Inteligencia de Negocio

Práctica 3: Competición en Kaggle

Grado Ing. Informática

Francisco Carrillo Pérez
Grupo 1 martes a las 09:30

Contents

List of Figures

List of Tables

1 Introducción

En este documento voy a presentar mi trabajo en el problema de regresión en precios de casas.

En la carpeta submissions se pueden encontrar todas las subidas que se han hecho a Kaggle ordenadas cronológicamente.

Hay una sección por cada subida que se ha hecho a Kaggle junto con una tabla de resultados que recoge de forma breve lo que se explica en cada sección, además de una sección exclusivamente para el script final que se utilizó.

2 21 diciembre 11:30

Como primer acercamiento al problema se decidió hacer un preprocesamiento básico a la hora de tratar con los datos. En primer lugar, se observó el número de valores perdidos que se encontraban en cada variable. Cómo se puede observar en la Figura 1. Las variables con más de un 50% de valores perdidos se decidió que sería mejor quitarlas que probar técnicas de tratamiento de valores perdidos, tanto del conjunto de entrenamiento como el de test.

Para observar como se distribuía la variable a predecir, en este caso SalePrice, se decidió pintarla. En la Figura 2 se puede observar la distribución de esta variable. Como se puede observar, la distribución no se encuentra centrada ni sigue una distribución normal. Para ello lo mejor es tomar el logaritmo de la variable, ya que esto ayudará al modelo a predecirla de una forma más efectiva. En la Figura 3 se puede observar como al tomar el logaritmo de la variable a predecir la variable sigue una distribución normal.

Se decidió hacer una matriz de correlación para observar cuáles eran las variables más correlacionadas con la variable a predecir. Para ello se creó un heatmap, como se puede observar en la Figura 4. De esta forma se decidió también eliminar las variables menos correlacionadas con la variable a predecir.

En cuanto a la tratamiento de los valores perdidos se decidió usar la media de la variable en el caso de las variables numéricas y la modo en el caso de las variables numéricas.

También se decidió crear una nueva variable total sqfootage (TotalSF) que junta el tamaño del bajo, de la primera planta y de la segunda planta. Comprobando la correlación tiene una correlación muy alta con respecto a la variable a predecir.

Con respecto a los algoritmos utilizados, se comenzaron usando dos algoritmos muy potentes como son el Gradient Boosting pero la implementación a mano de Microsoft LGBM la cuál destaca por su rápida ejecución, junto con el algoritmo de Random Forest para problemas de regresión. La elección de parámetros para ambos algoritmos se realizó con un RandomGridSearch de 10 iteraciones.

Para las predicciones, se uso la suma de los exponenciales de las predicciones divididas entre 2.

El código que se usó para las tareas anteriormente descritas se puede observar a continuación:

```
# import data
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')

# keep ID for submission
train_ID = train['Id']
test_ID = test['Id']

# split data for training
y_train = train['SalePrice']
```

```
X_train = train.drop(['Id','SalePrice'], axis=1)
X_test = test.drop('Id', axis=1)

# Log transformation of labels
y_train = np.log(y_train)

# I decided to get rid of features that have more than half of missing information or
    do not correlate to SalePrice
X_train.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)

# I decided to get rid of features that have more than half of missing information or
    do not correlate to SalePrice
X_test.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)

# add a new feature 'total sqfootage'
X_train['TotalSF'] = X_train['TotalBsmtSF'] + X_train['1stFlrSF'] + X_train['2ndFlrSF']
X_test['TotalSF'] = X_test['TotalBsmtSF'] + X_test['1stFlrSF'] + X_test['2ndFlrSF']
X_train.drop(['TotalBsmtSF', '1stFlrSF', '2ndFlrSF'], axis=1, inplace=True)
X_test.drop(['TotalBsmtSF', '1stFlrSF', '2ndFlrSF'], axis=1, inplace=True)

# MSZoning NA in pred. filling with most popular values
X_train['MSZoning'] = X_train['MSZoning'].fillna(X_train['MSZoning'].mode()[0])

# LotFrontage NA in all. I suppose NA means 0
X_train['LotFrontage'] = X_train['LotFrontage'].fillna(X_train['LotFrontage'].mean())

# Alley NA in all. NA means no access
X_train['Alley'] = X_train['Alley'].fillna('NOACCESS')

# MasVnrType NA in all. filling with most popular values
X_train['MasVnrType'] = X_train['MasVnrType'].fillna(X_train['MasVnrType'].mode()[0])

# BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2
# NA in all. NA means No basement
for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2'):
X_train[col] = X_train[col].fillna('NoBSMT')

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

# Missing values treatment

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

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

```
# FireplaceQu NA in all. NA means No Fireplace
X_train['FireplaceQu'] = X_train['FireplaceQu'].fillna('NoFP')

# GarageType, GarageFinish, GarageQual NA in all. NA means No Garage
for col in ('GarageType', 'GarageFinish', 'GarageQual'):
X_train[col] = X_train[col].fillna('NoGRG')

# GarageCars NA in pred. I suppose NA means 0
X_train['GarageCars'] = X_train['GarageCars'].fillna(0.0)

# SaleType NA in pred. filling with most popular values
X_train['SaleType'] = X_train['SaleType'].fillna(X_train['SaleType'].mode()[0])

# MSZoning NA in pred. filling with most popular values
X_test['MSZoning'] = X_test['MSZoning'].fillna(X_test['MSZoning'].mode()[0])

# LotFrontage NA in all. I suppose NA means 0
X_test['LotFrontage'] = X_test['LotFrontage'].fillna(X_test['LotFrontage'].mean())

# Alley NA in all. NA means no access
X_test['Alley'] = X_test['Alley'].fillna('NOACCESS')

# MasVnrType NA in all. filling with most popular values
X_test['MasVnrType'] = X_test['MasVnrType'].fillna(X_test['MasVnrType'].mode()[0])

# BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2
# NA in all. NA means No basement
for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2'):
X_test[col] = X_test[col].fillna('NoBSMT')

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

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

# KitchenQual NA in pred. filling with most popular values
X_test['KitchenQual'] = X_test['KitchenQual'].fillna(X_test['KitchenQual'].mode()[0])

# FireplaceQu NA in all. NA means No Fireplace
X_test['FireplaceQu'] = X_test['FireplaceQu'].fillna('NoFP')

# GarageType, GarageFinish, GarageQual NA in all. NA means No Garage
for col in ('GarageType', 'GarageFinish', 'GarageQual'):
X_test[col] = X_test[col].fillna('NoGRG')

# GarageCars NA in pred. I suppose NA means 0
X_test['GarageCars'] = X_test['GarageCars'].fillna(0.0)

# SaleType NA in pred. filling with most popular values
X_test['SaleType'] = X_test['SaleType'].fillna(X_test['SaleType'].mode()[0])

'''
LGBM
'''
gbm = lgb.LGBMRegressor(bagging_fraction=0.8, bagging_freq=5, bagging_seed=9,
boosting_type='gbdt', colsample_bytree=1.0, feature_fraction=0.2319,
feature_fraction_seed=9, learning_rate=0.01, max_bin=55,
max_depth=-1, min_child_samples=20, min_child_weight=0.001,
min_data_in_leaf=2, min_split_gain=0.0, min_sum_hessian_in_leaf=11,
n_estimators=1986, n_jobs=-1, num_leaves=5, objective='regression',
```

```
random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True,
subsample=1.0, subsample_for_bin=200000, subsample_freq=1,
verbose=0)

# specify parameters and distributions to sample from
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
param_dist = {"num_leaves": sp_randint(5,100),
"learning_rate": [0.01,0.5],
"min_data_in_leaf": sp_randint(1, 30),
"n_estimators": sp_randint(10, 2000)}
#run randomized search
n_iter_search = 10
random_search = RandomizedSearchCV(gbm, param_distributions=param_dist,
n_iter=n_iter_search)
random_search.fit(X_train,y_train)
print(random_search.best_estimator_)

'''
Random Forest Regressor
'''

RFR = ensemble.RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
    max_features='auto', max_leaf_nodes=400,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=5, min_samples_split=4,
min_weight_fraction_leaf=0.0, n_estimators=1347, n_jobs=1,
oob_score=False, random_state=None, verbose=0, warm_start=False)

# specify parameters and distributions to sample from
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
param_dist = {"max_leaf_nodes": sp_randint(30, 300),
"min_samples_leaf": sp_randint(1,100),
"min_samples_split": sp_randint(1, 30),
"n_estimators": sp_randint(100, 6000)}
#run randomized search
n_iter_search = 10
random_search = RandomizedSearchCV(RFR, param_distributions=param_dist,
n_iter=n_iter_search)
random_search.fit(X_train,y_train)
print(random_search.best_estimator_)

