《数据挖掘导论》实验五

Kaggle 实战：房价预测

November 20, 2020

Contents

1 简介 **2**
- 1.1 实验目的 . . . 2
- 1.2 实验内容 . . . 2

2 数据集 **2**
- 2.1 数据集简介 . . . 2
- 2.2 评价指标 . . . 3
- 2.3 数据集读取 . . . 3

3 特征工程 **4**
- 3.1 异常值检测 . . . 5
- 3.2 数据变换 . . . 6
- 3.3 缺失值处理 . . . 8
- 3.4 相关性分析 . . . 10
- 3.5 特征组合 . . . 10
- 3.6 特征编码 . . . 10
- 3.7 特征降维 . . . 13

4 机器学习模型 **14**
- 4.1 回归模型 . . . 15
- 4.2 集成方法 . . . 16
  - 4.2.1 Bagging . . . 16
  - 4.2.2 Boosting . . . 17
  - 4.2.3 Stacking . . . 18

5 提交结果 **20**

6 参考资源 **22**

1 简介

1.1 实验目的

- 了解 Kaggle 平台。
- 熟悉特征工程的基本步骤，掌握基本的数据分析和数据处理手段。
- 能熟练使用经典的机器学习算法解决实际问题，并在此基础上使用集成方法提升模型性能。

1.2 实验内容

- 加载数据集，对数据进行分析。
- 对数据进行特征工程。
- 使用基本的机器学习算法来完成任务。
- 使用集成学习方法进一步提升性能。
- 在 Kaggle 上提交模型结果。

2 数据集

2.1 数据集简介

Kaggle是一个著名的供机器学习爱好者交流的平台，房价预测（House Prices）是 Kaggle 社区上一个非常经典而有趣的实际问题，我们的任务是基于竞赛方所提供的爱荷华州埃姆斯的住宅数据信息，预测每间房屋的销售价格。官方提供的数据集包含以下文件：

- train.csv：训练集，其包含 1460 间房屋的信息，每间房屋包含 80 个特征，并且每间房屋有对应的售出价格。
- test.csv：测试集，包含了 1459 待预测房屋的信息，每间房屋包含 80 个特征。
- data_description.txt：该文件对房屋的所有特征及其可能的取值进行了描述。
- sample_submission.csv：一个提交示例，这也是我们最终需要整理成的数据格式，包含了测试集样本的 id 及其对应的预测价格 SalePrice，以便提交。

这里我们列举一下几个关键特征作为说明，更多的信息可以在 data_description.txt 中找到。

- SalePrice：房屋的售价，这也是需要我们预测的目标变量，它是连续的。
- LotArea：房屋的地段面积，以平方英尺为单位，它也是连续的。
- MSZoning：房屋所在的区域类型，它有如下的可能取值。
  - A：Agriculture，农业区
  - C：Commercial，商业区
  - FV：Floating Village Residential，流动村住宅
  - I：Industrial，工业区
  - RH：Residential High Density，高密度住宅区
  - RP：Residential Low Density，低密度住宅区
  - RM：Residential Low Density Park ，低密度住宅公园
  - RM：Residential Medium Density，中密度住宅区

小结 可以看到，房屋的特征多达 80 个，每个特征下面也有很多属性，基本涵盖了我们买卖房屋所考虑的因素，但某些特征只对部分样本起作用。如 PoolArea（泳池面积）和 PoolQC（泳池品质）只针对有泳池的房屋样本，而没有配备泳池的样本在该特征和属性下为缺失值。

虽然特征很多，但相比之下，有些特征显得冗余。如 TotalBsmtSF（地下室的面积）、1stFlrSF（一楼的面积）、2stFlrSF（二楼的面积）。直观地我们可以用一个新的特征 TotalBS（总面积）来表示三者求和，诸如此类。

通过观察数据集，我们意识到在这个案例中，我们将会耗费较多的精力来进行数据处理。

2.2 评价指标

这个项目要预测的目标变量是连续值，其是一个典型的回归任务，其采用的评价指标是均方根误差（Root Mean Square Error, RMSE），其计算方式如下

$$\sqrt{\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N}} \tag{1}$$

其中 y_i 是真实的房屋价格，$\hat{y}_i$ 是预测的房屋价格，N 为样本数，因此 RMSE 值越小，说明模型的性能越好。

2.3 数据集读取

首先，我们导入加载数据所需要用到的库文件。

```
#import some necessary librairies
from scipy import stats
from scipy.stats import norm, skew
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
color = sns.color_palette()
sns.set_style('darkgrid')
```

我们使用 pandas 的 read_csv 来加载数据集，其会将数据转换为 dataframe 格式。

```
train = pd.read_csv('input/train.csv')
test = pd.read_csv('input/test.csv')
```

然后我们查看一下数据的格式。

```
print(type(train))
# <class 'pandas.core.frame.DataFrame'>
print(train.shape)
# (1460, 81)
print(test.shape)
# (1459, 80)
train.head()
```

这同我们之前讨论的一致，训练集可以看作一个 1460×81 的矩阵，测试集可以看作 1459×80 的矩阵。矩阵的第一维是样本，第二维是特征，训练集相比测试集多了一维 SalePrice。房屋的特征繁多，包含连续和离散变量，且每个特征下有不同的取值并存在缺失值 NaN。因此，我们要必要在训练模型前对特征进一步加工，即**特征工程**。

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	Mo
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	

