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

Memoria Práctica 3.
Competición de Kaggle.

Alberto Armijo
armijoalb@correo.ugr.es

8 de enero de 2018

Índice

Índice de figuras

Índice de tablas

1. Introducción.

En esta práctica se explicarán los pasos seguidos durante la competición hasta conseguir el resultado que se puede ver en la clasificación.

La competición de Kaggle se trata de resolver un problema de regresión para predecir el valor de una casa según un conjunto de características, como el número de habitaciones, el espacio que tiene el garaje, el espacio de cada una de las plantas de la vivienda, el año en que se construyo, etc...

Para resolver el problema, se utilizarán algunos algoritmos que se han mostrado en la asignatura de prácticas, como por ejemplo XGBoost (Gradient Boosting).

2. Explicación código desarrollado.

En este apartado se explicarán las medidas tomadas a la hora de preprocesar los datos y ejecutar los diferentes algoritmos utilizados para predecir el valor de las viviendas.

2.1. Estudio de los datos.

Lo primero que tenemos que miré fue el tamaño de nuestro conjunto de datos, para el training contamos con 1460 datos y para el test contamos con 1459 datos.

Lo siguiente que hice es comprobar comprobar las columnas que tienen valores perdidos y el porcentaje de valores perdidos por cada una de las variables. Estos son las variables con valores perdidos (en el conjunto de train).

```
total = train.isnull().sum().sort_values(ascending=False)
percent = (train.isnull().sum()/train.isnull().count()).
    sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total',
     'Percent'])
missing_data = missing_data[missing_data['Total'] > 0]
names = list(missing_data.index)
sns.barplot(x=names,y='Percent',data=missing_data)
plt.xticks(rotation=45)
plt.savefig('missing_data.pdf')
```

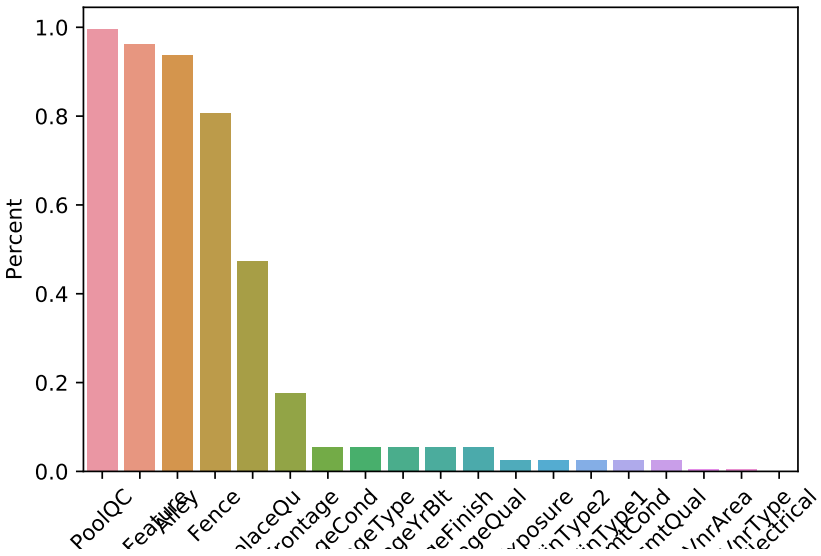

Figura 2.1: Datos pérdidos en train

Para todas estas variables tendremos o que borrar la variable o imputar los datos, eso se verá en el apartado de preprocesado de datos.

Lo siguiente que haremos es ver que variables numéricas son las más importantes, y que variables categóricas son las más importantes conforme a la variable SalePrice.

