequeña descripción del modelo empleado. Así se podrá observar de forma clara la evolución que se ha ido produciendo en los modelos y cómo esto ha influido menor o mayormente en el resultado obtenido.

Subida	Posición	Score	Fecha	Hora	Train RMSLE	Preprocesado	Algoritmos utilizados y parámetros
1	1049	0.12611	28/12/2017	17:30	ElasticNet: 0.11921 GBoosting: 0.07561	Eliminación características >50% NA y no correladas, imputación de NA en train, logaritmo etiquetas, dummies	ElasticNet(alpha=[0.0001..10], l1ratio=[0.01..0.99]) y GradientBoosting (estimadores=3000, learning_rate=0.05, max_depth=3)
2	1031	0.12559	31/12/2017	13:50	ElasticNet: 0.11921 GBoosting: 0.07588 XGBoost: 0.02117	Eliminación características >50% NA y no correladas, imputación de NA en train, logaritmo etiquetas, dummies	ElasticNet(alpha=[0.0001..10], l1ratio=[0.01..0.99]), GradientBoosting (estimadores=3000, learning_rate=0.05, max_depth=3) y XGBoost (estimadores=3000, learning_rate=0.05, max_depth=3)
3	1031	0.13545	31/12/2017	13:59	XGBoost: 0.02117	Eliminación características >50% NA y no correladas, imputación de NA en train, logaritmo etiquetas, dummies	XGBoost (estimadores=3000, learning_rate=0.05, max_depth=3)
4	1016	0.13044	02/01/2018	12:00	ElasticNet: 0.12619 GBoosting: 0.07604 XGBoost: 0.02307	Filtrado características con muchos NA y con información duplicada y no correladas, imputación de NA en train y test, logaritmo etiquetas, dummies	ElasticNet(alpha=[0.0001..10], l1ratio=[0.01..0.99]), GradientBoosting (estimadores=3000, learning_rate=0.05, max_depth=3) y XGBoost (estimadores=3000, learning_rate=0.05, max_depth=3)
5	1023	0.12832	02/01/2018	15:23	ElasticNet: 0.10232 GBoosting: 0.06869 XGBoost: 0.01971	Eliminación características >50% NA y no correladas, eliminación outliers, imputación de NA en train y test (utilizando mediana y moda), logaritmo etiquetas, dummies	ElasticNet(alpha=[0.0001..10], l1ratio=[0.01..0.99]), GradientBoosting (estimadores=3000, learning_rate=0.05, max_depth=3) y XGBoost (estimadores=3000, learning_rate=0.05, max_depth=3)
6	1024	0.12690	02/01/2018	16:39	ElasticNet: 0.09642 GBoosting: 0.06507 XGBoost: 0.01717	Eliminación outliers, imputación de NA en train y test para casi todas las variables (utilizando mediana y moda), logaritmo etiquetas, dummies	ElasticNet(alpha=[0.0001..10], l1ratio=[0.01..0.99]), GradientBoosting (estimadores=3000, learning_rate=0.05, max_depth=3) y XGBoost (estimadores=3000, learning_rate=0.05, max_depth=3)
7	1039	0.12720	03/01/2018	15:05	ElasticNet: 0.09510 GBoosting: 0.10801 XGBoost: 0.06490	Eliminación outliers, imputación de NA en train y test para casi todas las variables (utilizando mediana y moda), logaritmo etiquetas, dummies, transformación box-cox a todas las variables	ElasticNet(alpha=[0.0001..10], l1ratio=[0.01..0.99]), GradientBoosting (estimadores=3000, learning_rate=0.05, max_depth=3) y XGBoost (estimadores=3000, learning_rate=0.05, max_depth=3)
8	1041	0.13346	03/01/2018	16:14	ElasticNet: 0.09705	Eliminación outliers, imputación de NA en train y test para casi todas las variables (utilizando mediana y moda), logaritmo etiquetas, dummies, transformación box-cox solo a variables muy sesgadas	ElasticNet(alpha=[0.0001..10], l1ratio=[0.01..0.99])
9	492	0.11789	03/01/2018	18:38	0.1088	Eliminación outliers, imputación de NA en train y test para casi todas las variables (utilizando mediana y moda), logaritmo etiquetas, dummies, transformación box-cox solo a variables muy sesgadas, label encoding a algunas variables categóricas	Stacked Model con ElasticNet (alpha=0.0005, l1ratio=0.9), GradientBoosting (estimadores=3000, learning_rate=0.05, max_depth=4), Lasso (alpha=0.0005), y KernelRidge (alpha=0.6, grado=2, coef0=2.5)
10	502	0.11906	04/01/2018	9:33	0.1093	Eliminación outliers, imputación de NA en train y test para casi todas las variables (utilizando mediana y moda), logaritmo etiquetas, dummies, transformación box-cox solo a variables muy sesgadas, label encoding a algunas variables categóricas	Stacked Model con ElasticNet (alpha=0.0005, l1ratio=0.9), XGBoost (estimadores=2200, learning_rate=0.05, max_depth=3), Lasso (alpha=0.0005), y KernelRidge (alpha=0.6, grado=2, coef0=2.5)
11	450	0.11712	04/01/2018	10:11	0.1085	Eliminación outliers, imputación de NA en train y test para casi todas las variables (utilizando mediana y moda), logaritmo etiquetas, dummies, transformación box-cox solo a variables muy sesgadas, label encoding a algunas variables categóricas	Stacked Model con ElasticNet (alpha=0.0005, l1ratio=0.9), GradientBoosting (estimadores=3000, learning_rate=0.05, max_depth=4), XGBoost (estimadores=2200, learning_rate=0.05, max_depth=3), Lasso (alpha=0.0005), y KernelRidge (alpha=0.6, grado=2, coef0=2.5)

