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.

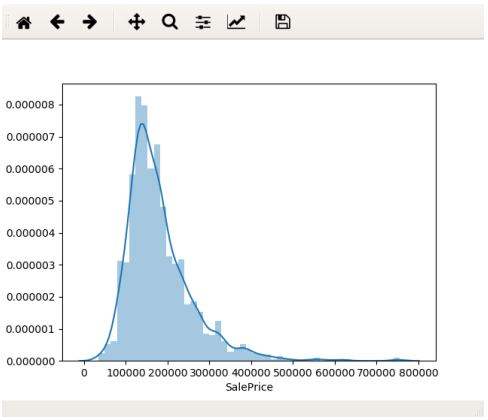

Figura 3.1: Valores sesgados de SalePrice.

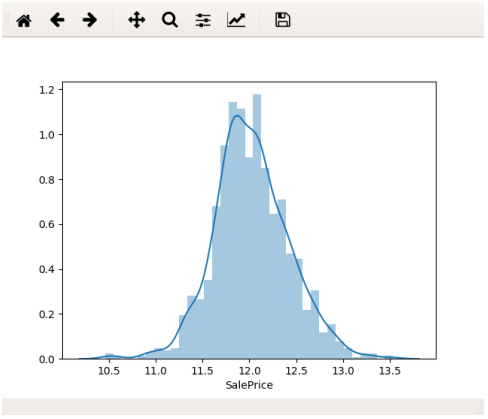

Figura 3.2: Valores no sesgados de SalePrice tras logaritmo.

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

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`