

Predicción del precio de viviendas en Barcelona

Aprendizaje Automático (GEI)

Cuatrimestre de Otoño, curso 2022/23



Javier Abella Nieto
Victor Teixidó López

Índice

| | |
|---|-----------|
| 1. Introducción | 2 |
| 2. Estudios previos | 3 |
| 3. Exploración de los datos | 4 |
| 4. Protocolo de remuestreo | 7 |
| 5. Métodos lineales | 8 |
| 5.1 Lasso regression | 8 |
| 5.2 Ridge regression | 10 |
| 5.3 Polynomial SVM | 11 |
| 6. Modelos no lineales | 14 |
| 6.1 Random Forest | 14 |
| 6.2 Gradient Boosting | 15 |
| 6.3 Stacked regressor | 17 |
| 7. Mejor modelo | 19 |
| 8. Análisis de interpretabilidad | 20 |
| 9. Conclusión | 22 |
| 10. Webgrafía | 23 |

1. Introducción

En este trabajo vamos a realizar un análisis de distintos modelos de aprendizaje automático con el objetivo de predecir el precio de distintas viviendas de Barcelona en base a sus características como el número de habitaciones, la planta, los metros cuadrados... El primer paso ha sido realizar una exploración de los datos a los cuales les hemos aplicado las técnicas de preprocesado que nos han parecido más adecuadas, como la eliminación de variables no suficientemente informativas o el agrupamiento de clases muy específicas dentro de una misma variable.

Debido a que el objetivo es predecir el precio final de una vivienda, hemos seleccionado una serie de modelos de regresión tanto lineales como no lineales. Los 3 modelos lineales que hemos utilizado son una regresión lasso, una regresión ridge y una SVM con kernel polinómico, y los modelos no lineales son un random forest, un gradient boosting y un stacked regressor. Una vez entrenados los modelos con los mejores hiperparámetros encontrados, compararemos los resultados obtenidos por cada uno de ellos y determinaremos cuál es el modelo que mejor se ajusta a los datos y proporciona predicciones más precisas. Además, analizaremos también el impacto de las características de las viviendas en el precio final de las mismas para cada uno de los modelos.

Al final, hemos obtenido que el modelo con mejor rendimiento ha sido la regresión lasso, con una puntuación R^2 en la fase test igual a 0.659.

El conjunto de datos con el que vamos a trabajar lo hemos obtenido del siguiente dataset de Kaggle, [Barcelona_Idealista_HousingPrices](#). Tal y como se explica, se trata de un conjunto de viviendas seleccionadas aleatoriamente de 10 distritos de Barcelona utilizando web scraping. Adicionalmente, hemos guardado el conjunto de datos en [este](#) repositorio de GitHub público para facilitar la lectura de los datos desde el notebook.

2. Estudios previos

Tras realizar una búsqueda de estudios previos sobre este mismo conjunto de datos no hemos encontrado ninguno. Además, en kaggle únicamente el autor del dataset ha realizado un notebook, aún así, en este notebook solo se realiza una mínima exploración de los datos pero no aplica ningún método de predicción de valores.

Hay muchos estudios sobre conjuntos de datos relacionados con los precios de las viviendas de Barcelona y el porqué de estos precios, pero que utilicen el conjunto de datos que nosotros hemos elegido no hay ninguno.

3. Exploración de los datos

Para la exploración del conjunto de datos hemos decidido hacer tanto una exploración individualizada como viendo también la correlación entre las distintas variables. En base a lo que hemos ido viendo hemos tomado las distintas decisiones para el procesamiento de los datos. Nuestro conjunto de datos cuenta con un total de 3265 individuos y un total de 11 atributos, uno de estos será el objetivo a predecir. Entre estas variables tenemos tanto numéricas como categóricas.

Vamos a describir brevemente las distintas columnas:

- **city:** Nombre de la ciudad a la que pertenece. Todas son Barcelona.
- **district:** El distrito de la ciudad al que pertenece.
- **neighborhood:** El barrio de la ciudad al que pertenece.
- **condition:** La condición en la que se encuentra. Nuevo, buenas condiciones o a reformar.
- **type:** De qué tipo es la vivienda. Penthouse, piso, dúplex, casa...
- **rooms:** El número de habitaciones.
- **area_m2:** Los metros cuadrados.
- **lift:** Si tiene o no ascensor.
- **views:** Sí dispone de vistas o no.
- **floor:** El número de planta en el que se encuentra.
- **prices:** El precio que tiene la vivienda. Esta será la variable a predecir.

Respecto a posibles outliers en variables numéricas vemos lo siguiente en nuestro conjunto de datos. Parecen haber bastantes valores que se escapan de la norma, podríamos imputarlos pero no lo haremos. El motivo de esto es porque consideramos que viviendas con altos metros cuadrados significan precios más altos y esta es una relación que nos interesa que nuestros modelos aprendan. Si imputáramos estos ejemplos de viviendas con metros cuadrados muy elevados, perderíamos esta información.

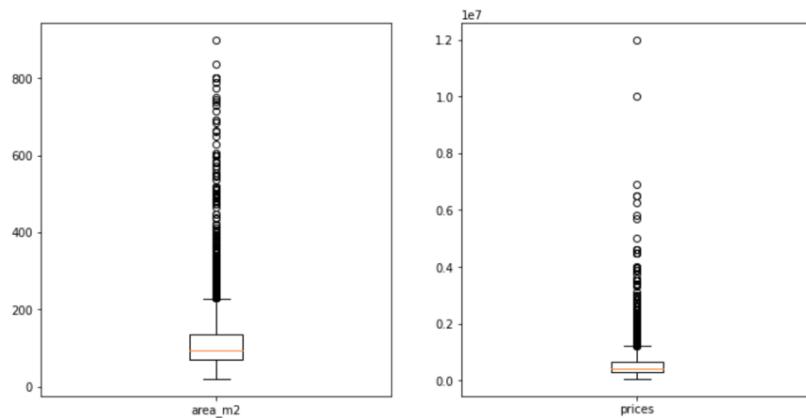


Figura 3.1: Boxplot variables numéricas

Vamos a ver ahora procesamientos individuales que hemos hecho a algunos atributos debido a patrones que hemos visto.