Final_labels = (np.exp(gbm.predict(X_test)) + np.exp(RFR.predict(X_test))) / 2
```

Figure 1: Cantidad de valores perdidos en cada variable

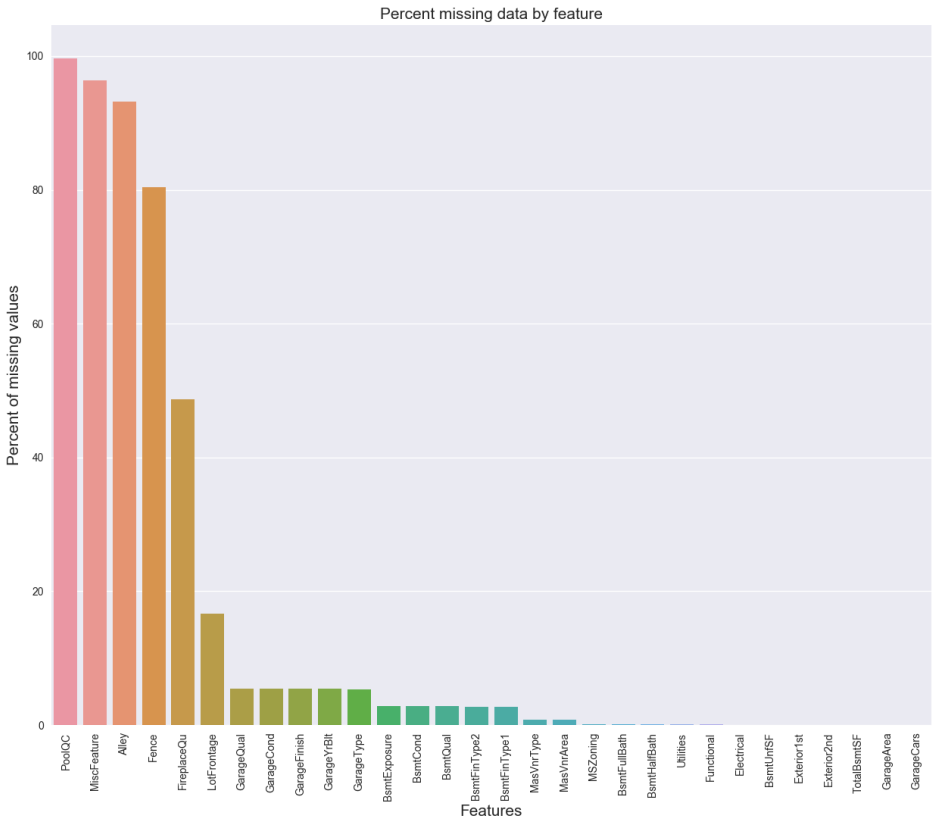


Figure 2: Distribución de la variable a predecir

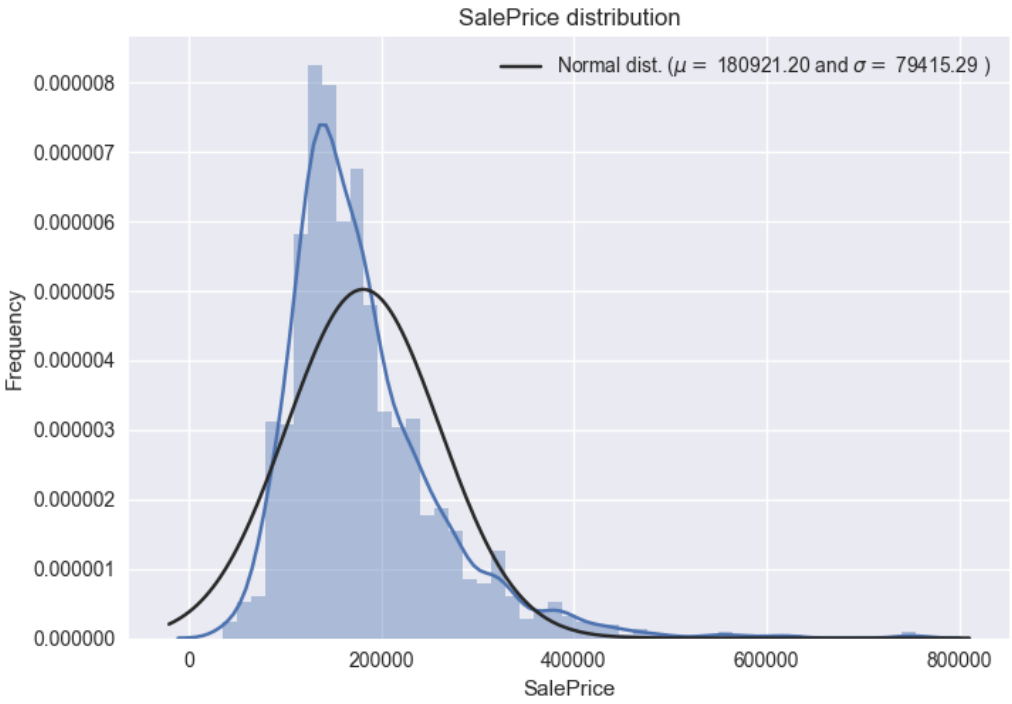


Figure 3: Distribución del logaritmo de la variable a predecir

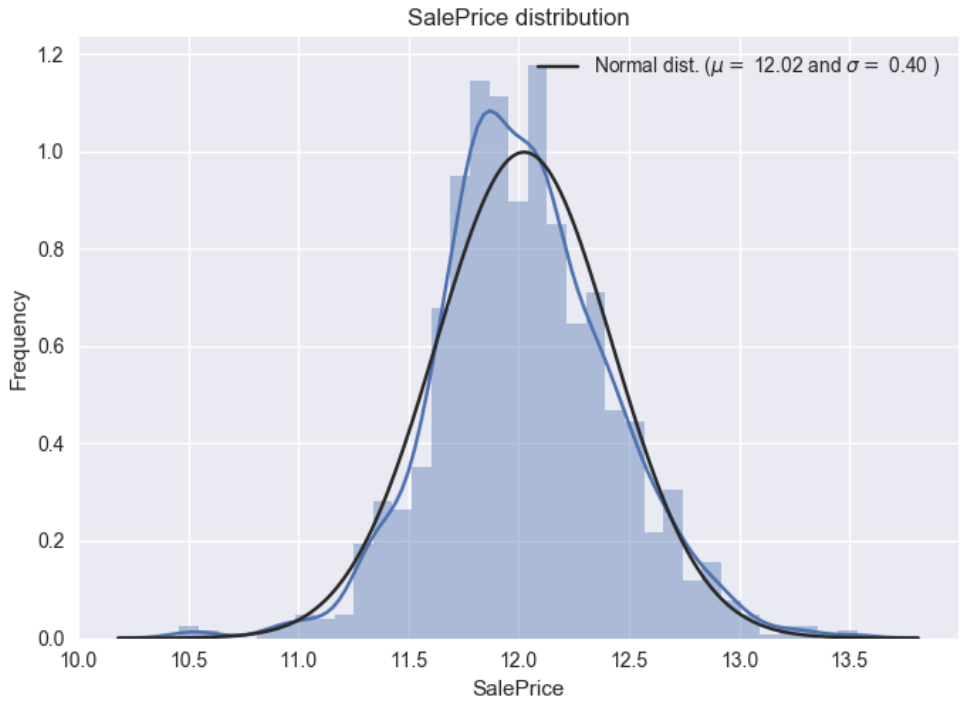
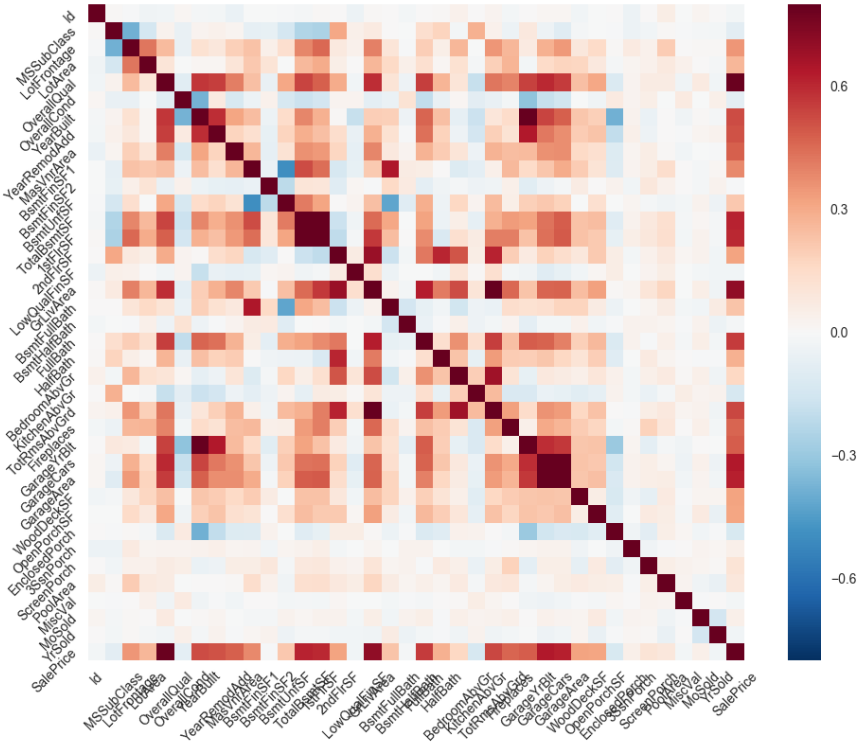


Figure 4: Heatmap en el conjunto de entrenamiento



3 22 diciembre 10:10

En este caso no realicé ningún cambio pero se me olvidó hacer el exponente de las predicciones y por eso el valor tan malo dentro del Leaderboard.

4 22 diciembre 10:40

Se decidió no eliminar las columnas con mayor porcentaje de valores perdidos ni aquellas variables menos correlacionadas con el valor a predecir, pero como se puede observar en la Tabla final, los resultados son inferiores a los anteriormente mejores obtenidos.

5 22 diciembre 17:41

Se volvieron a eliminar las variables con mayor porcentaje de valores perdidos y aquellas menos correlacionadas con el valor a predecir.

Se decidió prescindir del algoritmo de Random Forest, además de aumentar el rango de valores que se probaban con el RandomGridSearch. Se aumentó de forma significativa la eficacia del algoritmo de esta forma, como se puede observar en la tabla de resultados.

La configuración del algoritmo LGBM junto con el los parámetros de búsqueda del RandomGridSearch pueden ser observador a continuación:

```
gbm = lgb.LGBMRegressor(bagging_fraction=0.8, bagging_freq=5, bagging_seed=9,
boosting_type='gbdt', colsample_bytree=1.0, feature_fraction=0.2319,
feature_fraction_seed=9, learning_rate=0.01, max_bin=55,
max_depth=-1, min_child_samples=20, min_child_weight=0.001,
```

```
min_data_in_leaf=2, min_split_gain=0.0, min_sum_hessian_in_leaf=11,
n_estimators=3862, n_jobs=-1, num_leaves=5, objective='regression',
random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True,
subsample=1.0, subsample_for_bin=200000, subsample_freq=1,
verbose=0)

from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV

param_dist = {"num_leaves": sp_randint(5,100),
"learning_rate": [0.01,0.02,0.03,0.1,0.15,0.2,0.3,0.4,0.5],
"min_data_in_leaf": sp_randint(1, 30),
"n_estimators": sp_randint(10, 4000)}
#run randomized search
n_iter_search = 10
random_search = RandomizedSearchCV(gbm, param_distributions=param_dist,
n_iter=n_iter_search)
random_search.fit(X_train,y_train)
print(random_search.best_estimator_)
```

6 24 diciembre 11:18

Se aumentó el rango de búsqueda del RandomGridSearch, pero los resultados que se obtuvieron con los nuevos parámetros fueron inferiores a los anteriormente obtenidos, por lo que se decidió continuar con los anteriores.

La configuración del algoritmo LGBM es:

```
gbm = lgb.LGBMRegressor(bagging_fraction=0.8, bagging_freq=5, bagging_seed=9,
boosting_type='gbdt', colsample_bytree=1.0, feature_fraction=0.2319,
feature_fraction_seed=9, learning_rate=0.03, max_bin=55,
max_depth=-1, min_child_samples=20, min_child_weight=0.001,
min_data_in_leaf=6, min_split_gain=0.0, min_sum_hessian_in_leaf=11,
n_estimators=274, n_jobs=-1, num_leaves=52, objective='regression',
random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True,
subsample=1.0, subsample_for_bin=200000, subsample_freq=1,
verbose=0)

from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV

param_dist = {"num_leaves": sp_randint(5,100),
"learning_rate": [0.01,0.02,0.03,0.1,0.15,0.2,0.3,0.4,0.5],
"min_data_in_leaf": sp_randint(1, 30),
"n_estimators": sp_randint(10, 7000)}
#run randomized search
n_iter_search = 10
random_search = RandomizedSearchCV(gbm, param_distributions=param_dist,
n_iter=n_iter_search)
random_search.fit(X_train,y_train)
print(random_search.best_estimator_)
```

7 25 diciembre 13:38

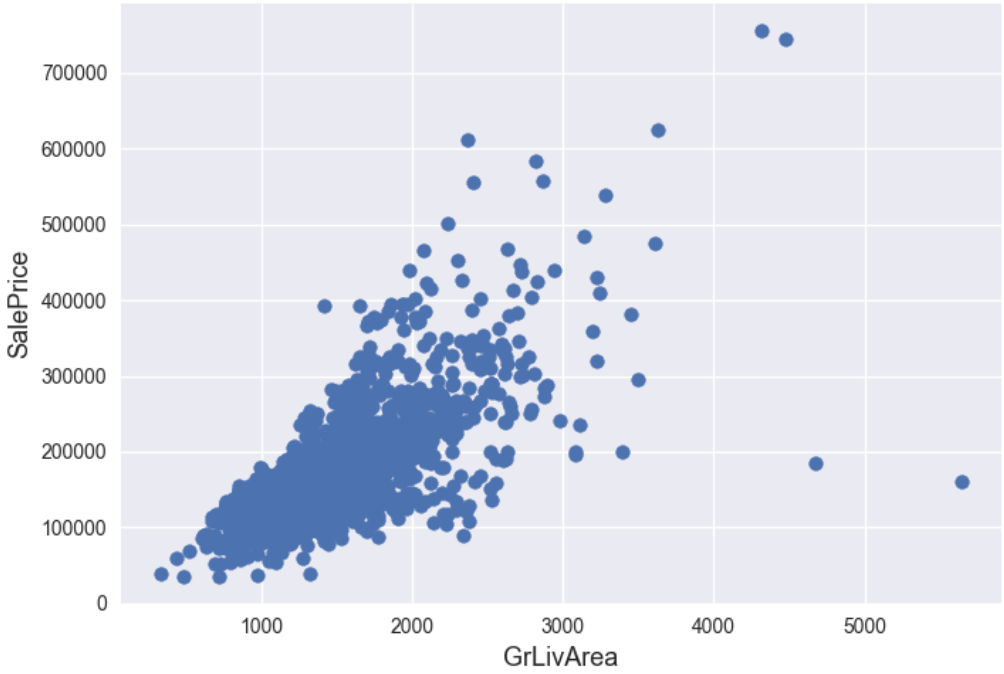
Ya habiendo encontrado unos parámetros que parecían buenos se decidió continuar con el preprocesamiento de los datos. Otro tema que aún no se había atajado era el tratamiento de los outliers.

Para observar los outliers se decidió comenzar a estudiar las variables más significativas. Una de la que se obtenía bastante comprensión es de la variable de GrLivArea. Como se puede observar

en la Figura 5, hay puntos cuya GrLivArea es muy grande pero que su precio es bastante bajo. Esto puede ser debido a que son casas muy viejas o que necesitan mucha reforma. Se decidió quitarlos ya que podían afectar a la hora de que nuestro modelo hiciese predicciones.