Tabla 2.1: Resultados descriptivos de cada una de las subidas a Kaggle (subidas 1-11).

Subida	Posición	Score	Fecha	Hora	Train RMSLE	Preprocesado	Algoritmos utilizados y parámetros
12	453	0.11722	04/01/2018	12:10	0.1078	Eliminación outliers, imputación de NA en train y test para casi todas las variables (utilizando mediana y moda), logaritmo etiquetas, dummies, transformación box-cox solo a variables muy sesgadas, label encoding a algunas variables categóricas, ranking de valores en variables categóricas	Stacked Model con ElasticNet (alpha=0.0005, l1ratio=0.9), GradientBoosting (estimadores=3000, learning_rate=0.05, max_depth=4), XGBoost (estimadores=2200, learning_rate=0.05, max_depth=3), Lasso (alpha=0.0005), y KernelRidge (alpha=0.6, grado=2, coef0=2.5)
13	447	0.11692	04/01/2018	15:11	0.1075	Eliminación outliers, imputación de NA en train y test para casi todas las variables (utilizando mediana y moda), logaritmo etiquetas, dummies, transformación box-cox solo a variables muy sesgadas, label encoding a algunas variables categóricas, ranking de valores en variables categóricas, simplificado del ranking y combinación de variables	Stacked Model con ElasticNet (alpha=0.0005, l1ratio=0.9), GradientBoosting (estimadores=3000, learning_rate=0.05, max_depth=4), XGBoost (estimadores=2200, learning_rate=0.05, max_depth=3), Lasso (alpha=0.0005), y KernelRidge (alpha=0.6, grado=2, coef0=2.5)
14	447	0.11793	04/01/2018	16:04	0.1073	Eliminación outliers, imputación de NA en train y test para casi todas las variables (utilizando mediana y moda), logaritmo etiquetas, dummies, transformación box-cox solo a variables muy sesgadas, ranking de valores en variables categóricas, simplificado del ranking y combinación de variables, label encoding a nuevas variables categóricas, añadiendo las simplificadas	Stacked Model con ElasticNet (alpha=0.0005, l1ratio=0.9), GradientBoosting (estimadores=3000, learning_rate=0.05, max_depth=4), XGBoost (estimadores=2200, learning_rate=0.05, max_depth=3), Lasso (alpha=0.0005), y KernelRidge (alpha=0.6, grado=2, coef0=2.5)
15	451	0.11765	04/01/2018	17:25	0.1069	Eliminación outliers, imputación de NA en train y test para casi todas las variables (utilizando mediana y moda), logaritmo etiquetas, dummies, transformación box-cox solo a variables muy sesgadas, ranking de valores en variables categóricas, simplificado del ranking y combinación de variables, label encoding a nuevas variables categóricas, añadiendo las simplificadas, transformaciones exponenciales de variables más correladas con la variable respuesta	Stacked Model con ElasticNet (alpha=0.0005, l1ratio=0.9), GradientBoosting (estimadores=3000, learning_rate=0.05, max_depth=4), XGBoost (estimadores=2200, learning_rate=0.05, max_depth=3), Lasso (alpha=0.0005), y KernelRidge (alpha=0.6, grado=2, coef0=2.5)
16	452	0.11774	04/01/2018	17:46	0.1069	Eliminación outliers, imputación de NA en train y test para casi todas las variables (utilizando mediana y moda), logaritmo etiquetas, dummies, transformación box-cox solo a variables muy sesgadas, ranking de valores en variables categóricas, simplificado del ranking y combinación de variables, label encoding a anteriores variables categóricas, añadiendo las simplificadas, transformaciones exponenciales de variables más correladas con la variable respuesta	Stacked Model con ElasticNet (alpha=0.0005, l1ratio=0.9), GradientBoosting (estimadores=3000, learning_rate=0.05, max_depth=4), XGBoost (estimadores=2200, learning_rate=0.05, max_depth=3), Lasso (alpha=0.0005), y KernelRidge (alpha=0.6, grado=2, coef0=2.5)