Para calcular las variables que están más correlacionadas, haremos una matriz de correlación, esto lo podemos hacer con pandas directamente. Después, crearemos un heatmap para representar las correlaciones entre las variables.

```
f, ax = plt.subplots(figsize=(12, 9))
corrmat = train.corr()
sns.heatmap(corrmat)
plt.xticks(rotation=45)
plt.savefig('correlation.pdf')
```

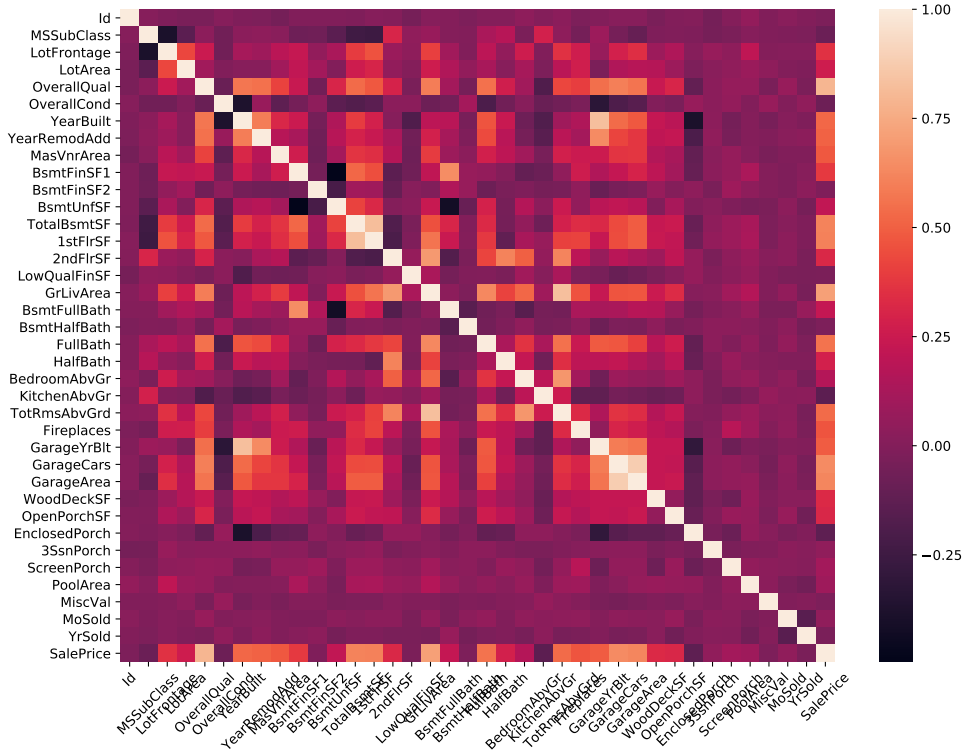

Figura 2.2: Correlación variables numéricas.

Como se puede ver, las variables que más correlación tienen con SalePrice son: OverQual, GrLivArea, GarageCars, GarageArea, TotalBsmtSF, 1stFlrSF, FullBath,TotRmsAbvGrd,YearBuilt.

Para calcular las variables categóricas más relacionadas con SalePrice, se ha utilizado el test ANOVA, después, calcularemos la disparidad de las variables haciendo log(1/resAnova[variable]). El código utilizado y las variables más relacionadas son las siguientes.

```
def anova(frame):
anv = pd.DataFrame()
anv['feature'] = qualitative
pvals = []
for c in qualitative:
samples = []
for cls in frame[c].unique():
s = frame[frame[c] == cls]['SalePrice'].values
samples.append(s)
pval = stats.f_oneway(*samples)[1]
pvals.append(pval)
anv['pval'] = pvals
```

```
    return anv.sort_values('pval')

a = anova(train)
a['disparity'] = np.log(1./a['pval'].values)
sns.barplot(data=a, x='feature', y='disparity')
plt.savefig('anova.pdf')
x=plt.xticks(rotation=90)
```