5 rows × 81 columns

3 特征工程

让我们首先对特征工程有一个整体的认识。

Reference：机器学习中，有哪些特征选择的工程方法？

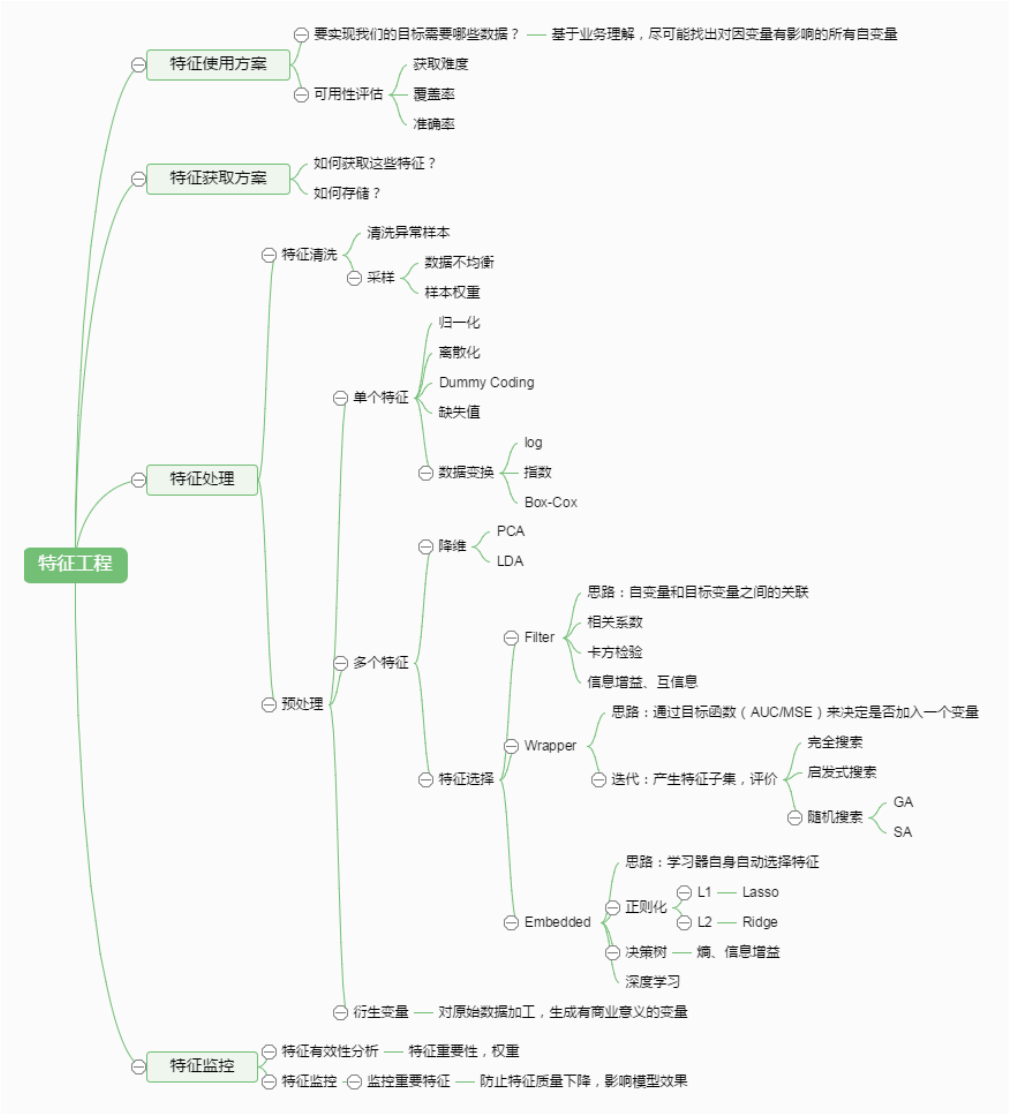
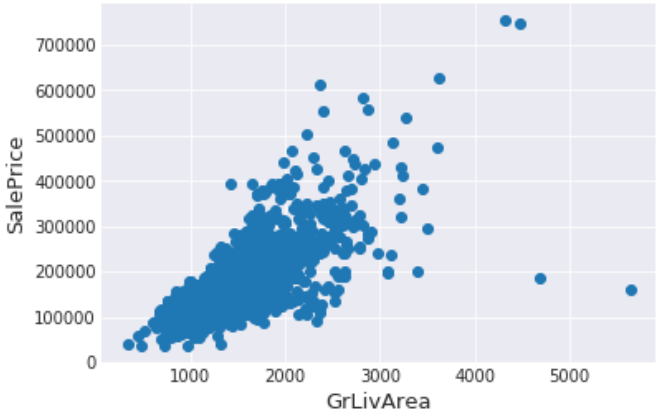


Figure 1: 特征工程总览

3.1 异常值检测

异常值（Outliers）检验是数据预处理中非常重要的一环，这里我们先做出房屋面积（GrLivArea）和其对应价格（SalePrice）的散点图，以便观察。

```
fig, ax = plt.subplots()
ax.scatter(x = train['GrLivArea'], y = train['SalePrice'])
plt.ylabel('SalePrice', fontsize=13)
plt.xlabel('GrLivArea', fontsize=13)
plt.show()
```