Tal y como se ve en la figura 3.1, la variable *rooms* tiene pocos individuos una vez pasamos el umbral de las 6 habitaciones. Tener por ejemplo un único ejemplo de vivienda que tenga 13 habitaciones no es muy significativo. Es por esto que hemos decidido agrupar todas las viviendas que tengan 6 o más habitaciones en única clase tal que ≥ 6 . Entendemos además, por nuestro conocimiento respecto a la inmobiliaria, que todas las viviendas de más de 6 habitaciones suelen ser más caras también.

```
3    1174
2     773
4     679
5     261
1     232
6      81
7      36
8      14
9       6
10      6
11      2
13      1
Name: rooms, dtype: int64
```

Figura 3.2: Conteo de viviendas con su número de habitaciones

Para la variable *floor*, hemos seguido un procesamiento bastante similar al que hemos visto hace un momento. Podemos ver como hay muchísimos números de pisos distintos pero que a partir del séptimo piso, la cantidad de individuos por piso empieza a ser no muy significativa. Por esto mismo, agrupamos todos los pisos por encima de este en una misma clase a la que hemos llamado *floor* ≥ 7 .

```
floor 1      681
floor 2      588
floor 3      452
floor 4      385
floor 5      259
ground floor 236
floor 6      160
mezzanine    146
Multiple     144
floor 7      86
floor 8      66
floor 9      27
floor 10     16
floor 14     7
floor 11     3
floor 12     3
floor 25     2
floor 16     2
floor 19     1
floor 21     1
Name: floor, dtype: int64
```

Figura 3.3: Conteo de viviendas para la variable type

Para el atributo *type* hemos optado por eliminar las clases con menos individuos. Debido al poco impacto en la suma total de los datos, de cara al preprocesamiento eliminaremos las instancias de viviendas que sean *terraced*, *semi-detached* o *rustic*.

```
Flat          2751
Penthouse     262
Duplex        108
Detached      55
House         54
Terraced      22
Semi-detached 12
Rustic         1
Name: type, dtype: int64
```

Figura 3.4: Conteo de viviendas para la variable type

De cara al preprocesamiento, hemos pasado a numéricas todas las variables categóricas para que puedan ser utilizadas por los modelos que vamos a entrenar. Las categorías que tengan relación de precedencia o niveles entre las distintas clases se transformarán en numéricas con números que definen dichas relaciones, tales que 0, 1, 2... Las variables categóricas con clases sin importancias, se les aplicará one-hot encoding para su transformación. Adicionalmente, hemos eliminado los valores nulos de nuestro conjuntos de datos y renombrado algunas variables para más claridad en las salidas.

Por último, hemos creado una copia del conjunto de datos y lo hemos estandarizado, esto es debido a que hay modelos como la regresión lasso o la regresión ridge, que obtienen mejores resultados si los datos están estandarizados.

Adicionalmente, utilizando pca hemos visto una representación en dos dimensiones de los diferentes rangos de precios, dicha representación la vemos en la figura 3.5. Podemos observar que hay una gran nube de una rosa pálida que define las viviendas con un precio alrededor de los 150000 euros. Algo más ladeado, vemos una línea que pasa de viviendas de precios bajos a individuos que representan viviendas de precios más elevados. Podemos observar como las diferencias entre las clases no son las más claras del mundo y que prácticamente solo existe una clase. Esto nos da pistas ya de que nuestros modelos no podrán rendir de la forma más óptima posible, posiblemente por falta de información y por overfittings que se vayan a producir.

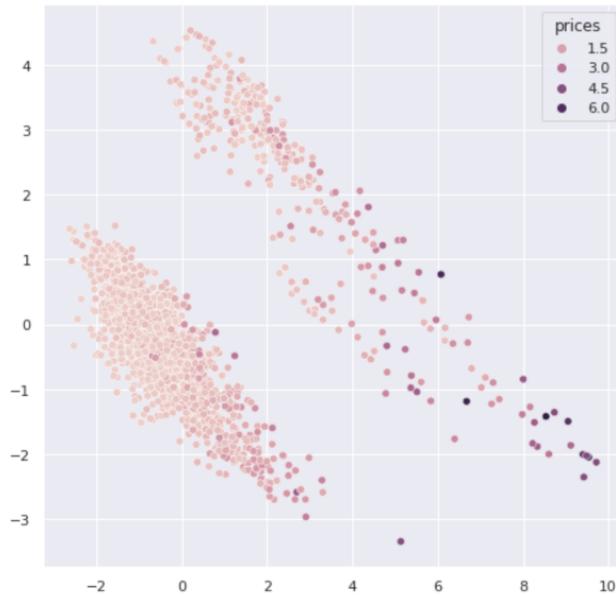


Figura 3.5: Representación con pca de los individuos del conjunto de datos

4. Protocolo de remuestreo

Para evaluar la precisión y la fiabilidad de los modelos que vamos a utilizar durante la práctica, vamos a seguir un seguido de protocolos. El primero de estos será dividir el conjunto original de datos en un conjunto de entrenamiento y un conjunto de test de forma aleatoria, en un 70% y un 30% respectivamente. La partición de entrenamiento la utilizaremos, como indica su nombre, para entrenar los distintos modelos y la partición test nos será útil para comprobar realmente que tan bien funciona un modelo con datos que no ha utilizado durante el aprendizaje. De esta forma podremos evaluar la capacidad de generalización de nuestro modelo.

Las métricas que utilizaremos para poder comparar el rendimiento de los diferentes modelos sobre los conjuntos de entrenamiento y test son dos. Por un lado, el *test_score* es un valor que nos indica que tan bien se comporta un modelo con datos que no han sido utilizados durante el entrenamiento, un valor más alto indicará una mayor robustez, este valor será el definitivo a la hora de decidir un modelo u otro. Por otro lado, el *cross_val_score* es una medida del rendimiento del modelo en los datos de entrenamiento, un valor alto indica que el modelo se ajusta bien a los datos y se amolda a las posibles relaciones que existan. Un *cross_val_score* muy elevado no siempre es indicador de un buen rendimiento por parte del modelo, ya que puede ser que exista cierta tendencia al overfitting.

Esta información que vamos a ir obteniendo de cada modelo la iremos almacenando en un dataframe para poder ir comparando a posteriori.

```
from sklearn.model_selection import cross_val_score
cv = 10

def save_results(clf, nclf, X, y, df):
    df.loc[nclf, 'test_score'] = clf.score(X, y)
    df.loc[nclf, 'cross_val_score'] = clf.best_score_
    return df

results_df = pd.DataFrame()
```

Figura 4.1: Función utilizada para el guardado de los resultados de rendimiento de los modelos

Como añadido adicional, comentar que ajustaremos el número de pliegues a 10. Hemos optado por este valor ya que computacionalmente es el máximo con el que hemos podido obtener tiempos de ejecución razonables.