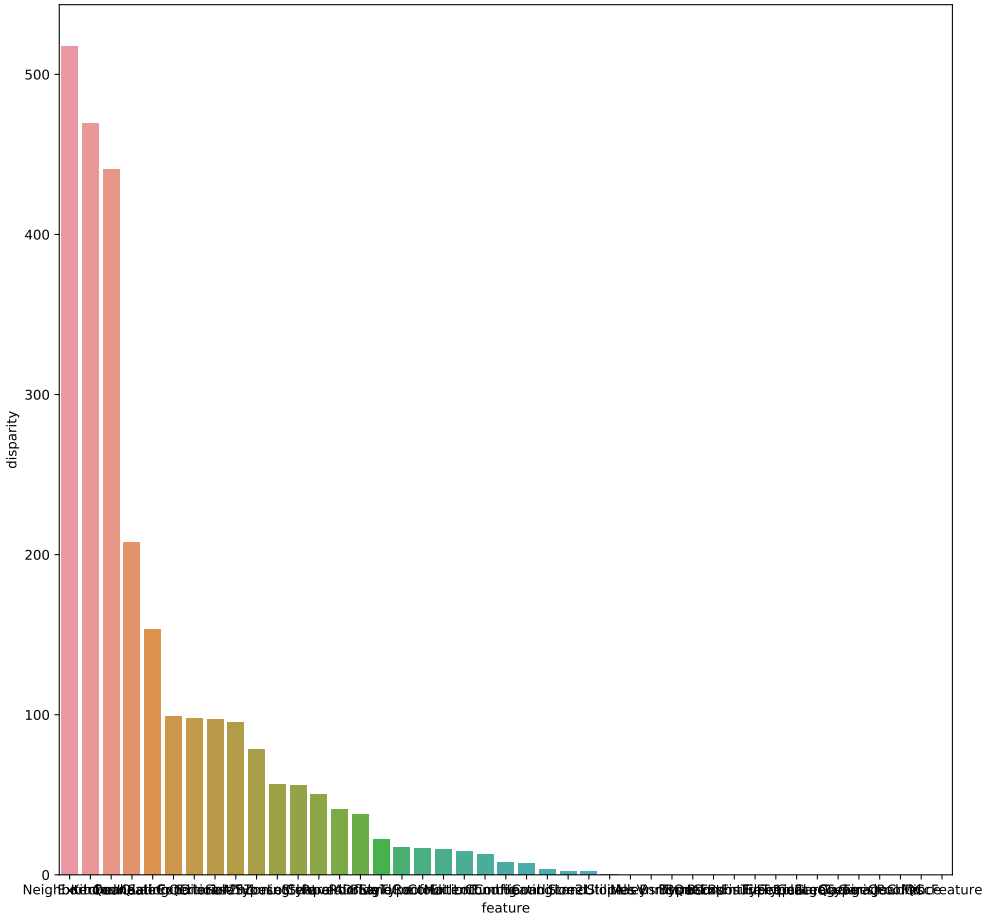

Figura 2.3: Relación variables categóricas.

Aunque en la gráfica no se pueda ver por problemas con los nombres de las variables,

las variables cualitativas más importantes son: Neighborhood, ExterQual, KitchenQual, Foundation, HeatingQC, SaleCondition, Exterior1st,...

Ahora que ya hemos estudiado los datos del conjunto de train, preprocesaremos los datos.

2.2. Preprocesado de datos.

Lo primero que haremos en este apartado será eliminar los outliers del conjunto de train, la única variable que contiene outliers es GrLivArea.