```
train = train.drop(train[(train['GrLivArea']>4000) &
    (train['SalePrice']<1000000)].index)
```

Figure 5: Outliers en la variable GrLivArea



8 25 diciembre 13:53

En este caso se decidió parar de trabajar en el procesamiento y se pensó en usar la técnica de stacking, ya que se pensaba que podría dar buenos resultados. Para ello se usó un stacking en el que se utilizaba la media de las predicciones de ambos podemos. Se creaba un meta-modelo que contenía los dos modelos que se deseaban y se entrenaban ambos, después la predicción se realizaba haciendo la media de las predicciones de ambos algoritmos.

El código es el siguiente:

```
from sklearn.base import BaseEstimator, TransformerMixin, RegressorMixin, clone
class AveragingModels(BaseEstimator, RegressorMixin, TransformerMixin):
   def __init__(self, models):
      self.models = models

   # we define clones of the original models to fit the data in
   def fit(self, X, y):
      self.models_ = [clone(x) for x in self.models]

      # Train cloned base models
```

```
        for model in self.models_:
        model.fit(X, y)

        return self

    #Now we do the predictions for cloned models and average them
    def predict(self, X):
        predictions = np.column_stack([
        model.predict(X) for model in self.models_
        ])
        return np.mean(predictions, axis=1)

averaged_models = AveragingModels(models = (gbm,ridge_model))
```

9 25 diciembre 20:49

En este caso se intentó hacer más complejo el modelo predictivo. Para ello se entrenó un algoritmo LassoCV. Los parámetros que se pusieron se eligieron mirando los kernerls de otros participantes de Kaggle. Luego se le asignaron unos pesos a cada modelo a la hora de hacer la predicción.

El algoritmo de LassoCV y el RandomGridSearch se puede observar a continuación:

```
# LASSOCV
from sklearn.linear_model import LassoCV
LassoCV = LassoCV(alphas = [1, 0.1, 0.001, 0.0005])
scores = cross_val_score(LassoCV, X_train, y_train,cv=5)
print("Accuracy LassoCV: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

Además, se observó en otros kernels de Kaggle que realizando una correción del skew, o asimetría estadística en castellano, se obtenían buenos resultados. Es por esto que se comprobó la asimetría estadística de las variables numéricas y aquellos que tenían un valor superior a 0.75, se corregían con la función boxcox1p de la librería scipy. El código es el siguiente:

```
from scipy.stats import norm, skew #for some statistics
numeric_feats = X_train.dtypes[X_train.dtypes != "object"].index
# Check the skew of all numerical features
skewed_feats = X_train[numeric_feats].apply(lambda x:
    skew(x.dropna())).sort_values(ascending=False)
skewness = pd.DataFrame({'Skew' :skewed_feats})
skewness = skewness[abs(skewness) > 0.75]

from scipy.special import boxcox1p
skewed_features = skewness.index
lam = 0.15
for feat in skewed_features:
     X_train[feat] = boxcox1p(X_train[feat], lam)
```

10 25 diciembre 20:58

Al observar los malos resultados que se obtuvieron en 9, se probó a eliminar el modelo de LassoCV. De esta forma se mejoró el resultado anterior pero seguía siendo muy inferior a lo mejor que se había obtenido.

11 26 diciembre 20:25

Al ver que corregir el skew en este caso hacía que los resultados no mejorasen se decidió prescindir de él. Además, se decidió crear una nueva variable que fue OverallGrade:

```
X_train["OverallGrade"] = X_train["OverallCond"] + X_train["BsmtCond"]
```

12 27 diciembre 13:56

Se volvió a intentar mejorar el modelo realizando un stack también de GradientBoostingRegressor y de Ridge Regression sumado al de LGBM con Ridge Regression, pero los resultados no fueron mejores. El código:

```
'''
Gradient Boosting
'''
from sklearn.ensemble import GradientBoostingRegressor
GBest = GradientBoostingRegressor(n_estimators=3000, learning_rate=0.05, max_depth=3,
    max_features='sqrt',
min_samples_leaf=15, min_samples_split=10, loss='huber')
averaged_models2 = AveragingModels(models = (GBest,ridge_model))
```

13 27 diciembre 16:32

En este caso se decidió crear nuevas variables que fuesen polinomios de variables existentes. Generando polinomios de variables existentes que estuviesen correlacionadas con el objetivo se pensaba que ayudarían al algoritmo. El código sería el siguiente:

```
X_train['TotalSF2'] = X_train['TotalSF']**2
X_test['TotalSF2'] = X_test['TotalSF]**2

X_train['TotalSF3'] = X_train['TotalSF']**3
X_test['TotalSF3'] = X_test['TotalSF]**3

X_train['OverallCond2'] = X_train['OverallCond']**2
X_test['OverallCond2'] = X_test['OverallCond]**2

X_train['OverallCond3'] = X_train['OverallCond']**3
X_test['OverallCond3'] = X_test['OverallCond]**3
```

14 27 diciembre 16:36

Al no conseguir mejorar el resultado, se probó a volver al modelo anterior, dónde las predicciones se hacían con un stacking de LGBM y Ridge Regression más un LGBM solo para las predicciones. Para ello se usaron dos pesos al hacer las predicciones, uno de 0.8 para el stacking y otro de 0.2 para el LGBM solo.

15 27 diciembre 16:41

Como prueba, se cambiaron los pesos a 0.7 para el stacking y 0.3 para LGBM para ver si se obtenían mejores resultados, pero se empeoró.

16 28 diciembre 10:59

Cómo no se observaba una mejora con la creación de las nuevas variables, excepto con la variable TotalSF, se decidió eliminar todas las variables que se habían creado excepto esta misma, manteniendo el mismo modelo que se utilizaba en 14.

17 28 diciembre 11:13

Se decidió probar a eliminar las variables menos relacionadas con el objetivo, pero esto llevó a un peor resultado que manteniendo estas variables.

18 30 diciembre 17:38

Me encontraba en un punto que no sabía cómo podía mejorar los resultados que estaba obteniendo. Me puse a buscar kernels dentro de la competición que me ayudasen a tener ideas y decidí comenzar

desde 0 con el preprocesamiento, cogiendo distintas ideas de distintos kernels.

Para ello que realicé era muy parecido a lo que estaba realizando hasta el momento pero con algunos cambios. Se volvió a hacer el logaritmo de la variable a predecir. Se eliminaron las variables que tenían muchos valores perdidos. Además, se eliminaron algunas variables cuyo significado no me parecía muy intuitivo. A las variables numéricas se les corrigió el skew, además de escalarlas a media cero y varianza unidad. En cuanto al tratamiento de los valores perdidos, se simplificó y se rellenaban los valores perdidos con el valor más frecuente de la columna. Además, se eliminaban outliers como anteriormente.

El código es el siguiente:

```
#outliers treatment
train = train.drop(train[(train['GrLivArea']>4000) &
    (train['SalePrice']<1000000)].index)
y = train.SalePrice
y = np.log1p(y)
train.drop('SalePrice',1,inplace=True)
all_data = pd.concat([train,test])

##drop attributes with many missing values
#too many missing values
all_data.drop("Alley", 1,inplace=True)
all_data.drop("Fence", 1,inplace=True)
all_data.drop("MiscFeature", 1,inplace=True)
all_data.drop("PoolQC", 1,inplace=True)
all_data.drop("FireplaceQu", 1,inplace=True)
#non-intuitive features
#all_data.drop("GarageArea", 1,inplace=True)
all_data.drop("MSSubClass", 1,inplace=True)
all_data.drop("GarageYrBlt", 1,inplace=True)
all_data.drop("RoofMatl", 1,inplace=True)
#convert year to age
Yr = max(all_data['YearBuilt'])
all_data['BuildingAge'] = all_data['YearBuilt'].apply(lambda x: Yr-x if not
    pd.isnull(x) else 'None')
all_data['RemodelAge'] = all_data['YearRemodAdd'].apply(lambda x: Yr-x if not
    pd.isnull(x) else 'None')
all_data['SellAge'] = all_data['YrSold'].apply(lambda x: Yr-x if not pd.isnull(x) else
    'None')
#drop old variables
all_data.drop("YearBuilt", 1,inplace=True);
all_data.drop("RemodelAge", 1,inplace=True);
all_data.drop("YrSold", 1,inplace=True);
# add a new feature 'total sqfootage'
all_data['TotalSF'] = all_data['TotalBsmtSF'] + all_data['1stFlrSF'] +
    all_data['2ndFlrSF']
all_data.drop(['1stFlrSF', '2ndFlrSF'], axis=1, inplace=True)

#extraxt train and test datasets
train = all_data[:train.shape[0]]
test = all_data[train.shape[0]:]

#divide features into numeric and categorical
numeric = [c for c in train.columns if train.dtypes[c] != 'object']
numeric.remove('Id')
print("Number of Numeric Attributes = %s" % (len(numeric)))
categorical = [c for c in train.columns if train.dtypes[c] == 'object']
print("Number of Categorical Attributes = %s" % (len(categorical)))
```

```
##preprocessing of numeric features
feature = train[numeric].dropna()
skewed_feats = feature.apply(skew) #compute skewness
skewed_feats = skewed_feats[(skewed_feats > 0.5) | (skewed_feats < -0.5)]
skewed_feats = skewed_feats.index

#log transformation to resolve skewness on train and test data
all_data[skewed_feats] = np.log1p(all_data[skewed_feats])

pd.options.mode.chained_assignment = None # default='warn'
#standard scale (zero mean and unit variance)
all_data[numeric] = all_data[numeric].fillna(all_data[numeric].mean())
x_train = all_data[:train.shape[0]]
x_test = all_data[train.shape[0]:]
columns=x_train.columns
x_train[numeric] = StandardScaler().fit_transform(x_train[numeric])
x_train = pd.DataFrame(x_train, index=x_train.index, columns=x_train.columns)
x_test[numeric] = StandardScaler().fit_transform(x_test[numeric])
x_test = pd.DataFrame(x_test, index=x_test.index, columns=x_test.columns)
all_data = pd.concat([x_train,x_test])


##preprocessing of categorical features
feature = all_data[categorical]
#identify the most frequent level of each categorical feature
frequentLevel = feature.apply(lambda x: x.value_counts().idxmax())
def itemReplace (column,value):
frequentLevel[column] = value
itemReplace('BsmtFinSF1','None')
itemReplace('BsmtFinSF2','None')
itemReplace('GarageType','None')
itemReplace('GarageFinish','None')
itemReplace('GarageQual','None')
itemReplace('GarageCond','None')
all_data[categorical] = feature.fillna(frequentLevel)
'
```

Además, para la predicción solo se utilizó el algoritmo LGBM, con los parámetros definidos en 5.