Tabla 2.2: Resultados descriptivos de cada una de las subidas a Kaggle (subidas 12-16).

Subida	Posición	Score	Fecha	Hora	Train RMSLE	Preprocesado	Algoritmos utilizados y parámetros
16	452	0.11774	04/01/2018	17:46	0.1069	Eliminación outliers, imputación de NA en train y test para casi todas las variables (utilizando mediana y moda), logaritmo etiquetas, dummies, transformación box-cox solo a variables muy sesgadas, ranking de valores en variables categóricas, simplificado del ranking y combinación de variables, label encoding a anteriores variables categóricas, añadiendo las simplificadas, transformaciones exponenciales de variables más correladas con la variable respuesta	Stacked Model con ElasticNet (alpha=0.0005, l1ratio=0.9), GradientBoosting (estimadores=3000, learning_rate=0.05, max_depth=4), XGBoost (estimadores=2200, learning_rate=0.05, max_depth=3), Lasso (alpha=0.0005), y KernelRidge (alpha=0.6, grado=2, coef0=2.5)
17	454	0.11830	04/01/2018	18:53	0.1070	Eliminación outliers, imputación de NA en train y test para casi todas las variables (utilizando mediana y moda), logaritmo etiquetas, dummies, transformación box-cox solo a variables muy sesgadas, label encoding a anteriores variables categóricas, ranking de valores en variables categóricas, simplificado del ranking y combinación de variables	Stacked Model con ElasticNet (alpha=0.0002, l1ratio=0.9), GradientBoosting (estimadores=3000, learning_rate=0.02, max_depth=3), XGBoost (estimadores=2200, learning_rate=0.02, max_depth=3), Lasso (alpha=0.0005), y KernelRidge (alpha=0.6, grado=2, coef0=2.5)
18	456	0.11097	04/01/2018	19:25	0.1066	Eliminación outliers, imputación de NA en train y test para casi todas las variables (utilizando mediana y moda), logaritmo etiquetas, dummies, transformación box-cox solo a variables muy sesgadas, label encoding a anteriores variables categóricas, ranking de valores en variables categóricas, simplificado del ranking y combinación de variables	Stacked Model con ElasticNet (alpha=0.0002, l1ratio=0.9), GradientBoosting (estimadores=3000, learning_rate=0.02, max_depth=3) y Lasso (alpha=0.0005)

Tabla 2.3: Resultados descriptivos de cada una de las subidas a Kaggle (subidas 17-19).

3. Explicación de las subidas

En esta sección se incluirá un subapartado por cada subida, donde se indicará qué ha cambiado con respecto a la subida anterior, qué ha propiciado incluir estos cambios y qué resultados han reflejado en el problema para valorar si se ha mejorado el modelo o no.