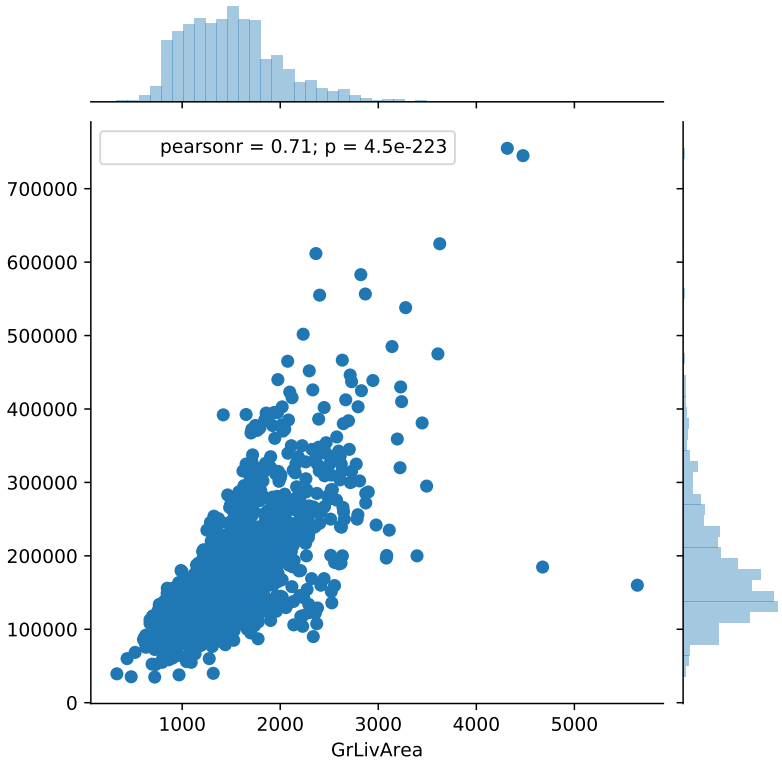

Figura 2.4: Valores GrLivArea conforme a SalePrice

Como se puede ver, hay dos valores para los cuales el valor de venta de la casa es bajo para un terreno muy grande. Por ello eliminaremos esos valores.

Lo siguiente que haremos es imputar valores pérdidos tanto para train como para test. Para ello, lo primero que haremos es unir ambos conjunto en un solo DataFrame, y después comenzaremos a imputar los valores perdidos.

Los variables con valores perdidos son las siguientes: 'PoolQC', 'MiscFeature', 'Alley', 'Fence', 'FireplaceQu', 'LotFrontage', 'GarageCond', 'GarageQual', 'GarageYrBlt', 'GarageFinish', 'GarageType', 'BsmtCond', 'BsmtExposure', 'BsmtQual', 'BsmtFinType2',

'BsmtFinType1', 'MasVnrType', 'MasVnrArea', 'MSZoning', 'BsmtHalfBath', 'Utilities', 'Functional', 'BsmtFullBath', 'BsmtFinSF1', 'Exterior1st', 'Exterior2nd', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'SaleType', 'Electrical', 'KitchenQual', 'GarageArea', 'GarageCars'.

Las variables 'Fence', 'Alley' y 'MiscFeatures' tienen muchos valores perdidos, por lo que he decidido eliminarlas.

Para todas las variables numéricas, he imputado todos los valores perdidos con 0s.

Para las variables cualitativas se han empleados diferentes formas de imputar los datos. Para las variables de tipo calidad, como por ejemplo 'GarageQual', se ha creado un diccionario en python donde se representa cada uno de los diferentes valores con un número del 1 a 5, y los valores perdidos se representan con un 0.

```
diccionario = {np.nan:0, "Po":1, "Fa":2, "TA":3, "Gd":4, "Ex":5}
name = [name for name in missing_data if "QC" in name
or "Qual" in name or "Cond" in name
or "Qu" in name]

for n in name:
        features[n] = features[n].map(diccionario).astype(int)
```

Este mismo proceso se ha seguido para las variables 'GarageFinish' y 'Utilities'.
Para las variables 'BsmtFinTypeX' y 'BmstExposure' se ha optado por reemplazar los valores perdidos por los valores más votados.

```
name = ["BsmtFinType2","BsmtFinType1","BsmtExposure"]
for n in name:
        features[n] = features[n].fillna(features[n].mode()[0])
```

El mismo proceso se ha seguido para las variables 'Electrical' y 'Functional'.

Para el resto de características que tenemos que imputar los datos, se han sustituido por valores de dentro de sus características, para ello se ha utilizado la siguiente función.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
def factorize(data, var, fill_na = None):
if fill_na is not None:
data[var].fillna(fill_na, inplace=True)
le.fit(data[var])
data[var] = le.transform(data[var])
return data
```

Lo siguiente que se hará será binarizar todas las variables cualitativas. Para ello se ha utilizado lo siguiente.

```
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)
```

Lo último que haremos será modificar las variables que tienen una asimetría alta, para ello calcularemos la asimetría de cada de las variables y aquellas que tengan un valor mayor a 0.75 se modificará su valor con el logaritmo. Para usar el logaritmo se ha optado por la función log1p() ya que algunos valores tienen valores iguales a 0 y por lo tanto nos pueden ocasionar problemas a la hora de utilizarlos en los modelos. Tras esto se han estandarizado los datos con el paquete StandardScaler de sklearn.