我们可以发现，右下角存在两个明显的异常点。即房屋面积很大的情况下，对应的房屋价格却非常低廉，这不符我们的常理，也会对模型的学习有很大的影响，因此我们将他们移除。

```
#Deleting outliers
train = train.drop(train[(train['GrLivArea']>4000) & (train['SalePrice']<300000)].index)

#Check the graphic again
fig, ax = plt.subplots()
ax.scatter(train['GrLivArea'], train['SalePrice'])
plt.ylabel('SalePrice', fontsize=13)
plt.xlabel('GrLivArea', fontsize=13)
plt.show()
```

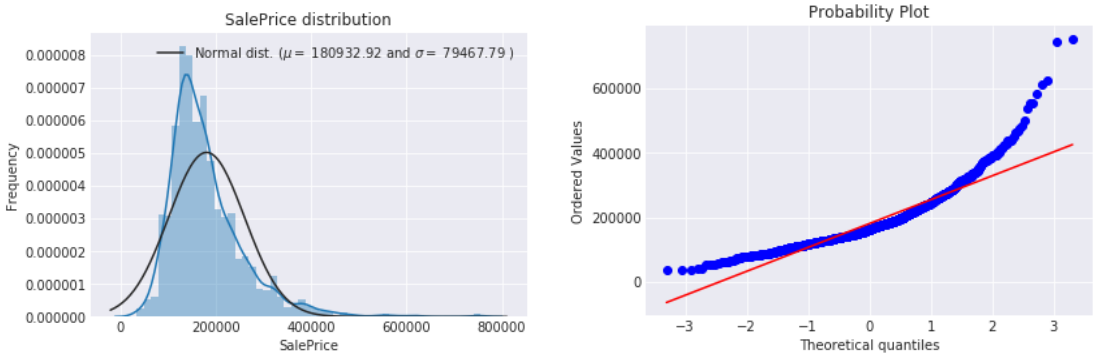
现在，我们已将这两个异常值清理掉了。

小结 有时候移除异常值并不总是可靠的，这里我们移除这两个异常点是因为它们的存在相当不合理。如果我们将训练集中的异常值完全移除，那么模型将难以应对测试集存在的异常值。此外，异常值的存在在某种程度上也使得模型更具鲁棒性（robustness）。

3.2 数据变换

由于房价 SalePrice 是我们需要预测的变量，现在我们对其作进一步的分析。我们首先作出它的分布曲线并绘制其分位图（QQ Plot——Quantile-Quantile Plot，如果点近似落在 $y = x$ 上，说明该分布与正态分布相似）。

```
# Get the fitted parameters used by the function
(mu, sigma) = norm.fit(train['SalePrice'])
print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))
#    mu = 12.02 and sigma = 0.40

#Now plot the distribution
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)],
loc='best')
plt.ylabel('Frequency')
plt.title('SalePrice distribution')

#Get also the QQ-plot
fig = plt.figure()
res = stats.probplot(train['SalePrice'], plot=plt)
plt.show()
```



(a) 分布曲线　　(b) QQ plot

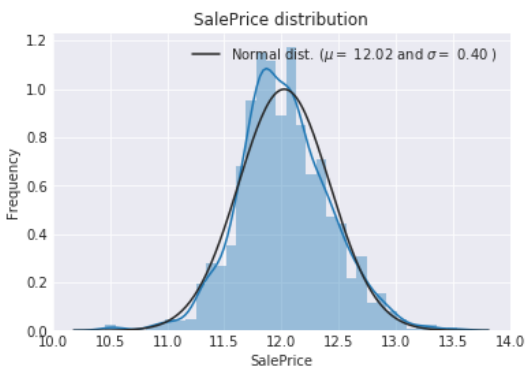
通过观察，发现数据分布整体左偏，而对模型而言，数据符合正态分布是一个很好地性质。因此我们接下来对该变量进行变换使其更接近于正态分布。这里我们对目标变量采用 log **变换**。

Reference：机器学习中，在统计学中为什么要对变量取对数？

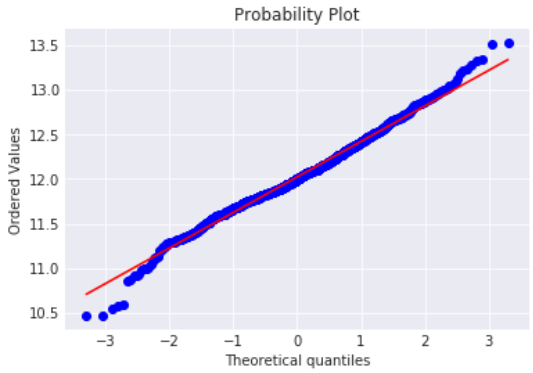
```
#We use the numpy fuction log1p which  applies log(1+x) to all elements of the column
train["SalePrice"] = np.log1p(train["SalePrice"])

#Check the new distribution
sns.distplot(train['SalePrice'] , fit=norm);

# Get the fitted parameters used by the function
(mu, sigma) = norm.fit(train['SalePrice'])
print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))
# mu = 12.02 and sigma = 0.40
```

```
#Now plot the distribution
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)],
loc='best')
plt.ylabel('Frequency')
plt.title('SalePrice distribution')

#Get also the QQ-plot
fig = plt.figure()
res = stats.probplot(train['SalePrice'], plot=plt)
plt.show()
```

可以看到，log 变换并不改变均值 mu 和方差 sigma，但数据的分布现在更接近正态分布了。



(c) 分布曲线



(d) QQ plot

对于其他特征，我们将各特征的 Skewed（偏度）考虑其中，偏度可以很好地体现该特征分布和正态分布的差异。

```
numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index

# Check the skew of all numerical features
skewed_feats = all_data[numeric_feats].apply(lambda x: skew(x.dropna())).sort_values(
                                            ascending=False)
print("\nSkew in numerical features: \n")
skewness = pd.DataFrame({'Skew' :skewed_feats})
skewness.head(10)
```