5. Métodos lineales

5.1 Lasso regression

El primer modelo que vamos a entrenar es una regresión lasso. El primer paso será elegir los hiperparámetros que vamos a explorar para poder ajustar el modelo adecuadamente, podemos ver los hiperparámetros en la figura 5.1.1. Para el entrenamiento utilizaremos *GridSearchCV* y haremos uso de los datos estandarizados tal y como hemos explicado antes.

```
param = {  
    'alpha':[1e-4,0.001,0.01,0.1, 0.5,1,5,10,50,100,300,500],  
    'max_iter':[1000,10000,50000],  
    'selection':["cyclic", "random"]  
}  
  
lasso = Lasso()  
  
lasso_gs = GridSearchCV(lasso, param, cv = cv,n_jobs = -1);  
lasso_gs.fit(X_train_s,y_train);
```

Figura 5.1.1: Hiperparámetros explorados para la regresión Lasso

Tras entrenar el modelo miramos cuáles han sido los mejores resultados que se han obtenido durante el entrenamiento. En la siguiente tabla podemos ver la configuración de los mejores 5 modelos obtenidos, para la toma de métricas y rendimiento del modelo utilizaremos la primera de estas 5.

| params | mean_test_score | rank_test_score |
|---|-----------------|-----------------|
| {'alpha': 5, 'max_iter': 1000, 'selection': 'random'} | 0.761 | 1 |
| {'alpha': 10, 'max_iter': 1000, 'selection': 'random'} | 0.761 | 2 |
| {'alpha': 0.01, 'max_iter': 50000, 'selection': 'random'} | 0.761 | 3 |
| {'alpha': 0.0001, 'max_iter': 10000, 'selection': 'random'} | 0.761 | 4 |
| {'alpha': 0.0001, 'max_iter': 50000, 'selection': 'random'} | 0.761 | 5 |

Figura 5.1.2: Mejores 5 configuraciones en base a los resultados obtenidos durante el entrenamiento

En la siguiente figura podemos observar gráficamente la puntuación R² obtenida sobre los datos del conjunto test.

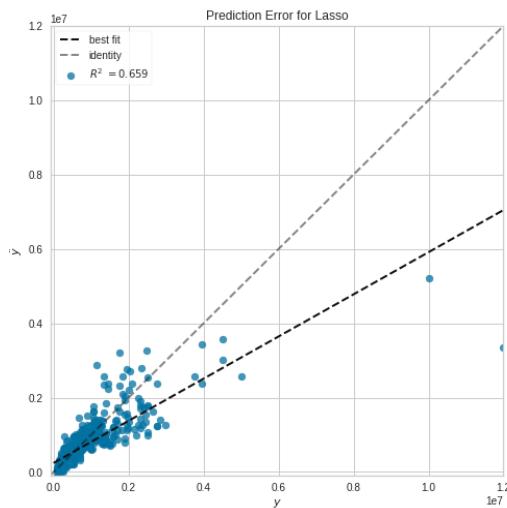


Figura 5.1.3: Resultados obtenidos sobre los datos test utilizando el modelo entrenado

El valor R^2 obtenido es 0.659 el cual no es un resultado especialmente bueno. Comparado con el valor obtenido durante el entrenamiento es bastante peor, lo que puede dar ciertos indicios de que se está produciendo algo de overfitting en nuestro modelo. Además, en nuestros datos podemos observar que la mayoría de los precios de vivienda son bajos comparándolos con los precios de algunas viviendas que son bastante más caras. Lo que hace que como vemos en el gráfico la mayoría de precios se acumulen al principio y el best fit se desvíe tanto.

Por último, miramos el nivel de importancia de los atributos para ver cuáles son los que más definen nuestro modelo. Como podemos ver en la gráfica, el atributo con más importancia es el que define la condición de una vivienda, este al parecer es muy significativo y destaca notablemente respecto a los demás atributos. Fijándonos en el resto de variables que definen una vivienda podemos ver que sus importancias son casi negligibles, aún así, por destacar algo podríamos mencionar que los 3 siguientes atributos más significativos son el número de habitaciones, la superficie y si tiene o no ascensor.

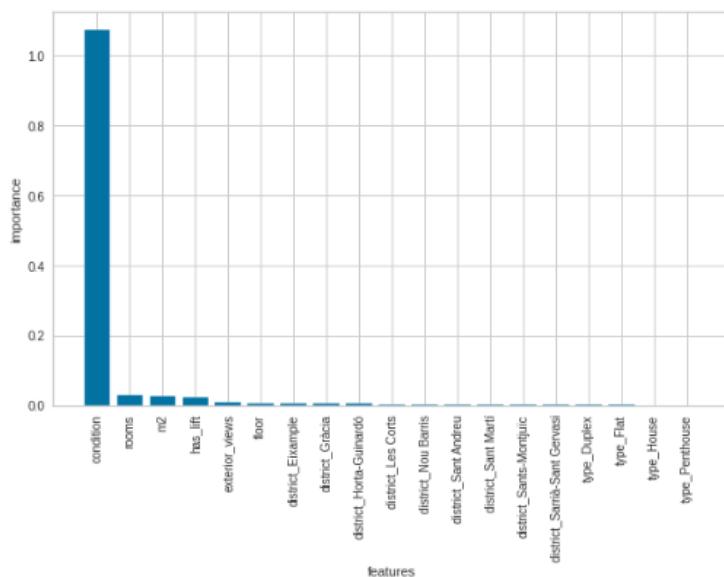


Figura 5.1.4: Representación de la importancia de los atributos a la hora de predecir el valor de una vivienda

5.2 Ridge regression

El siguiente modelo a analizar será el de Ridge Regression. Empezaremos por explorar los hiperparámetros del modelo como hicimos en el anterior modelo, utilizando *GridSearchCV* y los datos estandarizados . En la siguiente figura podemos ver los parámetros explorados.

```
param_grid = {
    'alpha': [0.01, 0.1, 1, 10, 100, 200, 500],
    'max_iter':[1000,10000,50000],
    'solver':['auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga', 'lbfgs']
}

ridge = Ridge()
ridge_gs = GridSearchCV(ridge, param_grid, cv=cv)
ridge_gs.fit(X_train_s, y_train);
```

Figura 5.2.1: Hiperparámetros explorados para la Ridge Regression

En la siguiente figura observamos las 5 mejores combinaciones de hiperparámetros y los respectivos resultados obtenidos. Podemos ver cómo al igual que en el anterior modelo, hay varias combinaciones que dan el mismo resultado, por consistencia utilizaremos el primero que sale en la tabla.

| params | mean_test_score | rank_test_score |
|---|-----------------|-----------------|
| {'alpha': 10, 'max_iter': 10000, 'solver': 'sparse_cg'} | 0.762 | 1 |
| {'alpha': 10, 'max_iter': 50000, 'solver': 'sparse_cg'} | 0.762 | 1 |
| {'alpha': 10, 'max_iter': 1000, 'solver': 'sparse_cg'} | 0.762 | 1 |
| {'alpha': 10, 'max_iter': 10000, 'solver': 'sag'} | 0.762 | 4 |
| {'alpha': 10, 'max_iter': 1000, 'solver': 'saga'} | 0.762 | 5 |

Figura 5.2.2: Hiperparámetros explorados para la Ridge Regression

Vamos a ver ahora cuál es la puntuación R^2 con el conjunto de test con este modelo.

