### 方法设计与实现

Kaggle题目：<https://www.kaggle.com/c/house-prices-advanced-regression-techniques>

1. 读取csv文件，并将特征值和标签分开储存。

train = pd.read\_csv('./input/train.csv')

test = pd.read\_csv('./input/test.csv')

train\_labels = train.pop('SalePrice')

features = pd.concat([train, test], *keys*=['train', 'test'])

1. 删除多余特征，仅保留与SsalePrice相关的特征，其中被删除的特征如下：数据值超过一半为NA，且与SsalePrice无关。

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)

1. 对剩余特征做数据填补以及格式转换，其中以类型转换、均值填充、补零为主。

*# MSSubClass as str*

features['MSSubClass'] = features['MSSubClass'].astype(str)

*# MSZoning NA in pred. filling with most popular values*

features['MSZoning'] = features['MSZoning'].fillna(features['MSZoning'].mode()[0])

*# LotFrontage  NA in all. I suppose NA means 0*

features['LotFrontage'] = features['LotFrontage'].fillna(features['LotFrontage'].mean())

*# Alley  NA in all. NA means no access*

features['Alley'] = features['Alley'].fillna('NOACCESS')

*# Converting OverallCond to str*

features.OverallCond = features.OverallCond.astype(str)

*# MasVnrType NA in all. filling with most popular values*

features['MasVnrType'] = features['MasVnrType'].fillna(features['MasVnrType'].mode()[0])

*# BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2*

*# NA in all. NA means No basement*

for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2'):

    features[col] = features[col].fillna('NoBSMT')

*# 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])

*# KitchenAbvGr to categorical*

features['KitchenAbvGr'] = features['KitchenAbvGr'].astype(str)

*# KitchenQual NA in pred. filling with most popular values*

features['KitchenQual'] = features['KitchenQual'].fillna(features['KitchenQual'].mode()[0])

*# FireplaceQu  NA in all. NA means No Fireplace*

features['FireplaceQu'] = features['FireplaceQu'].fillna('NoFP')

*# GarageType, GarageFinish, GarageQual  NA in all. NA means No Garage*

for col in ('GarageType', 'GarageFinish', 'GarageQual'):

    features[col] = features[col].fillna('NoGRG')

*# GarageCars  NA in pred. I suppose NA means 0*

features['GarageCars'] = features['GarageCars'].fillna(0.0)

*# SaleType NA in pred. filling with most popular values*

features['SaleType'] = features['SaleType'].fillna(features['SaleType'].mode()[0])

*# Year and Month to categorical*

features['YrSold'] = features['YrSold'].astype(str)

features['MoSold'] = features['MoSold'].astype(str)

*# Adding total sqfootage feature and removing Basement, 1st and 2nd floor features*

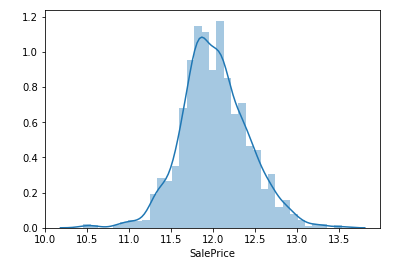
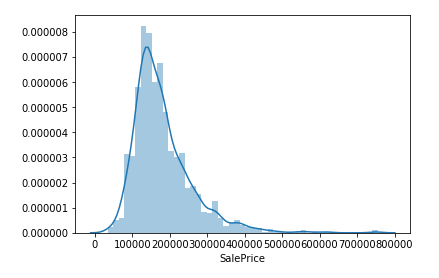
features['TotalSF'] = features['TotalBsmtSF'] + features['1stFlrSF'] + features['2ndFlrSF']

features.drop(['TotalBsmtSF', '1stFlrSF', '2ndFlrSF'], *axis*=1, *inplace*=True)

1. 对SsalePrice标签进行log取值，减小计算量，同时让数据分布接近正态分布。

train\_labels = np.log(train\_labels)

SsalePrice取对数前后的数值分布对比：

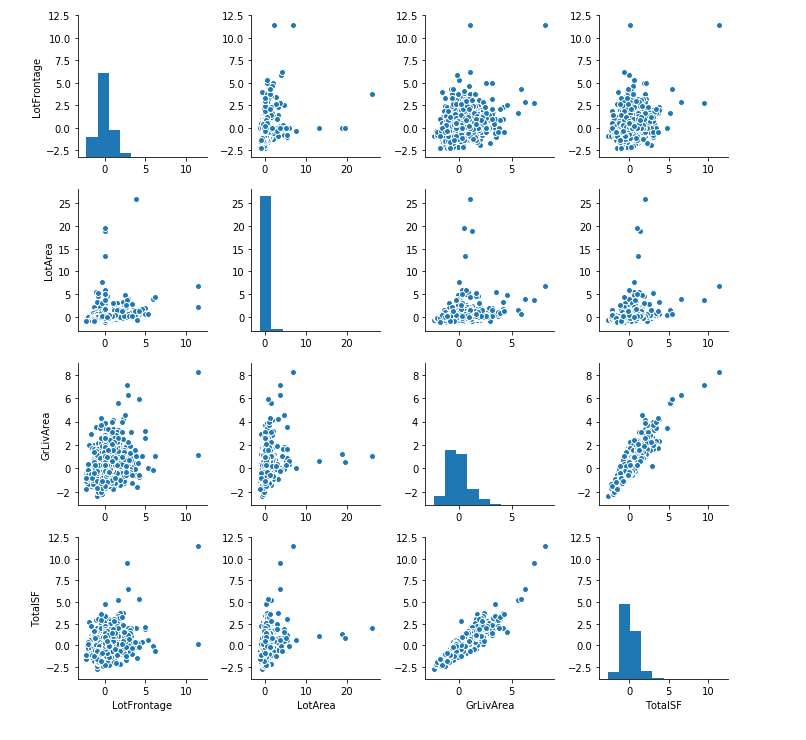


1. 将特征值为数字类型的特征进行标准化处理。

numeric\_features = features.loc[:,['LotFrontage', 'LotArea', 'GrLivArea', 'TotalSF']]