19 30 diciembre 18:05

Se añadió el algoritmo Elastic Net para realizar la predicción, mejorando el resultado que se obtenía utilizando solo el LGBM, añadiendo unos pesos de 0.7 para el LGBM y 0.3 para el Elastic Net.

20 30 diciembre 18:22

No se había creado la variable TotalSF que tanta correlación tenía con el valor a predecir, por lo que se volvió a crear y se entrenaron los modelos igual que anteriormente 19.

21 2 enero 15:21

Se decidió probar a modificar el modelo dándole un peso de 0.8 a las predicciones del LGBM ya que obtenía mejor RMSE solo y 0.2 para el Elastic Net, pero el resultado conjunto fue peor, por lo que se puede afirmar que seguramente el algoritmo LGBM estaba haciendo un poco de overfitting sobre el conjunto de entrenamiento.

22 4 enero 17:35

Para continuar con el preprocesamiento se deseaba comprobar si existían más outliers. Para ello se pintaron otras variables como LotArea,LotFrontage y TotalBsmtSF.

En la Figura 6 se puede observar como hay unos valores que parecen outliers cuando LotArea vale más de 100000 y el SalePrice es menor a 500000.

En la Figura 7 se puede observar como hay unos valores que parecen outliers cuando LotFrontage vale más de 300 y el SalePrice es menor a 300000.

En la Figura 8 se puede observar como hay unos valores que parecen outliers cuando TotalBsmtSF vale más de 5000 y el SalePrice es menor a 200000.

Por lo tanto se decidió eliminar estos puntos usando el código siguiente:

```
train = train.drop(train[(train['LotArea']>100000) &
    (train['SalePrice']<500000)].index)
train = train.drop(train[(train['LotFrontage']>300) &
    (train['SalePrice']<300000)].index)
train = train.drop(train[(train['TotalBsmtSF']>5000) &
    (train['SalePrice']<200000)].index)
```

Figure 6: Outliers en la variable LotArea

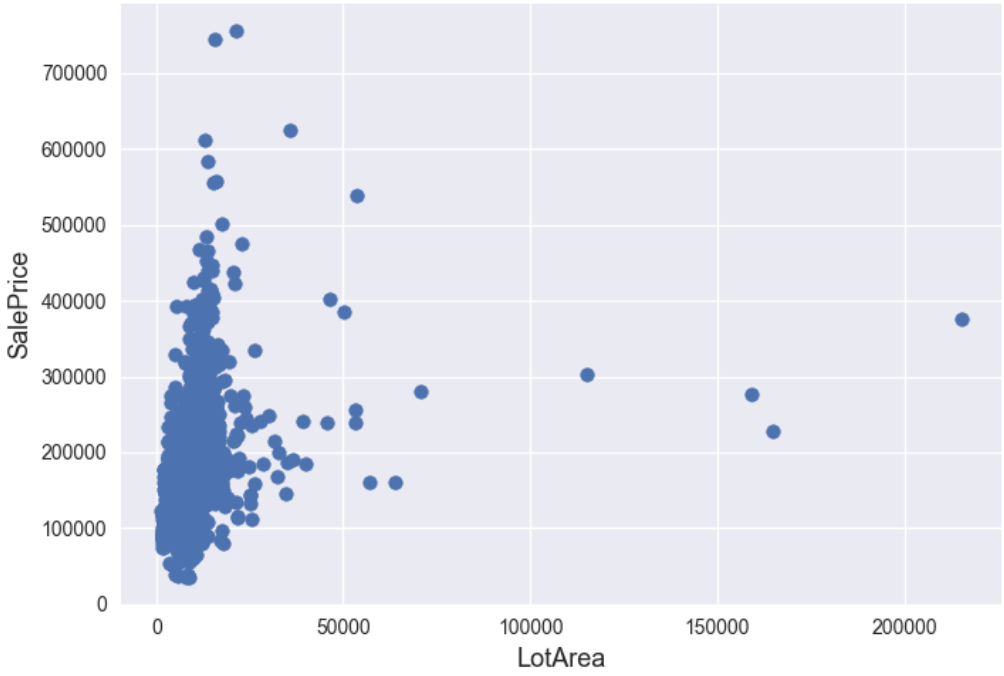


Figure 7: Outliers en la variable LotFrontage

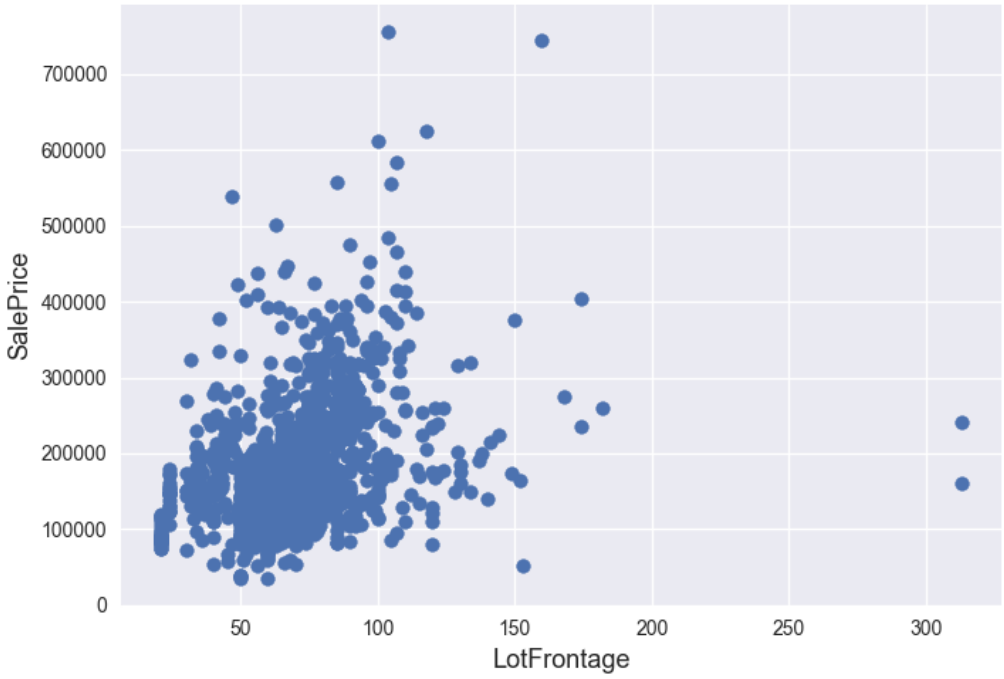
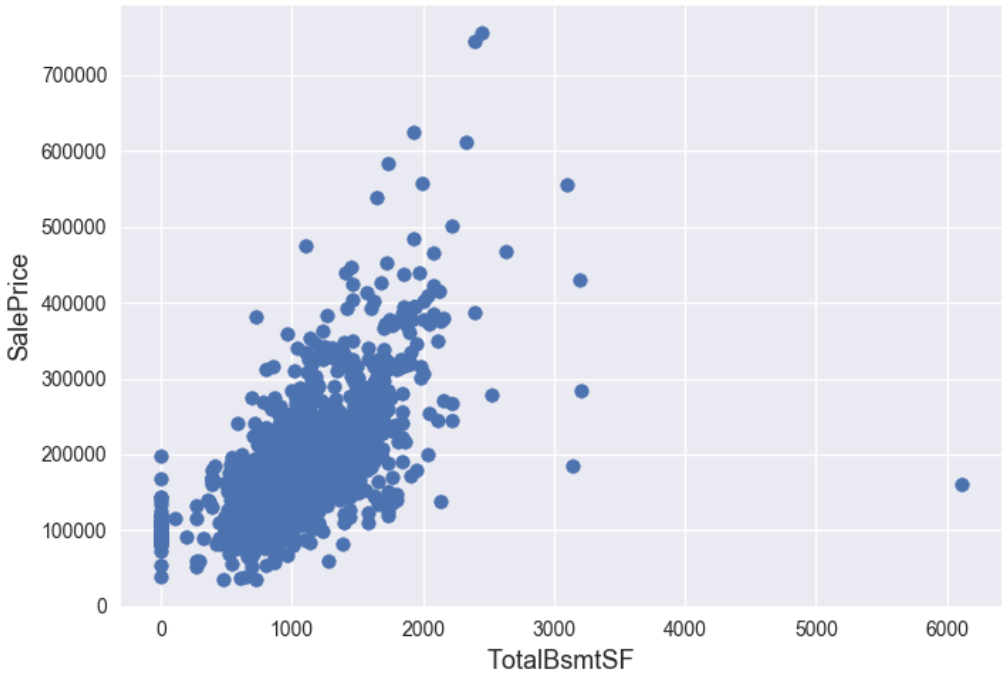