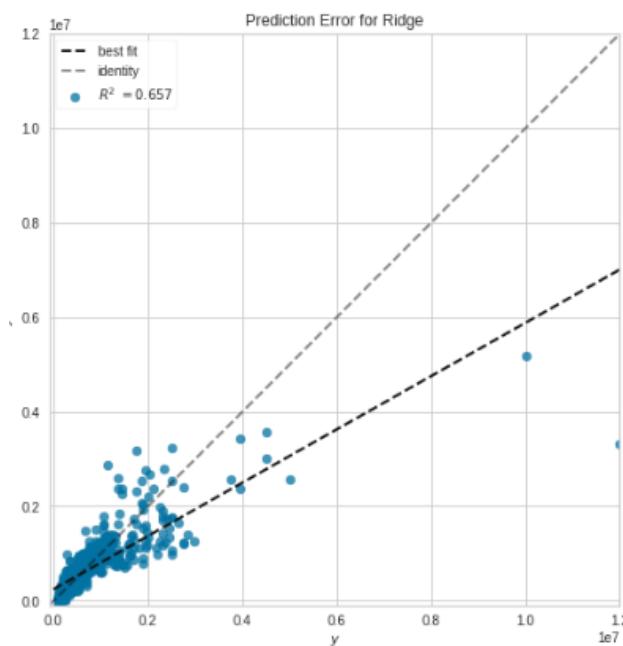


Figura 5.2.3: Resultados obtenidos sobre los datos test utilizando el modelo entrenado

La puntuación empeora respecto a la del test algo más que un 0.1, esto debido probablemente a un sobreajuste de parte del modelo a los datos de entrenamiento. Podemos ver por eso como comparando ambos resultados tenemos una pérdida de puntuación.

Por último miramos cual ha sido la importancia de los atributos del modelo en la figura 5.2.4. Como ya vimos en el anterior modelo el atributo que más importancia tiene es la condición del piso, que podría decirse que es prácticamente el único atributo que influye en la predicción. Aún así, este viene seguido por el número de habitaciones, la superficie y si tiene o no ascensor, luego vienen el resto de atributos cuyas importaciones son casi negligibles.

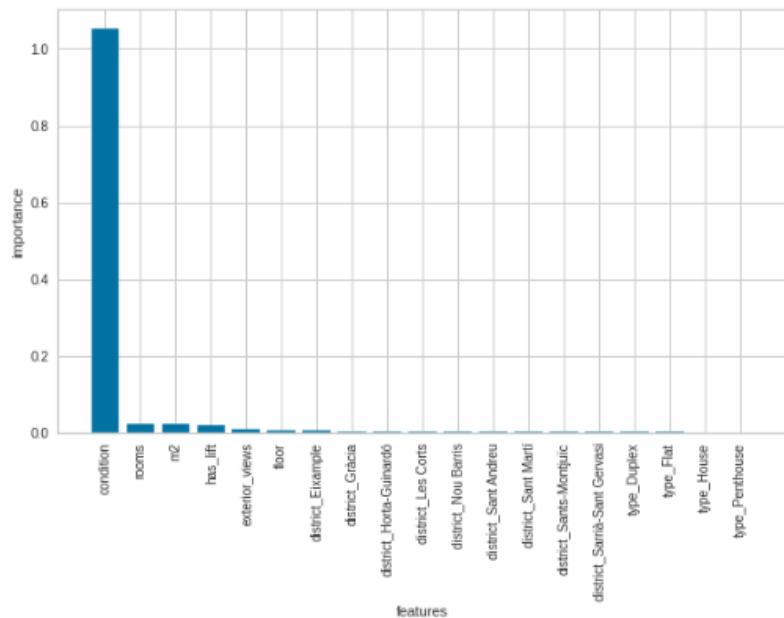


Figura 5.2.4: Representación de la importancia de los atributos a la hora de predecir el valor de una vivienda

5.3 Polynomial SVM

Tal y como mencionamos anteriormente, otro de los modelos de regresión lineal que vamos a probar es una máquina de soporte vectorial (o SVM) con un kernel polinómico. Al igual que con los otros modelos vistos hasta ahora, hemos realizado la exploración de los hiperparámetros con la ayuda de *GridSearchCV*. Los hiperparámetros que hemos explorado para este modelo son los siguientes. Además, hemos establecido el máximo número de iteraciones en 25000 y el tamaño de la caché en 2000, para garantizar que permitimos una correcta exploración de todos los parámetros definidos.

```
params = {'C': 10**np.linspace(-3,3,20),
          'gamma': 10**np.linspace(-9,3,20),
          'degree':[2,3]}

svm = SVR(kernel='poly', max_iter=25000, cache_size=2000)
svm_gs = GridSearchCV(svm, params, cv=cv, n_jobs=-1)
svm_gs.fit(X_train_s, y_train);
```

Figura 5.3.1: Mejores 5 configuraciones en base a los resultados obtenidos durante el entrenamiento

En la siguiente figura podemos observar las 5 configuraciones con las que hemos obtenido mejores resultados durante el proceso de entrenamiento. Para el resto del análisis y la toma de resultados, vamos a utilizar la mejor de estas configuraciones con la que hemos obtenido una puntuación de 0.74 durante el proceso de entrenamiento.

| params | mean_test_score | rank_test_score |
|--|-----------------|-----------------|
| {'C': 0.0379269019073225, 'degree': 3, 'gamma': 12.742749857031322} | 0.740 | 1 |
| {'C': 2.976351441631316, 'degree': 3, 'gamma': 2.976351441631313} | 0.740 | 2 |
| {'C': 233.57214690901213, 'degree': 3, 'gamma': 0.6951927961775591} | 0.740 | 3 |
| {'C': 0.07847599703514611, 'degree': 3, 'gamma': 12.742749857031322} | 0.738 | 4 |
| {'C': 483.2930238571752, 'degree': 3, 'gamma': 0.6951927961775591} | 0.738 | 5 |

Figura 5.3.2: Mejores 5 configuraciones en base a los resultados obtenidos durante el entrenamiento

Vamos a ver ahora la puntuación R^2 que hemos obtenido sobre el conjunto de datos de test utilizando este mismo modelo.

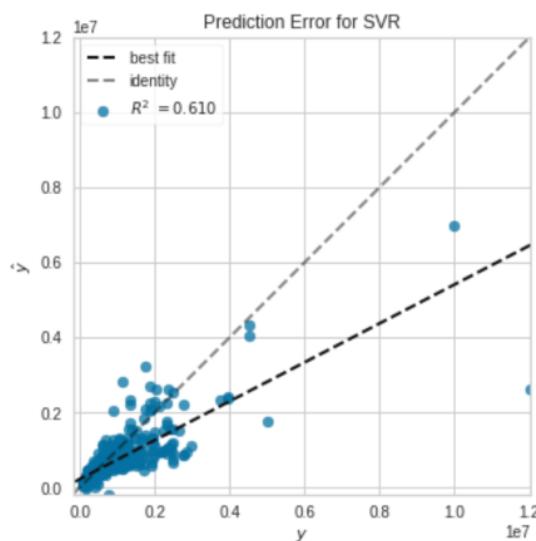


Figura 5.3.3: Resultados obtenidos sobre los datos test utilizando el modelo entrenado

Como podemos ver, hemos obtenido como valor de R^2 0.61, este valor comparado al que veíamos durante el entrenamiento es bastante peor, hemos perdido alrededor de 0.13 de acierto. Esto puede ser causa principalmente de dos aspectos, que la distribución de los individuos no sea parecida en el conjunto de entrenamiento y el conjunto test o que el modelo esté padeciendo de un pequeño sobreajuste sobre los datos que estamos utilizando a la hora de entrenar el modelo. Esta segunda opción parece ser el motivo de la pérdida de puntuación del entrenamiento a test. La solución ideal sería aumentar el número de pliegues que realizamos durante el entrenamiento del modelo, el problema de esto, es que computacionalmente no hemos sido capaces de asumir este aumento y por tanto hemos tenido que dejar el entrenamiento del modelo así.