随后，对于高偏度的特征，我们采用 **Box Cox 变换**，其形式如下。

$$y(\lambda) = \begin{cases} \frac{y^{\lambda}-1}{\lambda}, & \lambda \neq 0 \\ \ln y, & \lambda = 0 \end{cases}$$

上式中，λ 是一个待定参数，如果 $\lambda = 1$ 则退化为 log 变换，这里我们将 λ 设置为 0.15。我们以 0.75 作为偏度阈值，认为偏度高于该阈值的特征则需要进行特征变换。

Reference：如何数据正态化：Box-Cox 变换

```
print("There are {} skewed numerical features to Box Cox transform".format(skewness.shape[0])
                                                                    )
# There are 59 skewed numerical features to Box Cox transform

from scipy.special import boxcox1p
skewed_features = skewness.index
lam = 0.15
```

```
for feat in skewed_features:
#all_data[feat] += 1
all_data[feat] = boxcox1p(all_data[feat], lam)

#all_data[skewed_features] = np.log1p(all_data[skewed_features])
```

	Skew
MiscVal	21.940
PoolArea	17.689
LotArea	13.109
LowQualFinSF	12.085
3SsnPorch	11.372
LandSlope	4.973
KitchenAbvGr	4.301
BsmtFinSF2	4.145
EnclosedPorch	4.002
ScreenPorch	3.945

Figure 2: 各特征对应的偏度

3.3 缺失值处理

现在，我们来处理数据集中的缺失值，由于训练集和测试集都存在缺失值，因此我们首先将它们整合起来，再一块处理。

```
ntrain = train.shape[0]
ntest = test.shape[0]
y_train = train.SalePrice.values
all_data = pd.concat((train, test)).reset_index(drop=True)
all_data.drop(['SalePrice'], axis=1, inplace=True)
print("all_data size is : {}".format(all_data.shape))
# all_data size is : (2917, 79)
```

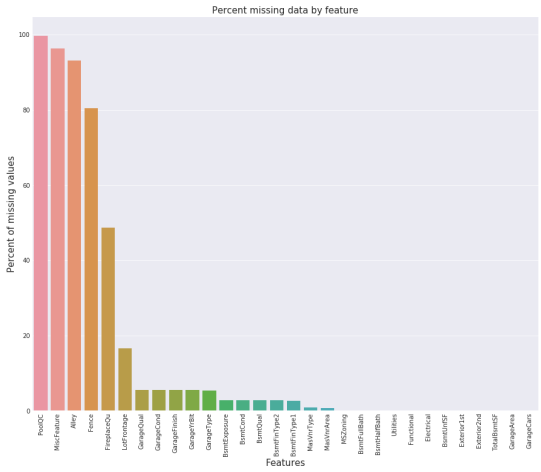
然后我们统计各特征中的缺失值数量并绘制相应的柱状图。

```
all_data_na = (all_data.isnull().sum() / len(all_data)) * 100
all_data_na = all_data_na.drop(all_data_na[all_data_na == 0].index).sort_values(ascending=
                                                False)[:30]
missing_data = pd.DataFrame({'Missing Ratio' :all_data_na})
missing_data.head(20)

f, ax = plt.subplots(figsize=(15, 12))
plt.xticks(rotation='90')
sns.barplot(x=all_data_na.index, y=all_data_na)
plt.xlabel('Features', fontsize=15)
plt.ylabel('Percent of missing values', fontsize=15)
plt.title('Percent missing data by feature', fontsize=15)
```

接下来，我们分别对缺失值进行填充。如下离散变量指代房屋是否具有某项特征，如果没有，我们用 None 填充即可。

	Missing Ratio
PoolQC	99.691
MiscFeature	96.400
Alley	93.212
Fence	80.425
FireplaceQu	48.680
LotFrontage	16.661
GarageQual	5.451
GarageCond	5.451
GarageFinish	5.451
GarageYrBlt	5.451
GarageType	5.382
BsmtExposure	2.811
BsmtCond	2.811
BsmtQual	2.777
BsmtFinType2	2.743
BsmtFinType1	2.708
MasVnrType	0.823
MasVnrArea	0.788
MSZoning	0.137
BsmtFullBath	0.069

(a) 缺失值统计



(b) 柱状图

```
all_data["PoolQC"] = all_data["PoolQC"].fillna("None")  # 泳池
all_data["MiscFeature"] = all_data["MiscFeature"].fillna("None") # 杂项功能
all_data["Alley"] = all_data["Alley"].fillna("None") # 小径
all_data["Fence"] = all_data["Fence"].fillna("None")  # 围栏
all_data["FireplaceQu"] = all_data["FireplaceQu"].fillna("None")  # 壁炉
for col in ('GarageType', 'GarageFinish', 'GarageQual', 'GarageCond'):  # 车库类型等
  all_data[col] = all_data[col].fillna('None')
for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2'): # 地下室
                                                                    类型等
  all_data[col] = all_data[col].fillna('None')
all_data["MasVnrType"] = all_data["MasVnrType"].fillna("None")  # 是否有瓷砖铺面
all_data['MSSubClass'] = all_data['MSSubClass'].fillna("None")  # 建筑类型
```