3.1. Subida 1

Para la primera subida se ha utilizado como base el kernel de Sergei Neviadomski [1], que con un modelo sencillo es capaz de dejar la puntuación rondando la posición 1000 de la competición. El preprocesado que se realiza es esencialmente el mismo que hay en el script que se nos dio por defecto, ya que el principal propósito de realizar esta subida es tomarla como punto de partida y tratar de mejorar a partir de ahí.

En primer lugar se eliminan aquellas variables que tienen aproximadamente el 50 % o más de los valores perdidos, y además se eliminan otras variables que el autor del kernel entiende no correladas con la variable respuesta, es decir, el precio de venta de la vivienda. Aquí se puede ver cómo se eliminan las variables usando las funcionalidades de Pandas y cuáles son esas variables.

```
features.drop(['Utilities', 'RoofMatl', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'Heating', 'LowQualFinSF',
               'BsmtFullBath', 'BsmtHalfBath', 'Functional', 'GarageYrBlt', 'GarageArea', 'GarageCond', 'WoodDeckSF',
               'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC', 'Fence', 'MiscFeature', 'MiscVal'],
              axis=1, inplace=True)
```

Además de esto, con el resto de variables que sí tienen NA entre los datos de train se hace una imputación de dichos valores, por ejemplo rellenando con la moda aquellas variables categóricas, con la media aquellas variables numéricas, y con 0 las variables numéricas que parecen denotar la ausencia de dicho valor. Incluso se categorizan aquellas variables numéricas que pueden actuar como tal. A continuación se incluye un ejemplo de imputación de valores perdidos.

```
# TotalBsmtSF  NA in pred. I suppose NA means 0
features['TotalBsmtSF'] = features['TotalBsmtSF'].fillna(0)

# Electrical NA in pred. filling with most popular values
features['Electrical'] = features['Electrical'].fillna(features['Electrical'].mode()[0])
```

Sobre las etiquetas (en este caso al no tratar un problema de clasificación es más adecuado llamarlas *variable respuesta*) se realiza una transformación logarítmica, con el fin de lograr que los valores estén menos sesgados y sigan una distribución más próxima a una normal.

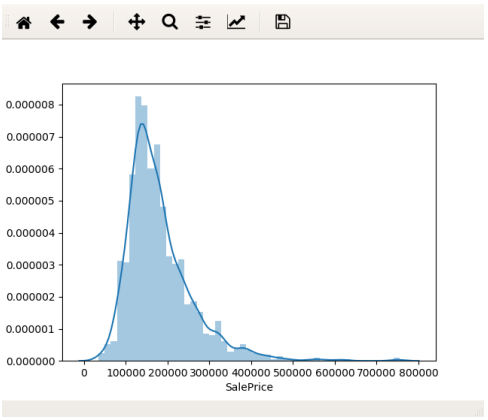

Figura 3.1: Valores sesgados de SalePrice.

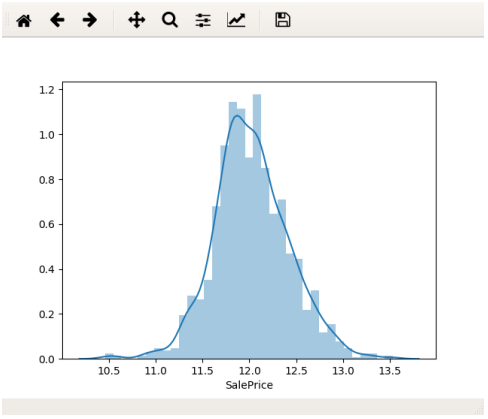

Figura 3.2: Valores no sesgados de SalePrice tras logaritmo.

Por último, como parte del preprocesado se han dividido las variables categóricas en "dummy variables", que son variables que corresponden a solo uno de los valores categóricos y que indican su presencia o no mediante un valor 0 ó 1. Aunque en este kernel se preprocesan dos variables con un dummies especial, la forma general de hacerlo es la

siguiente.

```
# Getting Dummies from all other categorical vars
for col in features.dtypes[features.dtypes == 'object'].index:
    for_dummy = features.pop(col)
    features = pd.concat([features, pd.get_dummies(for_dummy, prefix=col)], axis=1)
```

