

Inteligencia de Negocio Práctica 3: Competición en Kaggle

Germán José Padua Pleguezuelo

germanpadua@correo.ugr.es

Grupo 2 - 5º DGIIM






7 de enero de 2024



**UNIVERSIDAD
DE GRANADA**

House Prices - Advanced Regression Techniques

[Submit Prediction](#)[...](#)[Overview](#) [Data](#) [Code](#) [Models](#) [Discussion](#) [Leaderboard](#) [Rules](#) [Team](#) [Submissions](#)

#	Team	Members	Score	Entries	Last	Join
81	JoseJuanMorales_UGR_I N		0.11681	43	18h	
99	GermanJosePadua_UGR _IN		0.11788	23	1h	
 Your Best Entry! Your submission scored 0.11818, which is not an improvement of your previous score. Keep trying!						
387	DavidAlcala_UGR_IN		0.12249	16	1d	
401	DanielCarrasco_UGR_IN		0.12260	13	1d	

Índice

1. Introducción	4
2. Tabla de resultados	5
3. Desarrollo de las soluciones	8

1. Introducción

Esta práctica se centra en la competición "House Prices - Advanced Regression Techniques" de Kaggle, donde el objetivo principal es predecir el precio de venta de una vivienda basándose en un conjunto diverso de características.

El conjunto de datos proporcionado incluye 1460 instancias y 81 atributos, abarcando tanto variables numéricas como categóricas. La tarea es modelar y predecir el precio de venta (*SalePrice*), utilizando estos atributos.

El rendimiento de los modelos predictivos se evaluará utilizando la raíz del error cuadrático medio (RMSE) entre los logaritmos de los precios predichos y los observados.

En esta memoria, se documentan las diversas estrategias y técnicas de modelado exploradas para abordar este problema. Se inicia con métodos básicos como árboles de decisión y Random Forest, seguido por técnicas avanzadas como Stacking y algoritmos especializados. El proceso incluye la experimentación con diferentes combinaciones de algoritmos, ajustes de parámetros y técnicas de preprocesamiento de datos, todo con el objetivo de aproximarse lo más posible al precio real de las viviendas.

2. Tabla de resultados

Cuadro 1: Resultados

Fecha y hora	Pos.	Score (Tr.)	Score (Kagg.)	Preprocesado	Descripción de los algoritmos
12/12/2023 19:44	4736	-	0.40613	Ninguno	Fichero de ejemplo de resultados
21/12/2023 13:00	4098	0.04586	0.20633	Solamente valores perdidos. Sustituir en las variables categóricas por el valor más frecuente y en las numéricas por la mediana	Árbol de Decisión para regresión
24/12/2023 18:15	2557	0.02177	0.14762	Imputar valores perdidos por la media (numéricas) y por el valor más frecuente (categóricas)	Tuning básico de Random Forest
24/12/2023 19:11	1921	0.01883	0.13965	Imputar valores perdidos por la media (numéricas) y por el valor más frecuente (categóricas)	Stacking Regressor (Random Forest, XGBoost, KNN y gradient boosting).
24/12/2023 20:05	1614	0.016835	0.13573	Imputar valores perdidos por la media (numéricas) y por el valor más frecuente (categóricas)	Stacking Regressor (Random Forest, XGBoost, KNN y gradient boosting).
25/12/2023 18:15	1614	-9.336e-05	9.459	Imputar valores perdidos por la media (numéricas) y por el valor más frecuente (categóricas)	Stacking Regressor (Random Forest, XGBoost, KNN, gradient boosting y Lasso).
25/12/2023 19:00	1136	0.016235	0.13072	Imputar valores perdidos por la media (numéricas) y por el valor más frecuente (categóricas)	Stacking Regressor (Random Forest, XGBoost, KNN, gradient boosting).
26/12/2023 00:10	1136	0.0162	0.13146	Imputar valores perdidos con KNN (numéricas) y por el valor más frecuente (categóricas)	Stacking Regressor (Random Forest, XGBoost, KNN, gradient boosting, Lasso).