以下缺失值为连续变量，我们用 0 填充。

```
for col in ('GarageYrBlt', 'GarageArea', 'GarageCars'): # 车库面积等
  all_data[col] = all_data[col].fillna(0)
for col in ('BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF','TotalBsmtSF', 'BsmtFullBath', '
                                                    BsmtHalfBath'): # 地下室面积等
  all_data[col] = all_data[col].fillna(0)
all_data["MasVnrArea"] = all_data["MasVnrArea"].fillna(0) # 瓷砖铺面面积
```

对于某些特征的缺失值，我们不能简单地用 None 或者 0 来填充，我们需要理解其意义，并采用一些特殊手段例如插值。如变量 LotFrontage 表示街道的面积，尽管该数据缺失了，但我们可以从“相邻的房屋对应的街道面积应该是相似的”来填充该值，因此，我们计算统计所有房屋 LotFrontage 的中位数来填充该缺失值。

```
#Group by neighborhood and fill in missing value by the median LotFrontage of all the
                                                    neighborhood
all_data["LotFrontage"] = all_data.groupby("Neighborhood")["LotFrontage"].transform(
  lambda x: x.fillna(x.median()))
```

MSZoning 表明销售的房屋类型，若该值缺失，我们则统计最常见的房屋类型来填充它。

```
all_data['MSZoning'] = all_data['MSZoning'].fillna(all_data['MSZoning'].mode()[0])
```

Utilities 表明房屋的共用资源是否可用如电力、煤气、水等，而数据集中只存在 1 个 NoSeWa 和 2 个 NA，其他变量全为 AllPub，因此该变量对模型预测起不到什么作用，我们可以将之删除。

```
all_data = all_data.drop(['Utilities'], axis=1)
```

Functional 表明房屋类型，如果该值缺失，我们则用 typical（典型的）来填充。

```
all_data["Functional"] = all_data["Functional"].fillna("Typ")
```

部分特征只存在一个缺失值，对于这类缺失值，我们用特征中最常见的变量来填充它们。

```
all_data['Electrical'] = all_data['Electrical'].fillna(all_data['Electrical'].mode()[0])
all_data['KitchenQual'] = all_data['KitchenQual'].fillna(all_data['KitchenQual'].mode()[0])
all_data['Exterior1st'] = all_data['Exterior1st'].fillna(all_data['Exterior1st'].mode()[0])
all_data['Exterior2nd'] = all_data['Exterior2nd'].fillna(all_data['Exterior2nd'].mode()[0])
all_data['SaleType'] = all_data['SaleType'].fillna(all_data['SaleType'].mode()[0])
```

现在，数据集中的缺失值都被我们处理了，我们可以检验是否还存在缺失值。

```
#Check remaining missing values if any
all_data_na = (all_data.isnull().sum() / len(all_data)) * 100
all_data_na = all_data_na.drop(all_data_na[all_data_na == 0].index).sort_values(ascending=
                                        False)
missing_data = pd.DataFrame({'Missing Ratio' :all_data_na})
missing_data.head()
```

3.4 相关性分析

由于我们的特征繁多，因此特征之间的关联值得挖掘，这里我们计算各变量之间的协方差并绘制相应的热力图3（heatmap，颜色能直观反映关联的强弱）。

```
#Correlation map to see how features are correlated with SalePrice
corrmat = train.corr()
plt.subplots(figsize=(12,9))
sns.heatmap(corrmat, vmax=0.9, square=True)
```

通过该图我们可以直观地发现变量之间的关联程度，例如 GarageCars（车库的汽车容量）和 CarageArea（车库的面积大小），但这两个特征实际上指代的是同一个事物，因此我们思考数据集中的某些特征应该是可以被进一步加工的。

3.5 特征组合

根据我们的尝试，房屋的面积跟房价应该高度相关，因此，数据集中的房屋面积有 TotalBsmtSF（地下室面积）、1stFlrSF（一楼面积）、2ndFlrSF（二楼面积），因此我们增添一个新的特征 TotalSF（总面积）。

```
# Adding total sqfootage feature
all_data['TotalSF'] = all_data['TotalBsmtSF'] + all_data['1stFlrSF'] + all_data['2ndFlrSF']
```