```
# Transformamos las variables con mucha varianza con el
    logaritmo.
from scipy.stats import skew
skewed = x_train[numeric_features].apply(lambda x: skew(x
    .dropna().astype(float)))
skewed = skewed[skewed > 0.75]
skewed = skewed.index

x_train[skewed] = np.log1p(x_train[skewed])
x_test[skewed] = np.log1p(x_test[skewed])
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(x_train[numeric_features])

scaled = scaler.transform(x_train[numeric_features])
for i, col in enumerate(numeric_features):
x_train[col] = scaled[:, i]

scaled = scaler.transform(x_test[numeric_features])
for i, col in enumerate(numeric_features):
x_test[col] = scaled[:, i]
```

3. Tabla resultados en la competición.

	Nombre	Día	Hora	Posición	Score Kaggle	RMSLE training
0	submission_one	27/12/17	19:00:00	~1000	0.12539	0.092997
1	submission_x	27- 28/12	varias horas	~1000	0.13-0.15	0.1 – 0.078
2	submission_29_12_2	29/12/17	13:40:00	~1000	0.12566	0.092997
3	Submission_29_12 4	29/12/17	18:20:00	~1000	0.12576	0.078509
4	Submission_30 12 1	30/12/17	11:30:00	~1000	0.13142	0.050387
5	Submission_30 12 2	30/12/17	14:58:00	~650	0.12095	0.075196
6	Submission_30 12 3	30/12/17	14:59:00	~600	0.12070	0.073795
7	Submission_30_12 4	30/12/17	14:59:00	~580	0.11963	0.086440
8	Submisision_30_12 5	30/12/17	15:00:00	~580	0.12039	0.075087
9	Submission_30_12 6	30/12/17	15:00:00	~580	0.12149	0.073679
10	Submission_30_12 7	30/12/17	15:00:00	~580	0.12127	0.086382
11	Submission_30_12 8	30/12/17	15:02:00	~580	0.12127	0.088779
12	Submission_31_12 1	31/12/17	14:05:00	550	0.11948	0.083367
13	Submission_31_12 2	31/12/17	14:09:00	550	0.11992	0.082508
14	Submission_31_12 3	31/12/17	14:15:00	550	0.11977	0.085418
15	Submission_31_12 4	31/12/17	11:23:00	550	0.12028	0.080700
16	Submission_31_12 5	31/12/17	15:28:00	550	0.11983	0.080339
17	Submission_31_12 6	31/12/17	15:38:00	550	0.11994	0.078922

El resto de elementos de la tabla ocupan demasiado espacio para representarlas en filas, por lo tanto, se hará un apartado por cada una de las soluciones subidas a Kaggle describiendo brevemente cada uno de los elementos.

3.1. Submission_one

3.1.1. Breve descripción del preprocesado.

El preprocesado es el que viene por defecto en el archivo "kaggle.py".

3.1.2. Breve descripción de los algoritmos utilizados.

Los algoritmos que se utilizan para obtener la solución son GradientBoosting y Elastic-Net; para obtener la solución final, se hace una combinación de los dos, utilizando para calcular para cada uno de los precios la suma de la mita de cada una de los precios calculados.

3.1.3. Configuración de los parámetros de los algoritmos.

La configuración de los parámetros es la que viene por defecto en el archivo "kaggle.py"

3.2. Submission_x

3.2.1. Breve descripción del preprocesado.

El preprocesado es el que viene por defecto en el archivo "kaggle.py"

3.2.2. Breve descripción de los algoritmos utilizados.

Los algoritmos utilizados durante las entregas realizadas en los días 27/12 y 28/12 fueron ElasticNet, GradientBoosting, XGBoost, LightGBM. Se realizaron entregas probando la solución aportada por los algoritmos solamente o haciendo modelos complejos entre ellos.

3.2.3. Configuración de los parámetros de los algoritmos.

La configuración para ElasticNet y GradientBoosting es la misma que se aportaba en el archivo "kaggle.py", los algoritmos de XGBoost y LightGBM utilizan un número de estimadores igual a 3000.

3.3. Submission-29-12-2

3.3.1. Breve descripción del preprocesado.

El preprocesado es el que viene por defecto en el archivo "kaggle.py".

3.3.2. Breve descripción de los algoritmos utilizados.

Los algoritmos que se utilizan para obtener la solución son GradientBoosting y ElasticNet; para obtener la solución final, se hace una combinación de los dos, utilizando para calcular para cada uno de los precios la suma de la mita de cada una de los precios calculados.

3.3.3. Configuración de los parámetros de los algoritmos.

La configuración de los parámetros es la que viene por defecto en el archivo "kaggle.py"

3.4. Submission-29-12-4

3.4.1. Breve descripción del preprocesado.