Fecha y hora	Pos.	Score (Tr.)	Score (Kagg.)	Preprocesado	Descripción de los algoritmos
26/12/2023 3:00	1136	-	0.14632	Imputar valores perdidos con KNN (numéricas) y por el valor más frecuente (categóricas)	Stacking Regressor (Random Forest, XGBoost, LightGBM y LinearRegressor).
26/12/2023 13:50	995	0.01524	0.12894	Imputar valores perdidos con KNN (numéricas) y por el valor más frecuente (categóricas)	Stacking Regressor (tuning Random Forest, XGBoost, LightGBM, Gradient Boosting, KNN, Lasso y Ridge).
26/12/2023 16:00	995	0.01524	0.129	Imputar valores perdidos con KNN (numéricas) y por el valor más frecuente (categóricas)	Stacking Regressor (tuning Random Forest, XGBoost, LightGBM, Gradient Boosting, KNN, Lasso y Ridge). Probado con distintos alphas para Ridge
26/12/2023 19:22	500	0.12381	0.12458	Imputar valores perdidos con KNN (numéricas) y por el valor más frecuente (categóricas). Aplicar log a SalePrice	Stacking Regressor (tuning Random Forest, XGBoost, LightGBM, Gradient Boosting, KNN, Lasso y Ridge)
02/01/2024 01:15	207	0.1224	0.12018	Borrar variables con muchos valores perdidos. Imputar con KNN (numéricas) y por el valor más frecuente (categóricas). Aplicar log a SalePrice	Stacking Regressor (tuning Random Forest, XGBoost, LightGBM, Gradient Boosting, KNN, Lasso y Ridge)
06/01/2024 16:00	117	0.12154	0.11914	Borrar variables con muchos valores perdidos. Imputar con KNN (numéricas) y por el valor más frecuente (categóricas). Aplicar log a SalePrice	Stacking Regressor añadiendo catboost y SVR
06/01/2024 17:11	117	0.12215	0.12094	Borrar variables con muchos valores perdidos. Imputar con KNN (numéricas) y por el valor más frecuente (categóricas). Aplicar log a SalePrice	Stacking Regressor con la selección de las 50 variables más importantes en algunos algor.

Fecha y hora	Pos.	Score (Tr.)	Score (Kagg.)	Preprocesado	Descripción de los algoritmos
06/01/2024 18:50	117	0.121056	0.12099	Imputar con KN-N/moda. OneHot Encoding. Selección de características. Elimino 5 outliers.	Stacking Regressor
06/01/2024 19:48	107	0.11999	0.11847	Imputar con KN-N/moda. OneHot Encoding. Selección de características. Elimino 5 outliers.	Stacking Regressor
06/01/2024 21:12	99	0.1195	0.11788	Imputar con KN-N/moda. OneHot Encoding. Selección de características. Elimino 5 outliers.	Stacking Regressor algunos solamente con las variables menos importantes
07/01/2024 1:22	99	0.11127	0.12336	Imputar con KN-N/moda. OneHot Encoding. Selección de características. Elimino 5 outliers.	Stacking Regressor algunos solamente con las variables menos importantes.
07/01/2024 2:30	99	0.1108	0.12385	Imputar con KN-N/moda. OneHot Encoding. Selección de características. Elimino 5 outliers.	Stacking Regressor algunos solamente con las variables menos importantes.
07/01/2024 14:06	99	0.12895	0.11907	Imputar con KN-N/moda. OneHot Encoding. Selección de características. Elimino 5 outliers.	Stacking Regressor (Random Forest, XGBoost, LightGBM, Gradient Boosting, KNN, Lasso, CatBoost, SVR y Ridge)
07/01/2024 15:44	99	0.11944	0.11811	Imputar con KN-N/moda. OneHot Encoding. Selección de características. Elimino 5 outliers.	Stacking Regressor (Random Forest, XGBoost, LightGBM, Gradient Boosting, KNN, Lasso, CatBoost, SVR y Ridge)
07/01/2024 16:53	99	0.119757	0.11818	Imputar con KN-N/moda. OneHot Encoding. Selección de características. Elimino 5 outliers.	Stacking Regressor (Random Forest, XGBoost, LightGBM, Gradient Boosting, KNN, Lasso, CatBoost, SVR y Ridge)

3. Desarrollo de las soluciones

En primer lugar, comencé ejecutando los códigos de ejemplo proporcionados por el profesor: un árbol de decisión y un Random Forest haciendo una búsqueda para encontrar los mejores parámetros.