También hemos representado la importancia de los distintos atributos de una vivienda a la hora de predecir el precio de la misma con este modelo. Como podemos ver en la siguiente figura, parece que el atributo decisivo a la hora de predecir un precio, es la condición de la propia vivienda. Aún así, también parece que las habitaciones, los metros cuadrados y si tiene ascensor, tiene algo, aunque sea mínimo, de importancia.

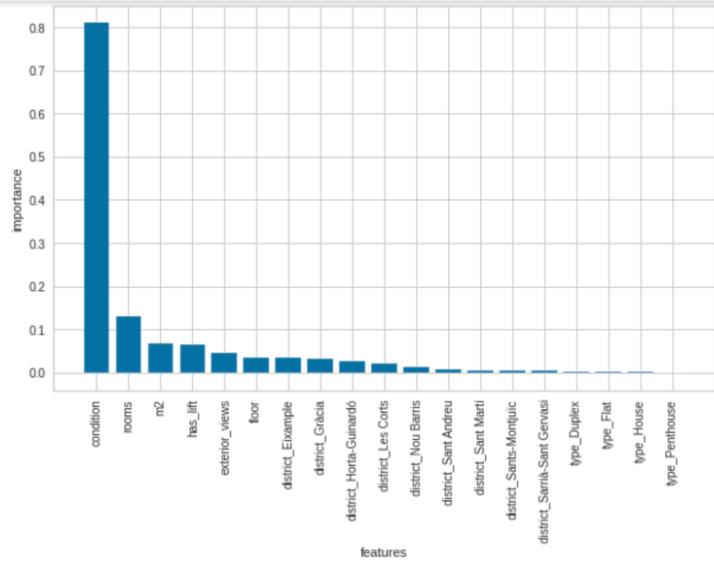


Figura 5.3.4: Representación de la importancia de los atributos a la hora de predecir el valor de una vivienda

6. Modelos no lineales

6.1 Random Forest

El primer modelo no lineal que vamos a entrenar es un modelo que utiliza random forest. Vamos a seguir el proceso habitual de explorar los distintos hiperparámetros del modelo, elegir los mejores y entrenar al modelo con estos.

```
param = {'n_estimators': [5, 10, 25, 40, 50, 75, 100, 200],
         'criterion':['mse'],
         'max_depth':[None, 1, 2, 3, 5, 8, 9,10,15],
         'min_samples_leaf':[1, 2, 3, 5, 10]}

rf = RandomForestRegressor(random_state=1)
rf_gs = GridSearchCV(rf,param, cv=cv, n_jobs=-1)
rf_gs.fit(X_train, y_train);
```

Figura 6.1.1: Hiperparámetros explorados para el Random Forest

Aquí podemos observar las mejores combinaciones de hiperparámetros para el modelo de Random Forest. Podemos ver que con este modelo ha mejorado la puntuación sobre el conjunto de entrenamiento respecto a los modelos lineales, en este caso con un valor de casi 0.8. Vamos a elegir la primera configuración como nuestro modelo a utilizar.

| | params | mean_test_score | rank_test_score |
|-----|---|-----------------|-----------------|
| 285 | {'criterion': 'mse', 'max_depth': 10, 'min_samples_leaf': 1, 'n_estimators': 75} | 0.799 | 1 |
| 286 | {'criterion': 'mse', 'max_depth': 10, 'min_samples_leaf': 1, 'n_estimators': 100} | 0.799 | 2 |
| 5 | {'criterion': 'mse', 'max_depth': None, 'min_samples_leaf': 1, 'n_estimators': 75} | 0.799 | 3 |
| 6 | {'criterion': 'mse', 'max_depth': None, 'min_samples_leaf': 1, 'n_estimators': 100} | 0.798 | 4 |
| 287 | {'criterion': 'mse', 'max_depth': 10, 'min_samples_leaf': 1, 'n_estimators': 200} | 0.798 | 5 |

Figura 6.1.2: Mejores 5 configuraciones en base a los resultados obtenidos durante el entrenamiento

Vamos a ver ahora cómo rinde el modelo utilizando los datos del conjunto de entrenamiento. En este caso, tal y como se aprecia en la figura 6.1.3, hemos conseguido una puntuación R^2 igual a 0.622 un valor peor a los que habíamos visto hasta ahora. Si lo pensamos fríamente, el haber ganado puntuación en el entrenamiento deriva en un mayor overfitting y de ahí la pérdida de acierto en el conjunto test. Tal y como íbamos razonando hasta ahora, la pequeñez de nuestro conjunto de datos y la poca diversidad de este juega en nuestra contra.

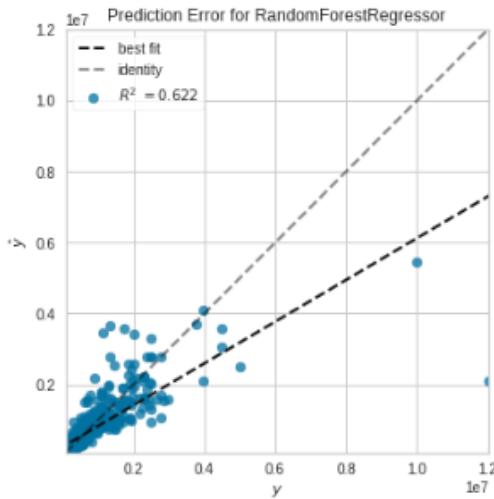


Figura 6.1.3: Resultados obtenidos sobre los datos test utilizando el modelo entrenado

En cuanto a la importancia de atributos vemos que la variable *condition* sigue siendo la más importante y en este modelo de una forma aún más acentuada. Le siguen los atributos que definen el número de habitaciones, los metros cuadrados y la existencia de un ascensor o no. El resto, al igual que con los otros modelos, tienen importancias casi inexistentes.