numeric\_features\_standardized = (numeric\_features - numeric\_features.mean())/numeric\_features.std()

四个特征标准化之后的结果如下图所示：



1. 对训练集中的数据进行划分，分别为训练集和验证集。

*### Splitting features*

train\_features = features.loc['train'].drop('Id', *axis*=1).select\_dtypes(*include*=[np.number]).values

test\_features = features.loc['test'].drop('Id', *axis*=1).select\_dtypes(*include*=[np.number]).values

*### Splitting standardized features*

train\_features\_st = features\_standardized.loc['train'].drop('Id', *axis*=1).select\_dtypes(*include*=[np.number]).values

test\_features\_st = features\_standardized.loc['test'].drop('Id', *axis*=1).select\_dtypes(*include*=[np.number]).values

*# ### Splitting to train and validation sets*

*### Shuffling train sets*

train\_features\_st, train\_features, train\_labels = shuffle(train\_features\_st, train\_features, train\_labels, *random\_state* = 5)

*### Splitting*

x\_train, x\_test, y\_train, y\_test = train\_test\_split(train\_features, train\_labels, *test\_size*=0.1, *random\_state*=200)

x\_train\_st, x\_test\_st, y\_train\_st, y\_test\_st = train\_test\_split(train\_features\_st, train\_labels, *test\_size*=0.1, *random\_state*=200)

1. 对结果进行验证，使用R2 决定系数作为评价指标，以及使用5折交叉验证对训练集的结果进行准确率的统计。

**ElasticNetCV方法：**

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)

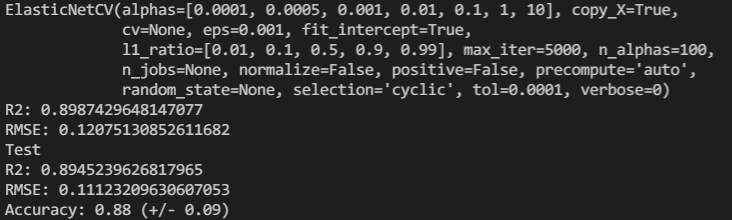
train\_test(ENSTest, x\_train\_st, x\_test\_st, y\_train\_st, y\_test\_st)

*# Average R2 score and standart deviation of 5-fold cross-validation*

scores = cross\_val\_score(ENSTest, train\_features\_st, train\_labels, *cv*=5)

print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() \* 2))

**结果：**



**Gradient Boosting方法：**

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)

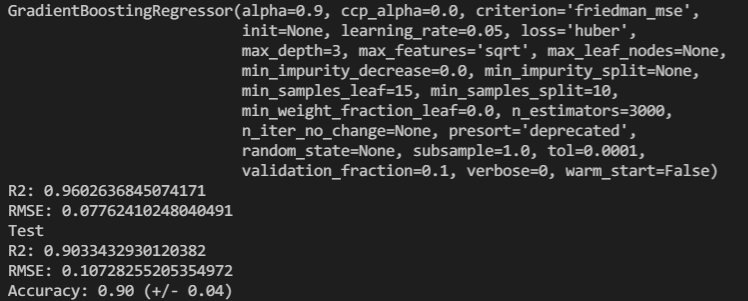
train\_test(GBest, x\_train, x\_test, y\_train, y\_test)

*# Average R2 score and standart deviation of 5-fold cross-validation*

scores = cross\_val\_score(GBest, train\_features\_st, train\_labels, *cv*=5)

print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() \* 2))

**结果：**



1. 将预测结果写入本地，其中最终结果取Gradient Boosting和Elastic Net两种方法所预测结果的均值。

*# Retraining models*

GB\_model = GBest.fit(train\_features, train\_labels)

ENST\_model = ENSTest.fit(train\_features\_st, train\_labels)

*## Getting our SalePrice estimation*

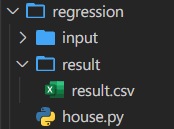
Final\_labels = (np.exp(GB\_model.predict(test\_features)) + np.exp(ENST\_model.predict(test\_features\_st))) / 2

*## Saving to CSV*

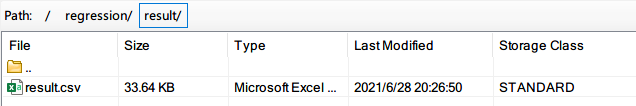
pd.DataFrame({'Id': test.Id, 'SalePrice': Final\_labels}).to\_csv('./result/result.csv', *index* =False)

### 将预测结果写入S3：

服务器文件：



S3文件：



由于提前在服务器端运行了s3桶挂载的脚本，因此只需将预测结果写在服务器相应的文件夹即可同步到S3的桶内。