3.6 特征编码

尽管有些特征它用了连续变量，但其代表的意义是类型，与数值的大小没有联系。如 MSSubClass（建筑类型）和 OverallCond（房屋的条件）。

Reference：OneHotEncoder 独热编码和 LabelEncoder 标签编码

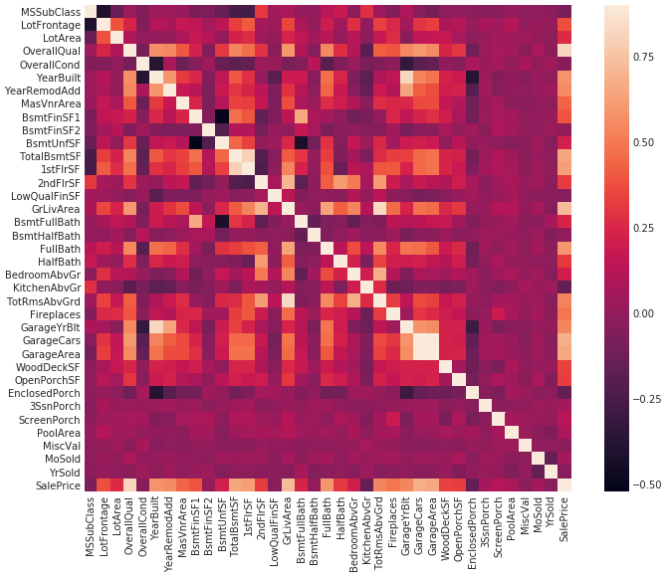


Figure 3: 特征相关性热力图

```
MSSubClass: Identifies the type of dwelling involved in the sale.

20  1-STORY 1946 & NEWER ALL STYLES
30  1-STORY 1945 & OLDER
40  1-STORY W/FINISHED ATTIC ALL AGES
45  1-1/2 STORY - UNFINISHED ALL AGES
50  1-1/2 STORY FINISHED ALL AGES
60  2-STORY 1946 & NEWER
70  2-STORY 1945 & OLDER
75  2-1/2 STORY ALL AGES
80  SPLIT OR MULTI-LEVEL
85  SPLIT FOYER
90  DUPLEX - ALL STYLES AND AGES
120 1-STORY PUD (Planned Unit Development) - 1946 & NEWER
150 1-1/2 STORY PUD - ALL AGES
160 2-STORY PUD - 1946 & NEWER
180 PUD - MULTILEVEL - INCL SPLIT LEV/FOYER
190 2 FAMILY CONVERSION - ALL STYLES AND AGES

OverallCond: Rates the overall condition of the house

10  Very Excellent
9 Excellent
8 Very Good
7 Good
6 Above Average
5 Average
4 Below Average
3 Fair
2 Poor
1 Very Poor
```

因此，我们将其转换为字符串，作为类别变量。

```
#MSSubClass=The building class
all_data['MSSubClass'] = all_data['MSSubClass'].apply(str)


#Changing OverallCond into a categorical variable
all_data['OverallCond'] = all_data['OverallCond'].astype(str)


#Year and month sold are transformed into categorical features.
all_data['YrSold'] = all_data['YrSold'].astype(str)
all_data['MoSold'] = all_data['MoSold'].astype(str)
```

为了让后续模型能使用类别特征，我们采用硬编码将字符串类型特征用离散数字来表示。例如 PoolQC（泳池品质）有如下的属性值，我们为每个属性值分配一个数值来表示。注意，这只是一种编码方式，该特征依然是离散的，不能看作连续变量，因此我们后续会再采用一次 Dummy 编码。

- Ex Excellent → 0
- Gd Good → 1
- TA Average/Typical → 2
- Fa Fair → 3
- NA No Pool → 4

```
  from sklearn.preprocessing import LabelEncoder
  cols = ('FireplaceQu', 'BsmtQual', 'BsmtCond', 'GarageQual', 'GarageCond',
    'ExterQual', 'ExterCond','HeatingQC', 'PoolQC', 'KitchenQual', 'BsmtFinType1',
    'BsmtFinType2', 'Functional', 'Fence', 'BsmtExposure', 'GarageFinish', 'LandSlope',
    'LotShape', 'PavedDrive', 'Street', 'Alley', 'CentralAir', 'MSSubClass', 'OverallCond',
    'YrSold', 'MoSold')
  # process columns, apply LabelEncoder to categorical features
  for c in cols:
    lbl = LabelEncoder()
    lbl.fit(list(all_data[c].values))
    all_data[c] = lbl.transform(list(all_data[c].values))

  # shape
  print('Shape all_data: {}'.format(all_data.shape))
```

我们接着对类型特征采用 dummy 编码，它是一种 one-hot 的编码方式，还是用前面 PoolQC（泳池品质）的例子来说明

- Ex Excellent → 0 → [1, 0, 0, 0, 0]
- Gd Good → 1 → [0, 1, 0, 0, 0]
- TA Average/Typical → 2 → [0, 0, 1, 0, 0]
- Fa Fair → 3 → [0, 0, 0, 1, 0]
- NA No Pool → 4 → [0, 0, 0, 0, 1]

这种编码方式，将会使数据的特征数量将会进一步增加。现在特征的维度变成 220 维了。

```
all_data = pd.get_dummies(all_data)
print(all_data.shape)
# (2917, 220)
all_data.head()
```

	1stFlrSF	2ndFlrSF	3SsnPorch	Alley	BedroomAbvGr	BsmtCond	BsmtExposure	BsmtFinSF1	BsmtFinSF2	BsmtFinType1	...	SaleCondition_Partial	SaleType_COD	SaleType_CWD	SaleType_Con	SaleType_ConLD	SaleType_ConLI	SaleType
0	856	854	0	1	3	4	3	706.000	0.000	2	...	0	0	0	0	0	0	
1	1262	0	0	1	3	4	1	978.000	0.000	0	...	0	0	0	0	0	0	
2	920	866	0	1	3	4	2	486.000	0.000	2	...	0	0	0	0	0	0	
3	961	756	0	1	3	1	3	216.000	0.000	0	...	0	0	0	0	0	0	
4	1145	1053	0	1	4	4	0	655.000	0.000	2	...	0	0	0	0	0	0	

5 rows × 220 columns

3.7 特征降维

在上一部分，可以看到，我们的特征数量已经到了 220，因此，我们这里可以尝试对数据集进行降维操作，这里我们考虑最经典的 PCA 降维方法。PCA 不仅仅只是对数据降维，更重要的，它可以去除特征之间的相关性，这非常有意义，因为高度相关的特征是相当冗余的，如前面讨论的 GarageCars（车库的汽车容量）和 CarageArea（车库的面积大小）。