Posteriormente, avancé hacia técnicas más sofisticadas como el Stacking, esto es, predecimos con varios algoritmos distintos y usamos un estimador final para ponderar los resultados de cada uno y obtener una predicción final. Sci-kit learn ya implementa Stacking, por lo que facilita el trabajo. Probé con distintos algoritmos y parámetros, y en general usé Ridge para combinar la salida de los algoritmos base. Para la implementación de Stacking me fijé en el siguiente Notebook de Kaggle [1]. Pero básicamente lo he hecho así:

```
1 # Importar las librerías de manejo de pipelines
2 from sklearn.pipeline import Pipeline
3
4 # Importar los modelos y regresores específicos
5 from xgboost import XGBRegressor
6 import lightgbm as lgb
7 from sklearn.linear_model import Ridge
8 from sklearn.ensemble import StackingRegressor
9
10 # XGBoost pipeline
11 xgboost_pipeline = Pipeline([
12     ('xgboost25', XGBRegressor(objective='reg:squarederror', colsample_bytree=0.5,
13         gamma=0.05,
14         learning_rate=0.05, max_depth=3,
15         min_child_weight=1.5, n_estimators=3000,
16         reg_alpha=0.5, reg_lambda=0.8,
17         subsample=0.5,
18         random_state =42, seed=42))
19 ])
20
21 # LightGBM pipeline
22 lightgbm_pipeline = Pipeline([
23     ('lightgbm', lgb.LGBMRegressor(objective='regression', verbose=-1,
24         bagging_fraction = 0.8, bagging_seed=9, seed=42, n_estimators=300, max_depth=4,
25         learning_rate=0.15))
26 ])
27
28 # Definimos los modelos bases
29 base_models = [
30     ('xgboost', xgboost_pipeline),
31     ('lightgbm', lightgbm_pipeline)
32 ]
33
34 # Definimos el regresor stacking
35 stacking_regressor = StackingRegressor(estimators=base_models, final_estimator=Ridge())
```

Listing 1: Extracto del código para Stacking

A continuación, intenté no usar la implementación de sci-kit learn para ver si podía incluir algún atributo más para el Stacking, como el error o la puntuación media en la validación cruzada por algoritmo. Sin embargo, no tuve mucho éxito, ya que el conjunto de test debe tener las mismas columnas que el conjunto de entrenamiento. Lo que sí que mejoré fue el preprocesamiento de los datos, implementé que los valores numéricos perdidos se imputen con KNN.

Para seguir mejorando los resultados, decidí eliminar las variables con más del 45% de valores perdidos. Esta decisión se basó en la insuficiencia de datos para una imputación efectiva en esas variables. Además, mirando la descripción original del dataset por el autor Dean Cock [2]. Él mismo sugiere que eliminemos 5 ejemplos del conjunto de entrenamiento que son outliers, porque tres de ellos los considera ventas parciales que no representan el valor real del mercado y los otros dos son ventas muy inusuales (casas muy grandes con precio muy bajo). También realicé una búsqueda de los mejores parámetros para los algoritmos utilizando GridSearch.

En el siguiente gráfico se observan claramente 4 de los 5 outliers mencionados.



Figura 1: Gráfico de la variable GrossLivArea frente a SalePrice

Finalmente, incluí más algoritmos al Stacking. Visualizando la importancia de las variables que cada algoritmo da me di cuenta que muchas coincidían, por lo que decidí duplicar la mayoría de los regresores pero introduciendo cierta selección de características. Algoritmos como Random Forest, Gradient Boosting, XGBoost y LightGBM incluyo tres variantes. La primera, predecir solamente con las 25 variables más importantes. En segundo lugar, seleccionar las 100 variables más importantes pero quitando las 5 mayores. Y tercero, predecir utilizando las 100 variables menos importantes. Con esto conseguimos expandir el espacio de búsqueda de soluciones y puede que en ciertos casos concretos, algunas variables que son menos importantes en general sean determinantes. También añadí dos regresores: CatBoost y SVR(Support Vector Regression). Por lo tanto, la arquitectura final del predictor fue la siguiente: como estimadores iniciales Random Forest, Gradient Boosting, XGBoost y LightGBM cada uno con las tres variantes mencionadas previamente; KNN, CatBoost y SVR. Y como estimador final Ridge. A pesar de experimentar con varios ajustes de parámetros, como la profundidad de los árboles en Random Forest y el número de estimadores, no logré superar una de las puntuaciones anteriores.

Esta arquitectura final, que combina una selección estratégica de características con una diversidad de algoritmos y un estimador final robusto, representa mi enfoque más completo para la predicción de precios de viviendas en la competición de Kaggle.

Referencias

- [1] URL: <https://www.kaggle.com/code/htetmyet/stacking-ensemble-regressor-model>.
- [2] URL: <https://jse.amstat.org/v19n3/decock/DataDocumentation.txt>.