Figure 8: Outliers en la variable TotalBsmtSF



23 4 enero 17:53

Se decidió crear una nueva variable LivArea_Total, de la siguiente forma:

```
all_data['LivArea_Total'] = all_data['GrLivArea'] + all_data['GarageArea'] +
    all_data['PoolArea']
```

El funcionamiento fue un poco inferior a anteriormente.

24 4 enero 18:01

Se probó a usar para predecir solo el modelo de LGBM porque daba mejor resultado en el RMSE, pero parece que estaba haciendo overfitting porque dio peor resultado que el modelo compuesto por LGBM y Elastic Net.

25 4 enero 18:43

Probando a jugar con los pesos se decidió darle el mismo peso a las predicciones del LGBM como al del Elastic Net y se observó que el desempeño era mucho mejor.

26 4 enero 19:05

Se decidió cambiar los parámetros del algoritmo de Elastic Net para comprobar si se podía conseguir una mejora, pero no se mejoró. Los parámetros eran los siguientes:

```
alpha=0.00004, l1_ratio=1.2
```

27 4 enero 19:11

Cómo último paso, se decidió volver al anterior tratamiento de los valores perdidos que se realizaba en 2. Esto mejoró levente el resultado del modelo.

28 Script final utilizado

```
'''
Fracisco Carrillo Perez
'''
import numpy as np
import pandas as pd
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from scipy.stats import skew
import scipy.stats as st
from math import sqrt
import os

from six.moves import cPickle as pickle
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.pylab import rcParams
import seaborn as sns

from sklearn import datasets, linear_model
from sklearn.linear_model import Ridge, RidgeCV, ElasticNet, LassoCV, Lasso
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn import cross_validation, metrics
from sklearn.cross_validation import KFold
from sklearn.metrics import mean_squared_error
import xgboost as xgb
```

```
#data_root = '../input' # Change me to store data elsewhere

#reading the train and test datasets
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")

# keep ID for submission
train_ID = train['Id']
test_ID = test['Id']

#-------------------------------------------------
trainSize = train.shape #size of the train dataset
testSize = test.shape #size of the test dataset
print("train dataset size = %s x %s" % (trainSize))
print("test dataset size = %s x %s" % (testSize))

#outliers treatment
train = train.drop(train[(train['GrLivArea']>4000) &
    (train['SalePrice']<1000000)].index)
train = train.drop(train[(train['LotArea']>100000) &
    (train['SalePrice']<500000)].index)
train = train.drop(train[(train['LotFrontage']>300) &
    (train['SalePrice']<300000)].index)
train = train.drop(train[(train['TotalBsmtSF']>5000) &
    (train['SalePrice']<200000)].index) # parecen ser el mismo

y = train.SalePrice
y = np.log1p(y)
train.drop('SalePrice',1,inplace=True)
all_data = pd.concat([train,test])

##drop attributes with many missing values
#too many missing values
all_data.drop("Alley", 1,inplace=True)
all_data.drop("Fence", 1,inplace=True)
all_data.drop("MiscFeature", 1,inplace=True)
all_data.drop("PoolQC", 1,inplace=True)
all_data.drop("FireplaceQu", 1,inplace=True)
#non-intuitive features
#all_data.drop("GarageArea", 1,inplace=True)
all_data.drop("MSSubClass", 1,inplace=True)
all_data.drop("GarageYrBlt", 1,inplace=True)
all_data.drop("RoofMatl", 1,inplace=True)
#convert year to age
Yr = max(all_data['YearBuilt'])
all_data['BuildingAge'] = all_data['YearBuilt'].apply(lambda x: Yr-x if not
    pd.isnull(x) else 'None')
all_data['RemodelAge'] = all_data['YearRemodAdd'].apply(lambda x: Yr-x if not
    pd.isnull(x) else 'None')
all_data['SellAge'] = all_data['YrSold'].apply(lambda x: Yr-x if not pd.isnull(x) else
    'None')
#drop old variables
all_data.drop("YearBuilt", 1,inplace=True);
all_data.drop("RemodelAge", 1,inplace=True);
all_data.drop("YrSold", 1,inplace=True);
# add a new feature 'total sqfootage'
all_data['TotalSF'] = all_data['TotalBsmtSF'] + all_data['1stFlrSF'] +
    all_data['2ndFlrSF']
all_data.drop(['1stFlrSF', '2ndFlrSF'], axis=1, inplace=True)

#extraxt train and test datasets
```

```
train = all_data[:train.shape[0]]
test = all_data[train.shape[0]:]

#divide features into numeric and categorical
numeric = [c for c in train.columns if train.dtypes[c] != 'object']
numeric.remove('Id')
print("Number of Numeric Attributes = %s" % (len(numeric)))
categorical = [c for c in train.columns if train.dtypes[c] == 'object']
print("Number of Categorical Attributes = %s" % (len(categorical)))

##preprocessing of numeric features
feature = train[numeric].dropna()
skewed_feats = feature.apply(skew) #compute skewness
skewed_feats = skewed_feats[(skewed_feats > 0.5) | (skewed_feats < -0.5)]
skewed_feats = skewed_feats.index

#log transformation to resolve skewness on train and test data
all_data[skewed_feats] = np.log1p(all_data[skewed_feats])

pd.options.mode.chained_assignment = None # default='warn'
#standard scale (zero mean and unit variance)
all_data[numeric] = all_data[numeric].fillna(all_data[numeric].mean())
x_train = all_data[:train.shape[0]]
x_test = all_data[train.shape[0]:]
columns=x_train.columns
x_train[numeric] = StandardScaler().fit_transform(x_train[numeric])
x_train = pd.DataFrame(x_train, index=x_train.index, columns=x_train.columns)
x_test[numeric] = StandardScaler().fit_transform(x_test[numeric])
x_test = pd.DataFrame(x_test, index=x_test.index, columns=x_test.columns)
all_data = pd.concat([x_train,x_test])

'''
##preprocessing of categorical features
feature = all_data[categorical]
#identify the most frequent level of each categorical feature
frequentLevel = feature.apply(lambda x: x.value_counts().idxmax())
def itemReplace (column,value):
frequentLevel[column] = value
itemReplace('BsmtFinSF1','None')
itemReplace('BsmtFinSF2','None')
itemReplace('GarageType','None')
itemReplace('GarageFinish','None')
itemReplace('GarageQual','None')
itemReplace('GarageCond','None')
all_data[categorical] = feature.fillna(frequentLevel)
'''
# MSZoning NA in pred. filling with most popular values
all_data['MSZoning'] = all_data['MSZoning'].fillna(all_data['MSZoning'].mode()[0])

# LotFrontage NA in all. I suppose NA means 0
all_data['LotFrontage'] =
    all_data['LotFrontage'].fillna(all_data['LotFrontage'].mean())

# MasVnrType NA in all. filling with most popular values
all_data['MasVnrType'] =
    all_data['MasVnrType'].fillna(all_data['MasVnrType'].mode()[0])

# BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2
# NA in all. NA means No basement
for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2'):
all_data[col] = all_data[col].fillna('NoBSMT')
```

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

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

# KitchenQual NA in pred. filling with most popular values
all_data['KitchenQual'] =
    all_data['KitchenQual'].fillna(all_data['KitchenQual'].mode()[0])

# GarageType, GarageFinish, GarageQual NA in all. NA means No Garage
for col in ('GarageType', 'GarageFinish', 'GarageQual'):
all_data[col] = all_data[col].fillna('NoGRG')