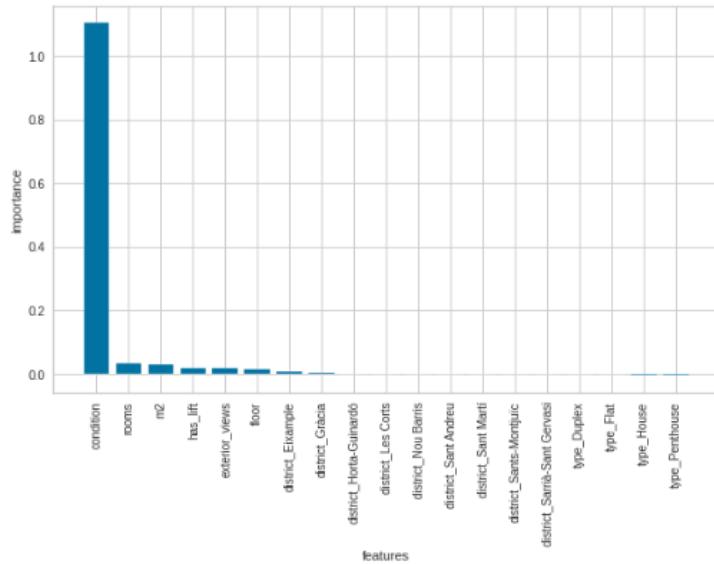


Figura 6.1.4: Representación de la importancia de los atributos a la hora de predecir el valor de una vivienda

6.2 Gradient Boosting

Tal y como hemos mencionado al principio, otro de los modelos de regresión no lineal que vamos a utilizar es un modelo que utiliza gradient boosting. Al igual que con los otros modelos vistos hasta ahora, hemos realizado la exploración de los hiperparámetros con la ayuda de *GridSearchCV*. Los hiperparámetros que hemos explorado para este modelo son los siguientes.

```
params = {'n_estimators': [10, 100, 250, 500, 1000],
          'learning_rate': [.001, .01, .1, 1],
          'max_depth': [1, 2, 5, 10],
          'subsample': [.5, .75, 1],
          'loss': ['squared_error', 'absolute_error']}}

gbr = GradientBoostingRegressor(random_state=1)
gbr_gs = GridSearchCV(gbr, params)
gbr_gs.fit(X_train, y_train)
```

Figura 6.2.1: Hiperparámetros explorados para la SVM cuadrática

En la siguiente figura podemos observar las 5 configuraciones con las que hemos obtenido mejores resultados durante el proceso de entrenamiento. Para el resto del análisis y la toma de resultados, vamos a utilizar la mejor de estas configuraciones con la que hemos obtenido una puntuación de 0.813 durante el proceso de entrenamiento, el cual es el valor más alto obtenido hasta ahora.

| params | mean_test_score | rank_test_score |
|---|-----------------|-----------------|
| {'learning_rate': 0.01, 'loss': 'absolute_error', 'max_depth': 10, 'n_estimators': 1000, 'subsample': 0.5} | 0.813 | 1 |
| {'learning_rate': 0.01, 'loss': 'squared_error', 'max_depth': 5, 'n_estimators': 500, 'subsample': 0.75} | 0.811 | 2 |
| {'learning_rate': 0.01, 'loss': 'absolute_error', 'max_depth': 10, 'n_estimators': 1000, 'subsample': 0.75} | 0.811 | 3 |
| {'learning_rate': 0.01, 'loss': 'squared_error', 'max_depth': 5, 'n_estimators': 1000, 'subsample': 0.75} | 0.810 | 4 |
| {'learning_rate': 0.01, 'loss': 'squared_error', 'max_depth': 5, 'n_estimators': 500, 'subsample': 0.5} | 0.810 | 5 |

Figura 6.2.2: Mejores 5 configuraciones en base a los resultados obtenidos durante el entrenamiento

Vamos a ver ahora la puntuación R^2 que hemos obtenido sobre el conjunto de datos de test utilizando este mismo modelo.

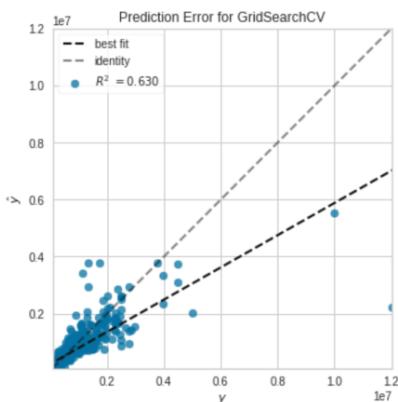


Figura 6.2.3: Resultados obtenidos sobre los datos test utilizando el modelo entrenado

Como podemos ver, hemos obtenido como valor de R^2 0.63, este valor comparado al que veíamos durante el entrenamiento es bastante peor, hemos perdido alrededor de 0.18 de acierto lo cual no es obvio. Tal y como hemos mencionado con los otros modelos, probablemente debido al sobreajuste que se está produciendo durante el entrenamiento del propio modelo. La situación ideal sería poder aumentar el número de pliegues realizados durante el entrenamiento pero debido al coste computacional que esto supone, no hemos podido hacerlo.

También hemos representado la importancia de los distintos atributos de una vivienda a la hora de predecir el precio de la misma con este modelo. Como podemos ver en la siguiente figura, parece que el atributo decisivo a la hora de predecir un precio, es la condición de la propia vivienda.

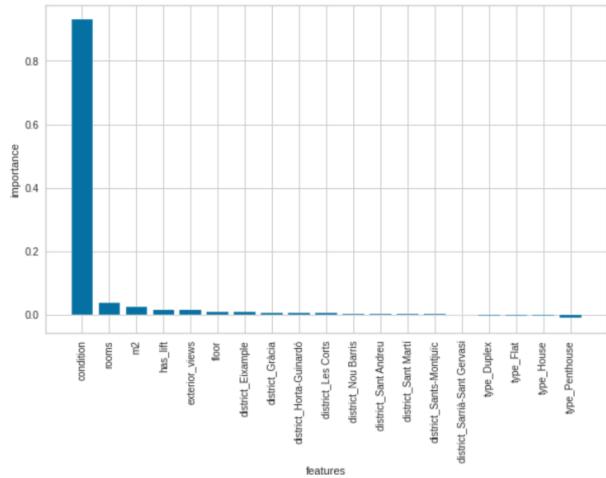


Figura 6.2.4: Representación de la importancia de los atributos a la hora de predecir el valor de una vivienda

6.3 Stacked regressor

El último modelo que vamos a ver durante este proyecto es un modelo que utiliza stacked regressor. Para entrenar este modelo, vamos a utilizar los dos modelos que mejores resultados nos han dado hasta ahora, la regresión ridge y la regresión lasso. Debido a que estamos combinando dos modelos previamente entrenados, este modelo no tiene hiperparámetros que explorar.

La puntuación de validación cruzada que hemos obtenido sobre los datos de entrenamiento es la que vemos en la figura 6.3.1, podemos ver que mantenemos la tendencia vista ahora donde se obtiene un valor sobre el conjunto de entrenamiento entre el 0.74 y 0.81 aproximadamente, en este caso con un valor igual a 0.762.

Stacked regressor 0.762

Figura 6.3.1: Valor de la validación cruzada sobre los datos de entrenamiento

Una vez entrenado el modelo, vamos a ver ahora la puntuación R^2 que hemos obtenido sobre el conjunto de datos de test utilizando este mismo modelo.