El preprocesado es el que viene por defecto en el archivo "kaggle.py".

3.4.2. Breve descripción de los algoritmos utilizados.

Para obtener la solución final, se han utilizado dos algoritmos, XGBoost y LassoCV. Para obtener la solución final, se calcula el precio de una vivienda como la suma de la mitad de cada uno de los precios calculados por los algoritmos anteriores.

3.4.3. Configuración de los parámetros de los algoritmos.

Para la configuración del algoritmo XGBoost, se han probado diferentes configuraciones, aumentando el número de estimadores, estableciendo valores diferentes para los atributos *reg_ lambda* y *reg_ alpha*, entre otros. Para el algoritmo de LassoCV se ha probado con cuatro valores de lambda diferentes (1, 0.1, 0.001, 0.0005).

3.5. Submission-30-12-1

3.5.1. Breve descripción del preprocesado.

El preprocesado es el que viene por defecto en el archivo "kaggle.py".

3.5.2. Breve descripción de los algoritmos utilizados.

Para obtener las soluciones, se ha utiliza el algoritmo XGBoost.

3.5.3. Configuración de los parámetros de los algoritmos.

Para la configuración del algoritmo XGBoost, se han probado diferentes configuraciones, aumentando el número de estimadores, estableciendo valores diferentes para los atributos *reg_ lambda* y *reg_ alpha*, entre otros.

3.6. Submission-30-12-2

3.6.1. Breve descripción del preprocesado.

Para esta solución se ha hecho una imputación de todos las variables excepto de *Misc-Features*, *Alley* y *Fence* que se han eliminado. También se han transformado todas las variables categóricas en numéricas, y además se han transformado aquellas variables con dispersión alta haciendo el logaritmo de estos. También se han estandarizado algunas variables de la misma forma que en el archivo "kaggle.py".

3.6.2. Breve descripción de los algoritmos utilizados.

Para obtener la solución final, se han utilizado dos algoritmos, XGBoost y LassoCV. Para obtener la solución final, se calcula el precio de una vivienda como la suma de la mitad de cada uno de los precios calculados por los algoritmos anteriores.

3.6.3. Configuración de los parámetros de los algoritmos.

Los parámetros utilizados son los mismos que para soluciones anteriores.

3.7. Submission-30-12-3

3.7.1. Breve descripción del preprocesado.

Para esta solución se ha hecho una imputación de todos las variables excepto de *MiscFeatures*, *Alley* y *Fence* que se han eliminado. También se han transformado todas las variables categóricas en numéricas, y además se han transformado aquellas variables con dispersión alta haciendo el logaritmo de estos. También se han estandarizado algunas variables de la misma forma que en el archivo "kaggle.py".

3.7.2. Breve descripción de los algoritmos utilizados.

Para obtener la solución final, se han utilizado los algoritmos XGBoost y LassoCV. El precio de una vivienda se calcula como la suma de la solución de XGBoost y el 90 % de la solución de LassoCV divida entre 2.

3.7.3. Configuración de los parámetros de los algoritmos.

Los parámetros utilizados son los mismos que para soluciones anteriores.

3.8. Submission-30-12-4

3.8.1. Breve descripción del preprocesado.

Para esta solución se ha hecho una imputación de todos las variables excepto de *MiscFeatures*, *Alley* y *Fence* que se han eliminado. También se han transformado todas las variables categóricas en numéricas, y además se han transformado aquellas variables con dispersión alta haciendo el logaritmo de estos. También se han estandarizado algunas variables de la misma forma que en el archivo "kaggle.py".

3.8.2. Breve descripción de los algoritmos utilizados.

Para obtener la solución final, se han utilizado los algoritmos XGBoost y LassoCV. El precio de una vivienda se calcula como la suma del 30 % del precio calculado por XGBoost y un 70 % del precio calculado por LassoCV.

3.8.3. Configuración de los parámetros de los algoritmos.

Los parámetros utilizados son los mismos que para soluciones anteriores.

3.9. Submission-30-12-5

3.9.1. Breve descripción del preprocesado.