# GarageCars NA in pred. I suppose NA means 0
all_data['GarageCars'] = all_data['GarageCars'].fillna(0.0)

# SaleType NA in pred. filling with most popular values
all_data['SaleType'] = all_data['SaleType'].fillna(all_data['SaleType'].mode()[0])
#transform categorical variables into dummy variables
all_data = pd.get_dummies(all_data)

# create new variable LivArea_Total
all_data['LivArea_Total'] = all_data['GrLivArea'] + all_data['GarageArea'] +
    all_data['PoolArea']

x_train = all_data[:train.shape[0]]
x_test = all_data[train.shape[0]:]
columns=x_train.columns
print(x_train.shape)
print(x_test.shape)

x_train.drop("Id", 1,inplace=True)
x_test.drop("Id", 1,inplace=True)
names = list(x_train)

x_train = np.array(x_train)
x_test = np.array(x_test)
y_train = np.array(y)
ntrain = x_train.shape[0]
ntest = x_test.shape[0]

from sklearn.linear_model import ElasticNet

'''
LGBM
'''

import lightgbm as lgb
from sklearn.model_selection import KFold, cross_val_score, train_test_split

gbm = lgb.LGBMRegressor(bagging_fraction=0.8, bagging_freq=5, bagging_seed=9,
boosting_type='gbdt', colsample_bytree=1.0, feature_fraction=0.2319,
feature_fraction_seed=9, learning_rate=0.01, max_bin=55,
max_depth=-1, min_child_samples=20, min_child_weight=0.001,
min_data_in_leaf=2, min_split_gain=0.0, min_sum_hessian_in_leaf=11,
n_estimators=3862, n_jobs=-1, num_leaves=5, objective='regression',
random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True,
subsample=1.0, subsample_for_bin=200000, subsample_freq=1,
```

```
verbose=0)

'''
# specify parameters and distributions to sample from
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
param_dist = {"num_leaves": sp_randint(5,1000),
"learning_rate": [0.01,0.02,0.03,0.05,0.1,0.15,0.2,0.3,0.4,0.5,0.6,0.7,0.8],
"min_data_in_leaf": sp_randint(1, 50),
"n_estimators": sp_randint(10, 8000)}
#run randomized search
n_iter_search = 20
random_search = RandomizedSearchCV(gbm, param_distributions=param_dist,
n_iter=n_iter_search)
random_search.fit(x_train,y_train)
print(random_search.best_estimator_)
'''

# Average R2 score and standart deviation of 5-fold cross-validation
scores = cross_val_score(gbm, x_train, y_train, cv=5)
print("Accuracy GBM: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
# Prints R2 and RMSE scores
def get_score(prediction, lables):
from sklearn.metrics import r2_score, mean_squared_error
print('R2: {}'.format(r2_score(prediction, lables)))
print('RMSE: {}'.format(np.sqrt(mean_squared_error(prediction, lables))))

gbm = gbm.fit(x_train,y_train)
predictions_gbm = gbm.predict(x_train)
get_score(predictions_gbm,y_train)

elesnet = ElasticNet(alpha=0.0004, l1_ratio=1.2)

'''
# specify parameters and distributions to sample from
from scipy.stats import randint as sp_randint
from numpy.random import random_sample
from sklearn.model_selection import RandomizedSearchCV
param_dist = {"alpha": random_sample(20),
"l1_ratio": sp_randint(1.0,6.0)}
#run randomized search
n_iter_search = 20
random_search = RandomizedSearchCV(elesnet, param_distributions=param_dist,
n_iter=n_iter_search)
random_search.fit(x_train,y_train)
print(random_search.best_estimator_)
'''
# Average R2 score and standart deviation of 5-fold cross-validation
scores = cross_val_score(elesnet, x_train, y_train, cv=5)
print("Accuracy elesnet: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

elesnet = elesnet.fit(x_train,y_train)
predictions_elesnet = elesnet.predict(x_train)
get_score(predictions_elesnet,y_train)

predictions = predictions_gbm*0.5 + predictions_elesnet*0.5
get_score(predictions,y_train)

'''
#Ridge Regularization model
'''
```

```
'''
# fit a ridge regularization model
from sklearn.linear_model import Ridge
ridge_model = Ridge(alpha=7.0)
scores = cross_val_score(ridge_model, x_train, y_train,cv=5)
print("Accuracy Ridge: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

ridge_model = ridge_model.fit(x_train,y_train)
predictions_ridge_model = ridge_model.predict(x_train)
get_score(predictions_ridge_model,y_train)

predictions = predictions_gbm*0.8 + predictions_elesnet*0.2
get_score(predictions,y_train)

from sklearn.base import BaseEstimator, TransformerMixin, RegressorMixin, clone
class AveragingModels(BaseEstimator, RegressorMixin, TransformerMixin):
def __init__(self, models):
self.models = models

# we define clones of the original models to fit the data in
def fit(self, X, y):
self.models_ = [clone(x) for x in self.models]

# Train cloned base models
for model in self.models_:
model.fit(X, y)

return self

#Now we do the predictions for cloned models and average them
def predict(self, X):
predictions = np.column_stack([
model.predict(X) for model in self.models_
])
return np.mean(predictions, axis=1)

averaged_models = AveragingModels(models = (elesnet,gbm))

# Average R2 score and standart deviation of 5-fold cross-validation
scores = cross_val_score(averaged_models, x_train, y_train, cv=5)
print("Accuracy Averaged: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

averaged_models = AveragingModels(models = (elesnet,gbm))
averaged_models = averaged_models.fit(x_train,y_train)
predictions_averaged = averaged_models.predict(x_train)
predictions = predictions_averaged*0.7 + predictions_elesnet*0.3
get_score(predictions,y_train)
'''
gbm = lgb.LGBMRegressor(bagging_fraction=0.8, bagging_freq=5, bagging_seed=9,
boosting_type='gbdt', colsample_bytree=1.0, feature_fraction=0.2319,
feature_fraction_seed=9, learning_rate=0.01, max_bin=55,
max_depth=-1, min_child_samples=20, min_child_weight=0.001,
min_data_in_leaf=2, min_split_gain=0.0, min_sum_hessian_in_leaf=11,
n_estimators=3862, n_jobs=-1, num_leaves=5, objective='regression',
random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True,
subsample=1.0, subsample_for_bin=200000, subsample_freq=1,
verbose=0)

elesnet = ElasticNet(alpha=0.0004, l1_ratio=1.2)
gbm = gbm.fit(x_train,y_train)
elesnet = elesnet.fit(x_train,y_train)
```