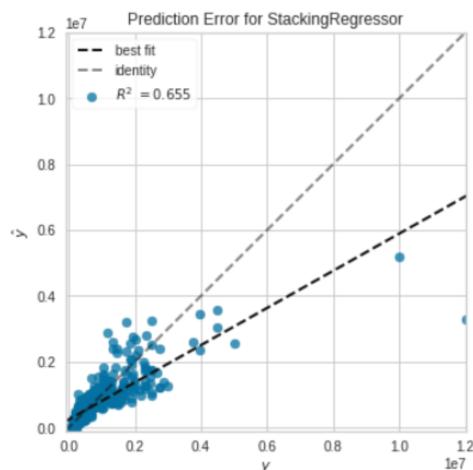


Figura 6.3.2: Resultados obtenidos sobre los datos test utilizando el modelo entrenado

Como podemos ver, hemos obtenido como valor de R^2 0.655, algo mejor que otros modelos, pero no mejor que la regresión lasso o la regresión ridge, este valor comparado al que veíamos durante el entrenamiento es bastante peor, hemos perdido alrededor de 0.1 de acierto. Tal y como hemos mencionado con los otros modelos, probablemente debido al sobreajuste que se está produciendo durante el entrenamiento del propio modelo. La situación ideal sería poder aumentar el número de pliegues realizados durante el entrenamiento pero debido al coste computacional que esto supone, no hemos podido hacerlo.

También hemos representado la importancia de los distintos atributos de una vivienda a la hora de predecir el precio de la misma con este modelo. Como podemos ver en la siguiente figura, parece que el atributo decisivo a la hora de predecir un precio, es la condición de la propia vivienda, el resto de atributos, por otro lado, tienen una importancia prácticamente negligible.

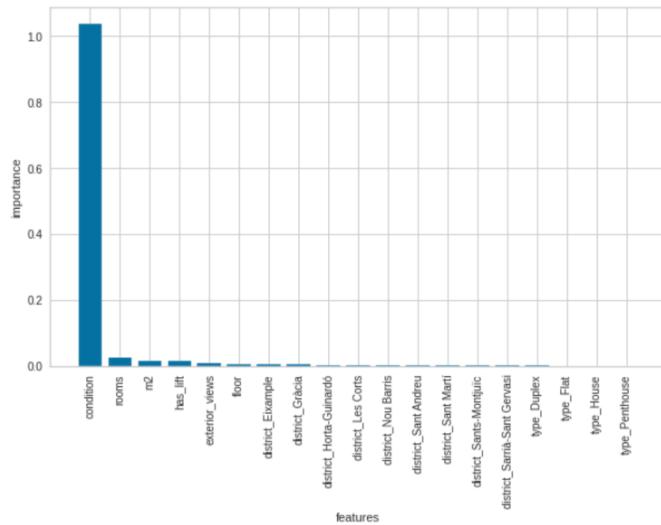


Figura 6.2.4: Representación de la importancia de los atributos a la hora de predecir el valor de una vivienda

7. Mejor modelo

Los dos modelos que en un principio pensábamos que iban a dar un peor resultado, por ser más sencillos a priori, han acabado siendo los que en comparación al resto han proporcionado mejores resultados sobre el conjunto test. Junto a estos dos se encuentra también el modelo que utiliza stacked regressor combinando las dos regresiones. El gradient boosting y el random forest son los dos modelos que más nos han decepcionado, podemos ver como sobre el conjunto de entrenamiento tienen valores de 0.8 pero luego sobre el conjunto test su rendimiento es mucho peor, debido claramente al overfitting.

| | test_score | cross_val_score |
|-------------------|------------|-----------------|
| LASSO regression | 0.659 | 0.761 |
| Ridge regression | 0.657 | 0.762 |
| Stacked regressor | 0.655 | 0.762 |
| Gradient boosting | 0.630 | 0.813 |
| Random forest | 0.622 | 0.799 |
| Polynomial SVM | 0.610 | 0.740 |

Figura 7.1: Rendimiento de los distintos modelos

El mejor de todos los modelos entonces ha sido el de la regresión lasso con la que hemos obtenido un valor R^2 de casi el 0.66, lo cual visto nuestro conjunto de datos y el resto de modelo podemos concluir que es un resultado aceptable. Además, podemos observar que es tan solo el modelo que mejor R^2 ha obtenido, sino que además, es el modelo que ha perdido menos puntuación a la hora de generalizar (a pesar de que sigue siendo una generalización no ideal). Por estos dos motivos, si tuviéramos que usar alguno de estos modelos, el modelo que utiliza la regresión lasso sería el que elegiríamos.

Pudiendo entrenar este modelo con un dataset más amplio y con mayor capacidad computacional probablemente nos permitiría aumentar nuestra tasa de acierto hasta obtener unos valores buenos de verdad. Aunque si se diera el caso, quizás la importancia de los atributos, los hiperparámetros y seguramente hasta el mejor modelo cambiarían probablemente.

Utilizar este modelo para predecir viviendas de otras ciudades probablemente no tendría un buen resultado. Quizás en otras ciudades los barrios tienen más importancia o la calidad de vida de la gente es mayor o menor y por tanto los precios cambian drásticamente también. Además de que nuestro modelo ha sido entrenado con información de los barrios de Barcelona que obviamente en otras ciudades no existirán.

8. Análisis de interpretabilidad

En los modelos hemos podido comprobar que el atributo con más importancia es el atributo que define la condición en la que se encuentra una vivienda (*condition*), seguida por los atributos que nos indican el número de habitaciones (*rooms*), su superficie (*m2*), si tiene o no ascensor (*has_lift*), si tiene vistas exteriores (*exterior_views*), el número de piso (*floor*) y por último seguido por los distintos distritos y el resto de variables. Este comportamiento se ha visto reflejado de manera idéntica en todos los modelos tanto lineales como no lineales, la importancia atribuida quizá varía levemente en algunos modelos, pero generalmente el comportamiento respecto a los atributos ha sido igual en todos los modelos. En este aspecto podemos destacar el modelo Polynomial SVM donde los demás atributos ganan una importancia destacable comparada con el resto de modelos.