Acabado el preprocesado, en el kernel de Sergei Neviadomski se utilizaban dos algoritmos para predecir los datos: ElasticNet y GradientBoosting. Sobre cada uno de ellos se realiza una cross-validation para obtener un error medio más representativo que no esté influenciado por seleccionar unos datos en vez de otros para el conjunto de train. Para este caso se obtiene como valor de la variable respuesta el valor medio entre los que hayan encontrado cada uno de los algoritmos previamente indicados. Aquí se puede ver la definición de los modelos y los parámetros de cada uno en detalle.

```
# Elastic Net
ENSTest = linear_model.ElasticNetCV(alphas=[0.0001, 0.0005, 0.001, 0.01, 0.1, 1, 10],
            l1_ratio=[.01, .1, .5, .9, .99], max_iter=5000).fit(x_train_st, y_train_st)
```

```
# Gradient Boosting
GBest = ensemble.GradientBoostingRegressor(n_estimators=3000, learning_rate=0.05,
                                           max_depth=3, max_features='sqrt',
                                           min_samples_leaf=15, min_samples_split=10,
                                           loss='huber').fit(x_train, y_train)
```

3.2. Subida 2

En la segunda subida no se ha cambiado el preprocesado, por lo que sigue teniendo los mismos pasos que en el apartado anterior (como se podía apreciar en la tabla 2.1). Lo único que ha cambiado entre la subida anterior y esta es que se ha añadido un nuevo algoritmo de regresión al modelo, **XGBoost**. Elijo añadirlo porque es un algoritmo conocido que obtiene buenos resultados, y los parámetros con los que lo inicializo son similares a los del otro regresor con boosting, Gradient Boosting.

```
# XGBoost
XGBest = xgb.XGBRegressor(max_depth=3, learning_rate=0.05,
                          n_estimators=3000).fit(x_train, y_train)
```

Con este nuevo algoritmo añadido al modelo, las predicciones mejoran ligeramente, pasando de 0.12611 a 0.12559.

3.3. Subida 3

Al ver que el modelo no mejora prácticamente nada decido probar XGBoost por separado, a pesar de que el error cuadrático medio que se obtiene en el conjunto de entrenamiento

es muy bajo, lo que da lugar a pensar que con casi toda seguridad habrá sobreajuste.

En efecto, se produce un sobreajuste y en el conjunto de test el error se dispara hasta 0.13545. La idea de dejar XGBoost solo no da un buen resultado.

3.4. Subida 4

Para la cuarta subida decido hacer yo mi propio filtrado de características y no dejar las que aparecían en el primer kernel. Para ello me baso en la información sobre lo que significan las variables y sus respectivos valores en la página de la competición de Kaggle, además de en algunas ayudas obtenidas del kernel explicativo de Pedro Marcelino [2]. El filtrado de variables se ha subdividido en varias partes, para diferenciar el motivo de filtrado de cada variable, como se puede apreciar. Previamente además se ha realizado una búsqueda sobre los datos para ver qué variables contienen más valores perdidos, y esos son los que se incluyen en el primer filtrado.

A continuación se incluyen el código del filtrado de atributos y la tabla con los atributos con mayor cantidad de valores perdidos, que han indicado cuáles deben pertenecer al grupo de filtrados.

	MissingValues	Percent
PoolQC	2909	0.996574
MiscFeature	2814	0.964029
Alley	2721	0.932169
Fence	2348	0.804385
FireplaceQu	1420	0.486468
LotFrontage	486	0.166495
GarageCond	159	0.054471
GarageQual	159	0.054471
GarageYrBlt	159	0.054471
GarageFinish	159	0.054471

```
# Checking for missing data, showing every variable with at least one missing value in train set
total_missing_data = features.isnull().sum().sort_values(ascending=False)
missing_data_percent = (features.isnull().sum()/features.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total_missing_data, missing_data_percent], axis=1, keys=['Total', 'Percent'])
print(missing_data[missing_data['Percent']> 0])

# I get rid of the features that have a lot of missing data
features.drop(['PoolQC', 'MiscFeature', 'Alley', 'Fence', 'LotFrontage'], axis=1, inplace=True)

# Now I drop those features with duplicated information
features.drop(['GarageType', 'GarageYrBlt', 'GarageFinish', 'GarageArea', 'GarageQual', 'GarageCond',
               '3SsnPorch', 'ScreenPorch', 'BsmtQual', 'BsmtCond', 'Heating', 'LandSlope', 'Exterior1st',
               'Exterior2nd', 'KitchenAbvGr', 'BedroomAbvGr', 'Fireplaces'], axis=1, inplace=True)

# Now the same for those features that seem non-related to SalePrice, or do not give much information about it
features.drop(['BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtFullBath',
               'BsmtHalfBath', 'BsmtUnfSF', 'Utilities', 'Street', 'MasVnrType', 'MasVnrArea'], axis=1, inplace=True)
```