Para esta solución se ha hecho una imputación de todos las variables excepto de *MiscFeatures*, *Alley* y *Fence* que se han eliminado. También se han transformado todas las variables categóricas en numéricas, y además se han transformado aquellas variables con

dispersión alta haciendo el logaritmo de estos. También se han estandarizado algunas variables de la misma forma que en el archivo "kaggle.py".

3.9.2. Breve descripción de los algoritmos utilizados.

Para obtener la solución final, se han utilizado dos algoritmos, XGBoost y LassoCV. Para obtener la solución final, se calcula el precio de una vivienda como la suma de la mitad de cada uno de los precios calculados por los algoritmos anteriores.

3.9.3. Configuración de los parámetros de los algoritmos.

Para esta solución, se han añadido un nuevo alpha al algorimto LassoCV, el nuevo valor es 0.00099.

3.10. Submission-30-12-6

3.10.1. Breve descripción del preprocesado.

Para esta solución se ha hecho una imputación de todos las variables excepto de *Misc-Features*, *Alley* y *Fence* que se han eliminado. También se han transformado todas las variables categóricas en numéricas, y además se han transformado aquellas variables con dispersión alta haciendo el logaritmo de estos. También se han estandarizado algunas variables de la misma forma que en el archivo "kaggle.py".

3.10.2. Breve descripción de los algoritmos utilizados.

Para obtener la solución final, se han utilizado los algoritmos XGBoost y LassoCV. El precio de una vivienda se calcula como la suma de la solución de XGBoost y el 90 % de la solución de LassoCV divida entre 2.

3.10.3. Configuración de los parámetros de los algoritmos.

La configuración es igual que para la solución **Submission-30-12-5**.

3.11. Submission-30-12-7

3.11.1. Breve descripción del preprocesado.

Para esta solución se ha hecho una imputación de todos las variables excepto de *Misc-Features*, *Alley* y *Fence* que se han eliminado. También se han transformado todas las variables categóricas en numéricas, y además se han transformado aquellas variables con dispersión alta haciendo el logaritmo de estos. También se han estandarizado algunas variables de la misma forma que en el archivo "kaggle.py".

3.11.2. Breve descripción de los algoritmos utilizados.

Para obtener la solución final, se han utilizado los algoritmos XGBoost y LassoCV. El precio de una vivienda se calcula como la suma del 30 % del precio calculado por XGBoost y un 70 % del precio calculado por LassoCV.

3.11.3. Configuración de los parámetros de los algoritmos.

La configuración es igual que para la solución **Submission-30-12-5**.

3.12. Submission-30-12-8

3.12.1. Breve descripción del preprocesado.

Para esta solución se ha hecho una imputación de todos las variables excepto de *Misc-Features*, *Alley* y *Fence* que se han eliminado. También se han transformado todas las variables categóricas en numéricas, y además se han transformado aquellas variables con dispersión alta haciendo el logaritmo de estos. También se han estandarizado algunas variables de la misma forma que en el archivo "kaggle.py".

3.12.2. Breve descripción de los algoritmos utilizados.

Para obtener la solución final, se han utilizado los algoritmos XGBoost y Lasso El precio de una vivienda se calcula como la suma del 30 % del precio calculado por XGBoost y un 70 % del precio calculado por Lasso

3.12.3. Configuración de los parámetros de los algoritmos.

Para el algoritmo Lasso, se ha utilizado un valor de lambda = 0.00099 y el número máximo de iteraciones es 50000.

3.13. Submission-31-12-1

3.13.1. Breve descripción del preprocesado.

Se ha utiliza el paquete StandardScaler para estandarizar los datos en vez del estandarizado que se utilizaba en el archivo "kaggle.py". Además ahora todos los algoritmos utilizan los datos estandarizados. La transformación logarítmica de las variables con asimetría alta se hace con la función log1p para evitar problemas con 0 para algunos datos.

3.13.2. Breve descripción de los algoritmos utilizados.

Para obtener la solución final, se han utilizado los algoritmos XGBoost y LassoCV. El precio de una vivienda se calcula como la suma del 30 % del precio calculado por XGBoost y un 70 % del precio calculado por LassoCV.