```
#Final_labels = np.exp(gbm.predict(x_test))
Final_labels = np.exp(gbm.predict(x_test))*0.5 + np.exp(elesnet.predict(x_test))*0.5
pd.DataFrame({'Id': test_ID, 'SalePrice': Final_labels}).to_csv('submission.csv',
    index =False)
```

29 Tabla de resultados

Table 1: Tabla de resultados con las distintas soluciones subidas a Kaggle

Fecha y Hora	**Posición**	**Score**	**RMSLE**	**Descripción Preproceso**	**Descripción Algoritmo**	**Configuración**
21 Dic. 11:30 2	1425	0.13712	0.10688	-String a categórico -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiendo de valores perdidos. - Creación de nueva variable	- Light Gradient Boosting Machines. - Random Forest Regressor.	Usado un RandomGridSearch para los parámetros de ambos algoritmos 5
22 Dic. 10:10 3	Inferior a la anterior-mente mejor	0.72509	0.13542	El mismo que anteriormente. Pero al hacer la predicción se me olvidó hacer el exponente.	- Light Gradient Boosting Machines. - Random Forest Regressor.	Usado un RandomGridSearch para los parámetros de ambos algoritmos 5
22 Dic. 10:40 4	Inferior a la anterior-mente mejor	0.14489	0.11545	El mismo que anteriormente. Pero eliminando el quitar las columnas con muchos valores perdidos	- Light Gradient Boosting Machines. - Random Forest Regressor.	Usado un RandomGridSearch para los parámetros de ambos algoritmos 5
22 Dic. 17:41 5	858	0.12375	0.08053	-String a categórico -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiendo de valores perdidos. - Creación de nueva variable	Light Gradient Boosting Machines	Usado un RandomGridSearch para los parámetros del algoritmo pero aumentando el rango de búsqueda

Fecha y Hora	Posición	Score	RMSLE	Descripción Preproceso	Descripción Algoritmo	Configuración
24 Dic. 11:18 6	Inferior a la anterior-mente mejor	0.12964	0.055334	-String a categórico -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiendo de valores perdidos. - Creación de nueva variable	Light Gradient Boosting Machines	Usado un RandomGridSearch para los parámetros del algoritmo pero aumentando el rango de búsqueda
25 Dic. 13:38 7	Inferior a la anterior-mente mejor	0.12403	0.07735	-Eliminación de outliers -String a categórico -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiendo de valores perdidos. - Creación de nueva variable	Light Gradient Boosting Machines	Usado un RandomGridSearch para los parámetros del algoritmo
25 Dic. 13:53 8	791	0.12250	0.088229	-Eliminación de outliers -String a categórico -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiendo de valores perdidos. - Creación de nueva variable	Stacking de Light Gradient Boosting Machines y de Ridge Regression - LGBM solo	Usado un RandomGridSearch para los parámetros de ambos algoritmos

Fecha y Hora	Posición	Score	RMSLE	Descripción Preproceso	Descripción Algoritmo	Configuración
25 Dic. 20:49 9	Inferior a la anterior-mente mejor	0.25856	0.099147	-Eliminación de outliers -String a categórico -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiendo de valores perdidos. - Creación de nueva variable - Skew de variables numéricas	Stacking de Light Gradient Boosting Machines y de Ridge Regression - LGBM solo -LassoCV solo	Usado un RandomGridSearch para los parámetros de ambos algoritmos
25 Dic. 20:58 10	Inferior a la anterior-mente mejor	0.32109	0.08828	-Eliminación de outliers -String a categórico -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiendo de valores perdidos. - Creación de nueva variable - Skew de variables numéricas	Stacking de Light Gradient Boosting Machines y de Ridge Regression - LGBM solo	Usado un RandomGridSearch para los parámetros de ambos algoritmos

Fecha y Hora	Posición	Score	RMSLE	Descripción Preproceso	Descripción Algoritmo	Configuración
26 Dic. 20:2511	Inferior a la anterior-mente mejor	0.12388	0.08810	-Eliminación de outliers -String a categórico -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiendo de valores perdidos. - Creación de nuevas variables	Stacking de Light Gradient Boosting Machines y de Ridge Regression - LGBM solo	Usado un RandomGridSearch para los parámetros de ambos algoritmos
27 Dic. 13:5612	Inferior a la anterior-mente mejor	0.12393	0.08416	-Eliminación de outliers -String a categórico -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiendo de valores perdidos. - Creación de nuevas variables	Stacking de Light Gradient Boosting Machines y de Ridge Regression - Stacking de Gradient Boosting y de Ridge Regression	Usado un RandomGridSearch para los parámetros de ambos algoritmos

Fecha y Hora	Posición	Score	RMSLE	Descripción Preproceso	Descripción Algoritmo	Configuración
27 Dic. 16:32 13	Inferior a la anterior-mente mejor	0.12400	0.08280	-Eliminación de outliers -String a categórico -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiendo de valores perdidos. - Creación de nuevas variables -Polinomios de variables existentes	Stacking de Light Gradient Boosting Machines y de Ridge Regression - Stacking de Gradient Boosting y de Ridge Regression	Usado un RandomGridSearch para los parámetros de ambos algoritmos
27 Dic. 16:3614	Inferior a la anterior-mente mejor	0.12276	0.08139	-Eliminación de outliers -String a categórico -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiendo de valores perdidos. - Creación de nuevas variables -Polinomios de variables existentes	Stacking de Light Gradient Boosting Machines y de Ridge Regression - LGBM solo	Usado un RandomGridSearch para los parámetros de ambos algoritmos y pesos

Fecha y Hora	Posición	Score	RMSLE	Descripción Preproceso	Descripción Algoritmo	Configuración
27 Dic. 16:4115	Inferior a la anterior-mente mejor	0.12400	0.08280	-Eliminación de outliers -String a categórico -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiendo de valores perdidos. - Creación de nuevas variables -Polinomios de variables existentes	Stacking de Light Gradient Boosting Machines y de Ridge Regression - LGBM solo	Usado un RandomGridSearch para los parámetros de ambos algoritmos y cambiando pesos
28 Dic. 10:5916	Inferior a la anterior-mente mejor	0.12278	0.08467	-Eliminación de outliers -String a categórico -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiendo de valores perdidos.	Stacking de Light Gradient Boosting Machines y de Ridge Regression - LGBM solo	Usado un RandomGridSearch para los parámetros de ambos algoritmos y cambiando pesos

Fecha y Hora	**Posición**	**Score**	**RMSLE**	**Descripción Preproceso**	**Descripción Algoritmo**	**Configuración**
28 Dic. 11:1317	Inferior a la anterior-mente mejor	0.12884	0.08999	-Eliminación de outliers -String a categórico -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiendo de valores perdidos. -Eliminación de variables menos correlacionadas	Stacking de Light Gradient Boosting Machines y de Ridge Regression - LGBM solo	Usado un RandomGridSearch para los parámetros de ambos algoritmos
30 Dic. 17:3818	647	0.12073	0.071028	-Eliminación de outliers -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiendo de valores perdidos. -Skew de valores numéricos. -Media cero y varianza unidad .	Light Gradient Boosting Machines	Usado un RandomGridSearch para los parámetros

Fecha y Hora	Posición	Score	RMSLE	Descripción Preproceso	Descripción Algoritmo	Configuración
30 Dic. 18:0519	471	0.11812	0.074678	-Eliminación de outliers -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiendo de valores perdidos. -Skew de valores numéricos. -Media cero y varianza unidad .	Light Gradient Boosting Machines y Elastic Net	Usado un RandomGridSearch para los parámetros y pesos para la predicción
30 Dic. 18:2220	378	0.11663	0.07475	-Eliminación de outliers -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiendo de valores perdidos. -Skew de valores numéricos. -Media cero y varianza unidad .- Creación nueva variable	Light Gradient Boosting Machines y Elastic Net	Usado un RandomGridSearch para los parámetros y pesos para la predicción

Fecha y Hora	**Posición**	**Score**	**RMSLE**	**Descripción Preproceso**	**Descripción Algoritmo**	**Configuración**
2 Ene. 15:2121	Inferior a la anterior-mente mejor	0.11753	0.074756	-Eliminación de outliers -String a categórico -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiendo de valores perdidos.	Light Gradient Boosting Machines y Elastic Net	Usado un RandomGridSearch para los parámetros y cambiados pesos para la predicción
4 Ene. 17:3522	233	0.11521	0.074556	-Más eliminación de outliers -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiendo de valores perdidos. -Skew de valores numéricos. -Media cero y varianza unidad .- Creación nueva variable	Light Gradient Boosting Machines y Elastic Net	Usado un RandomGridSearch para los parámetros y cambiados pesos para la predicción

Fecha y Hora	Posición	Score	RMSLE	Descripción Preproceso	Descripción Algoritmo	Configuración
4 Ene. 17:5323	Inferior a la anterior-mente mejor	0.11566	0.076028	-Más eliminación de outliers -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiento de valores perdidos. -Skew de valores numéricos. -Media cero y varianza unidad .- Creación nuevas variable	Light Gradient Boosting Machines y Elastic Net	Usado un RandomGridSearch para los parámetros y cambiados pesos para la predicción
4 Ene. 18:0124	Inferior a la anterior-mente mejor	0.11876	0.06984	-Más eliminación de outliers -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiento de valores perdidos. -Skew de valores numéricos. -Media cero y varianza unidad .- Creación nuevas variable	Light Gradient Boosting Machines	Usado un RandomGridSearch para los parámetros

Fecha y Hora	Posición	Score	RMSLE	Descripción Preproceso	Descripción Algoritmo	Configuración
4 Ene. 18:4325	188	0.11488	0.081496	-Más eliminación de outliers -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiento de valores perdidos. -Skew de valores numéricos. -Media cero y varianza unidad .- Creación nuevas variable	Light Gradient Boosting Machines y Elastic Net	Usado un RandomGridSearch para los parámetros y cambiados pesos para la predicción
4 Ene. 19:0526	Inferior a la anterior-mente mejor	0.11548	0.0775511	-Más eliminación de outliers -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiento de valores perdidos. -Skew de valores numéricos. -Media cero y varianza unidad .- Creación nuevas variable	Light Gradient Boosting Machines y Elastic Net	Usado un RandomGridSearch para los parámetros

Fecha y Hora	Posición	Score	RMSLE	Descripción Preproceso	Descripción Algoritmo	Configuración
4 Ene. 19:1127	181	0.11485	0.081659	-Más eliminación de outliers -Transformación logarítmica al valor a predecir -Quitar columnas con muchos valores perdidos. - Tratamiento de valores perdidos. -Skew de valores numéricos. -Media cero y varianza unidad .- Creación nuevas variable	Light Gradient Boosting Machines y Elastic Net	Usado un RandomGridSearch para los parámetros

30 Contenido Adicional

31 Bibliografía
References