Reference：【机器学习】降维——PCA（非常详细）

```
from sklearn.decomposition import PCA
pca = PCA(n_components=100)

all_data_pca=pca.fit_transform(all_data)
print(all_data_pca.shape)
# (2917, 100)
print(all_data_pca)
# [[ -1.68735421e+03   7.99920079e+01  -2.17674822e+02 ...,   1.67607351e-02 4.29845580e-02
                                                    2.09808600e-02]
# [ -5.34531719e+02   2.24907545e+01   6.33482673e+02 ...,   3.80966241e-02 1.25672152e-01  -
                                                    1.25540566e-01]
# [  1.11593525e+03   1.54299842e+02  -4.19555216e+02 ...,   7.85560695e-02 5.51572362e-02  -
                                                    1.41797260e-02] ...,
# [  9.85524381e+03  -4.91633183e+02   7.59628423e+02 ...,   2.05231442e-01 1.55375011e-01
                                                    1.04929548e-01]
# [  2.58496148e+02  -1.07432206e+03   4.61167581e+02 ...,  -2.66942970e-01 -2.24496682e-01
                                                    1.92731117e-01]
# [ -4.87977201e+02   5.99109134e+02  -3.42905019e+02 ...,  -8.62495514e-05 -1.41808277e-01
                                                    -7.16593745e-02]]
```

至此，我们已对原始数据进行了相对详实的特征工程操作了，我们将处理的特征分配回训练集和测试集

```
train = all_data[:ntrain]
test = all_data[ntrain:]
```

接下来我们将步入模型部分。

4 机器学习模型

机器学习门下涉及诸多算法，按照其是否依赖标签，可以简单地划分为监督（Supervised）和无监督算法（Unsupervised）。有监督任务中主要有回归（Regreesion）和分类（Classification），而无监督的代表性任务是聚类（Clustering）。即便我们选取了回归模型，其分支下依然有各式各样的子模型。

- Supervised learning
  - Linear Models
    - Ordinary Least Squares: Ridge, Lasso, Elastic-Net
    - Logistic regression
    - ···
  - Linear and Quadratic Discriminant Analysis
  - Kernel ridge regression
  - Support Vector Machines
  - Stochastic Gradient Descent
  - Nearest Neighbors
  - Gaussian Processes
  - Cross decomposition
  - Naive Bayes
  - Decision Trees
  - Ensemble methods
    - Stacking：Meta-model
    - Bagging：Random Forest
    - Boosting：Adaboost，GBDT
  - Multiclass and multilabel algorithms
  - Feature selection
  - Semi-Supervised
  - Isotonic regression
  - Probability calibration
  - Neural network models (supervised)
- Unsupervised learning
  - Gaussian mixture models
  - Manifold learning
  - Clustering
  - Biclustering
  - Decomposing signals in components (matrix factorization problems)
  - Covariance estimation
  - Novelty and Outlier Detection
  - Density Estimation

– Neural network models (unsupervised)

Reference：https://scikit-learn.org/stable/user_guide.html

如前面讨论的，这个任务是一个有监督回归问题，因此我们首先尝试用经典的回归模型来解决该问题，我们先导入相关的库函数。

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet, BayesianRidge,
                                                    LassoLarsIC
from sklearn.ensemble import RandomForestRegressor,  GradientBoostingRegressor
from sklearn.kernel_ridge import KernelRidge
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import RobustScaler
from sklearn.base import BaseEstimator, TransformerMixin, RegressorMixin, clone
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.metrics import mean_squared_error
import xgboost as xgb
import lightgbm as lgb
```

为了更好地评估模型的训练效果以及防止出现过拟合，我们采用 k 折交叉验证的方式来训练模型，我们简单地将 k 设置为 5，并将评价指标设置为均方差损失。

进一步了解交叉验证及 k 的选择，「交叉验证」到底如何选择 K 值？

```
#Validation function
n_folds = 5

def rmsle_cv(model):
  kf = KFold(n_folds, shuffle=True, random_state=42).get_n_splits(train.values)
  rmse= np.sqrt(-cross_val_score(model, train.values, y_train, scoring="
                                                    neg_mean_squared_error", cv = kf))
  return(rmse)
```

4.1 回归模型

我们依次尝试不同的回归模型，相关的函数在 scikit-learn 库中查询，这里我们将不对模型的参数做过多的讨论。

Reference：https://scikit-learn.org/stable/modules/linear_model.html

- 原始二乘回归（Ordinary Least Squares）

```
linear = make_pipeline(RobustScaler(), LinearRegression())
```

- Ridge 回归

```
ridge = make_pipeline(RobustScaler(), Ridge(alpha =0.9, random_state=1))
```

- LASSO 回归

```
lasso = make_pipeline(RobustScaler(), Lasso(alpha =0.0005, random_state=1))
```

- ElasticNet 回归

```
ENet = make_pipeline(RobustScaler(), ElasticNet(alpha=0.0005, l1_ratio=.9, random_state=
                                                    1))
```

- Kernel Ridge 回归