3.13.3. Configuración de los parámetros de los algoritmos.

Los parámetros utilizados para obtener la solución son los mismos que se han utilizado para **Submission-30-12-4** y anteriores.

3.14. Submission-31-12-2

3.14.1. Breve descripción del preprocesado.

Es igual que para **Submission-31-12-1**.

3.14.2. Breve descripción de los algoritmos utilizados.

Para obtener la solución final, se han utilizado los algoritmos XGBoost y LassoCV. El precio de una vivienda se calcula como la suma de la solución de XGBoost y el 90 % de la solución de LassoCV divida entre 2.

3.14.3. Configuración de los parámetros de los algoritmos.

Los parámetros utilizados para obtener la solución son los mismos que se han utilizado para **Submission-30-12-4** y anteriores.

3.15. Submission-31-12-3

3.15.1. Breve descripción del preprocesado.

Es igual que para **Submission-31-12-1**.

3.15.2. Breve descripción de los algoritmos utilizados.

Para obtener la solución final, se han utilizado dos algoritmos, XGBoost y LassoCV. Para obtener la solución final, se calcula el precio de una vivienda como la suma de la mitad de cada uno de los precios calculados por los algoritmos anteriores.

3.15.3. Configuración de los parámetros de los algoritmos.

Los parámetros utilizados para obtener la solución son los mismos que se han utilizado para **Submission-30-12-4** y anteriores.

3.16. Submission-31-12-4

3.16.1. Breve descripción del preprocesado.

Es igual que para **Submission-31-12-1**.

3.16.2. Breve descripción de los algoritmos utilizados.

Para obtener las soluciones se utilizan los algoritmos LassoCV, XGBoost y ElasticNet. Para obtener el valor de una vivienda se suma el 30 % del precio calculado por XGBoost, un 40 % del precio calculado por LassoCV y el resto por el precio calculado por ElasticNet.

3.16.3. Configuración de los parámetros de los algoritmos.

Los parámetros utilizados para obtener la solución son los mismos que se han utilizado para **Submission-30-12-4** y anteriores. Para el algoritmo de ElasticNet se han utilizado los mismos parámetros que se utilizan en "kaggle.py"

3.17. Submission-31-12-5

3.17.1. Breve descripción del preprocesado.

Es igual que para **Submission-31-12-1**.

3.17.2. Breve descripción de los algoritmos utilizados.

Para obtener las soluciones se utilizan los algoritmos LassoCV, XGBoost y ElasticNet. Para obtener el valor de una vivienda se suma el 30 % del precio calculado por XGBoost, un 45 % del precio calculado por LassoCV y el resto por el precio calculado por ElasticNet.

3.17.3. Configuración de los parámetros de los algoritmos.

Los parámetros utilizados para obtener la solución son los mismos que se han utilizado para **Submission-30-12-4** y anteriores. Para el algoritmo de ElasticNet se han utilizado los mismos parámetros que se utilizan en "kaggle.py"

3.18. Submission-31-12-6

3.18.1. Breve descripción del preprocesado.

Es igual que para **Submission-31-12-1**.

3.18.2. Breve descripción de los algoritmos utilizados.

Para obtener las soluciones se utilizan los algoritmos LassoCV, XGBoost y ElasticNet. Para obtener el precio de una vivienda, se calcula la suma de los precios obtenidos por los algoritmos y se divide entre 3.

3.18.3. Configuración de los parámetros de los algoritmos.

Los parámetros utilizados para obtener la solución son los mismos que se han utilizado para **Submission-30-12-4** y anteriores. Para el algoritmo de ElasticNet se han utilizado los mismos parámetros que se utilizan en "kaggle.py"

4. Distribución de las soluciones entregadas.

Por cada una de las soluciones que se han comentado antes, se ha proporciona una carpeta que contiene el archivo submissionxxx.csv y el script de python que se ha utilizado para obtener la solución.

Para algunos casos para los cuáles dentro del script de python solamente cambia cómo se calcula la solución final, se ha guardado en un mismo directorio los 3 archivos .csv que se pueden obtener y el script de python. Dentro del script se puede descomentar y comentar las líneas que calculan las solución para obtener las diferentes soluciones si se desea.