Cabe destacar que además en esta ocasión se están mirando también qué datos tienen valores perdidos tanto en train como en test, y por eso se tratan de corregir en el test también. El resultado obtenido no mejora el filtrado de variables que se realizaba en las anteriores subidas, quedándose en 0.13044.

3.5. Subida 5

Para la quinta subida vuelvo a utilizar el filtrado de características inicial. Además, en esta ocasión elimino outliers por primera vez. En la primera figura se pueden considerar outliers los dos datos que tienen un valor alto de área perteneciente a la vivienda, y sin embargo su precio de venta es reducido. Estos datos se pueden filtrar con la simple línea de código que se muestra a continuación, y deja el dataset más uniforme, como se aprecia en la segunda figura.

```
# Deleting outliers
train = train.drop(train[(train['GrLivArea']>4000) & (train['SalePrice']<300000)].index)
```

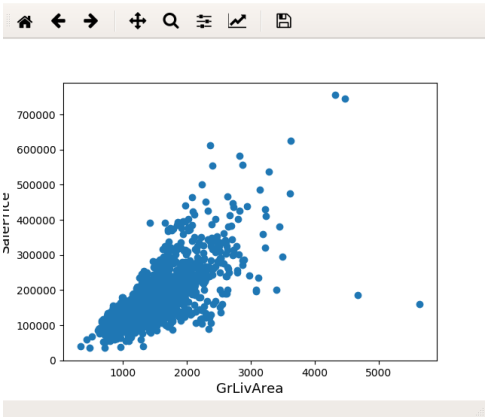

Figura 3.3: Dataset con outliers.

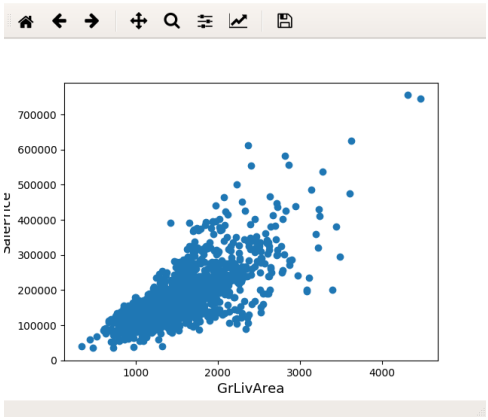

Figura 3.4: Dataset sin outliers.

Además de eliminar los outliers, se ha añadido la mediana a la hora de determinar los valores perdidos de variables numéricas, en lugar de la media. Además, se ha hecho de forma que determine la mediana del atributo en función del resto de valores de ese atributo para los ejemplos que pertenecen a su mismo grupo en OverallQual, que es una de las variables más correladas con la variable respuesta.

```
# LotFrontage NA filling with median according to its OverallQual value
median = features.groupby('OverallQual')['LotFrontage'].transform('median')
features['LotFrontage'] = features['LotFrontage'].fillna(median)
```

El objetivo de hacer esto es que siendo OverallQual una de las variables más determinantes a la hora de predecir el precio de la vivienda, si siguiesen dichas variables numéricas

con datos perdidos la misma distribución se podrían obtener buenos resultados. Sin embargo, por la poca cantidad de datos perdidos en dichas variables o por no realizar el preprocesado de forma adecuada, no mejora como estaba previsto, quedando en un error cuadrático medio de 0.12832.

Referencias

[1] Sergei Neviadomski, How to get Top 25 % with simple model (sklearn), Kaggle, `https://www.kaggle.com/neviadomski/how-to-get-to-top-25-with-simple-model-sklearn/notebook`

[2] Pedro Marcelino, Comprehensive data exploration with Python, Kaggle, `https://www.kaggle.com/pmarcelino/comprehensive-data-exploration-with-python/notebook`