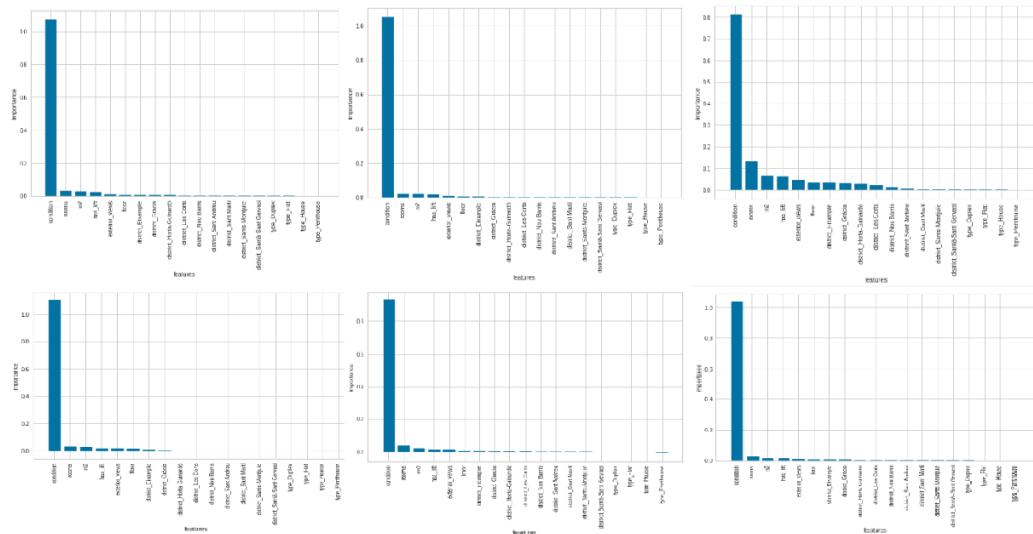


Figura 8.1: Representación de la importancia de los atributos de los 6 modelos

Tal y como hemos ido comentando en los distintos modelos, hemos observado un patrón que se ha ido repitiendo durante los modelos que hemos ido entrenando. En la figura 8.2 podemos ver la interpretación de las predicciones en contra de los valores reales de los individuos para el modelo ridge. Como podemos ver la distribución es algo anómala.

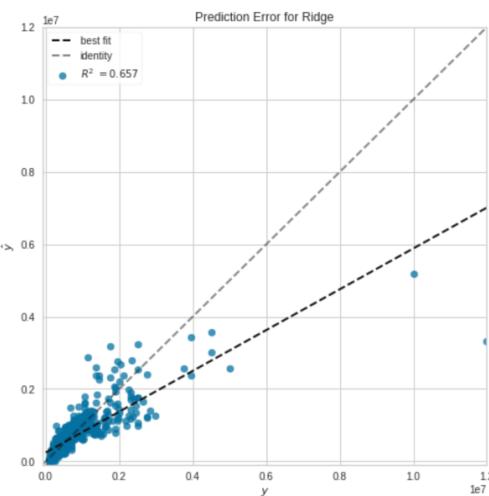


Figura 8.2: Representación de las predicciones en contra de los valores reales de los individuos

Esto se debe a que, por un lado, el número de viviendas de precios altos es mucho menor al resto de precios más bajos y por eso cuando lo representamos gráficamente, la mayoría de viviendas quedan acumuladas al principio y hay algunas que sobresalen respecto al resto. El otro motivo va relacionado también con el bajo número de viviendas de más coste, al tener menos información sobre estas probablemente el precio de estas se está estimando más bajo del que en realidad es.

Podemos ver ahora como con un modelo no lineal como puede ser el de gradient boosting tenemos un comportamiento bastante similar, incluso algo más acentuado. En la siguiente figura vemos que también las viviendas de precio “normal” se apelmazan al principio de la gráfica mientras algunas otras destacan sobre el resto. Probablemente aquí también estemos perdiendo viviendas caras que acaban siendo predichas con precios más bajos,

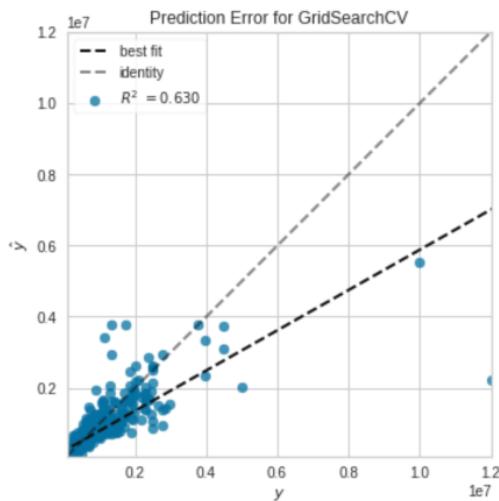


Figura 8.3: Representación de las predicciones en contra de los valores reales de los individuos

9. Conclusión

Creemos que el trabajo que hemos realizado durante el proyecto es adecuado y estamos contentos con todo lo que hemos aprendido y hemos podido profundizar sobre los distintos modelos con los que hemos trabajado. Por otro lado, nos hubiera gustado tener algo más de conocimiento sobre los modelos no lineales ya que probablemente esto nos hubiera facilitado en gran cantidad el trabajo hecho y hubiéramos sido más ágiles. Dado el conjunto de datos que teníamos, creemos que hemos obtenido, relativamente hablando, buenos resultados. Nos quedamos con la duda de si, por ejemplo, en el modelo que utilizaba random forest podríamos haber ajustado otros hiperparámetros que nos habrían podido ser de más utilidad, pero el límite computacional no nos permitía ajustar tanto parámetros.

La principal limitación que hemos tenido, más allá del computacional, ha sido el propio conjunto de datos. Consideramos que no estaba lo suficientemente bien balanceado y que no era tan extenso como nos habría gustado. Con más individuos en el conjunto de datos y con más viviendas de precios altos, estamos seguros que podríamos haber obtenido mejores resultados de los que hemos obtenido durante la práctica. Además, aparte del atributo que definía la condición de la vivienda, los demás atributos se consideraban prácticamente ruido, con más individuos quizás veríamos situaciones en las que este efecto quedaría minimizado considerablemente.

10. Webgrafía

[1] Aprendentatge Automàtic. (2022, September 28). Aprendentatge Automàtic.

<https://sites.google.com/upc.edu/aprendentatge-automatic/p%C3%A1gina-principal?authuser=2>

[2] Barcelona_Idealista_HousingPrices. (n.d.). Kaggle.

<https://www.kaggle.com/datasets/jorgeglez/barcelona-idealista-housingprices>

[3] Shah, R. (2021, June 23). GridSearchCV / Tune Hyperparameters with GridSearchCV. Analytics Vidhya.

<https://www.analyticsvidhya.com/blog/2021/06/tune-hyperparameters-with-gridsearchcv/>

[4] sklearn.decomposition.PCA — scikit-learn 1.2.0 documentation. (n.d.). Scikit-learn.

<https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>

[5] sklearn.ensemble.GradientBoostingRegressor — scikit-learn 1.2.0 documentation. (n.d.). Scikit-learn.

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html>

[6] sklearn.ensemble.RandomForestClassifier — scikit-learn 1.2.0 documentation. (n.d.). Scikit-learn.

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

[7] sklearn.ensemble.StackingRegressor — scikit-learn 1.2.0 documentation. (n.d.). Scikit-learn.

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingRegressor.html>

[8] sklearn.inspection.permutation_importance — scikit-learn 1.2.0 documentation. (n.d.). Scikit-learn.

https://scikit-learn.org/stable/modules/generated/sklearn.inspection.permutation_importance.html

[9] sklearn.linear_model.Lasso — scikit-learn 1.2.0 documentation. (n.d.). Scikit-learn.

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html

[10] sklearn.linear_model.Ridge — scikit-learn 1.2.0 documentation. (n.d.). Scikit-learn.

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

[11] SVM Hyperparameter Tuning using GridSearchCV / ML. (n.d.). GeeksforGeeks.

<https://www.geeksforgeeks.org/svm-hyperparameter-tuning-using-gridsearchcv-ml/>