```
KRR = KernelRidge(alpha=0.6, kernel='polynomial', degree=2, coef0=2.5)
```

然后，我们计算并输出每个模型的误差。

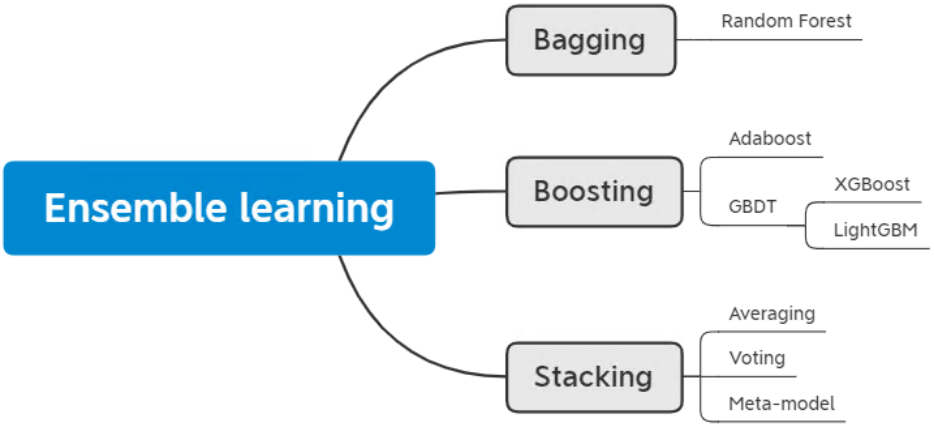
```
score = rmsle_cv(linear)
print("\nLinearRregression score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
# LinearRregression score: 0.1238 (0.0109)

score = rmsle_cv(ridge)
print("\nRidge score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
# Ridge score: 0.1185 (0.0099)

score = rmsle_cv(lasso)
print("\nLasso score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
# Lasso score: 0.1115 (0.0074)

score = rmsle_cv(ENet)
print("ElasticNet score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
# ElasticNet score: 0.1116 (0.0074)

score = rmsle_cv(KRR)
print("Kernel Ridge score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
# Kernel Ridge score: 0.1153 (0.0075)
```

可以看到，相比原始的二乘回归模型，各变体均能取得进一步的提升，这也与模型的参数选取有关，由于结果差异不大，我们便不再过多赘述。

4.2 集成方法

目前，我们已经尝试了基础的回归模型，接下来，我们尝试各类集成方法。通常地，集成方法可以在原有的模型上取得进一步的性能提升。集成方法主要可以分为以下三类。

Reference：集成学习-Boosting,Bagging 与 Stacking 、集成学习之 bagging,stacking,boosting



4.2.1 Bagging

- Bagging

```
Bagging = BaggingRegressor(base_estimator=LinearRegression(), n_estimators=10,
                                                random_state=1)
```

- Random Forest

```
RandomForest = RandomForestRegressor(n_estimators=100, criterion='mse')
```

Reference：独家 | 一文读懂随机森林的解释和实现
然后，我们计算并输出相应的误差。

```
score = rmsle_cv(Bagging)
print("Bagging score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
# Bagging score: 0.1190 (0.0092)

score = rmsle_cv(RandomForest)
print("RandomForest score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
# RandomForest score: 0.1447 (0.0054)
```

这里由于没有对参数进行广泛的尝试，所以性能并不一定比单模型会有提升。

4.2.2 Boosting

接下来，我们尝试另一种集成学习策略 Boosting，并尝试了其各类变体。

- Adaboost

```
AdaBoost = AdaBoostRegressor(base_estimator=LinearRegression(), n_estimators=50,
                                                learning_rate=1.0, loss='linear',
                                                random_state=None)
```

- GBDT

```
GBoost = GradientBoostingRegressor(n_estimators=3000, learning_rate=0.05,
  max_depth=4, max_features='sqrt',
  min_samples_leaf=15, min_samples_split=10,
  loss='huber', random_state =5)
```

- XGBoost

```
model_xgb = xgb.XGBRegressor(colsample_bytree=0.4603, gamma=0.0468,
  learning_rate=0.05, max_depth=3,
  min_child_weight=1.7817, n_estimators=2200,
  reg_alpha=0.4640, reg_lambda=0.8571,
  subsample=0.5213, silent=1,
  random_state =7, nthread = -1)
```

- LightGBM

```
model_lgb = lgb.LGBMRegressor(objective='regression',num_leaves=5,
  learning_rate=0.05, n_estimators=720,
  max_bin = 55, bagging_fraction = 0.8,
  bagging_freq = 5, feature_fraction = 0.2319,
  feature_fraction_seed=9, bagging_seed=9,
  min_data_in_leaf =6, min_sum_hessian_in_leaf = 11)
```

我们依次输出各模型的误差

```
score = rmsle_cv(AdaBoost)
print("AdaBoost score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
# AdaBoost score: 0.1688 (0.0089)

score = rmsle_cv(GBoost)
print("Gradient Boosting score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
# Gradient Boosting score: 0.1177 (0.0080)

score = rmsle_cv(model_xgb)
print("Xgboost score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
# Xgboost score: 0.1161 (0.0079)

score = rmsle_cv(model_lgb)
print("LGBM score: {:.4f} ({:.4f})\n" .format(score.mean(), score.std()))
# LGBM score: 0.1153 (0.0057)
```

Reference：【机器学习】决策树（中）——Random Forest、Adaboost、GBDT （非常详细）、【机器学习】决策树（下）——XGBoost、LightGBM（非常详细）

4.2.3 Stacking

这里，我们尝试通过模型融合来进一步提升我们的性能。我们首先采用平均策略。

- Averaging

```
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)
```

我们选取这 ENet，GBoost，KRR 和 Lasso，接着我们对其结果采用平均策略

```
averaged_models = AveragingModels(models = (ENet, GBoost, KRR, lasso))

score = rmsle_cv(averaged_models)
print(" Averaged base models score: {:.4f} ({:.4f})\n".format(score.mean(), score.std())
                                                            )
#  Averaged base models score: 0.1091 (0.0075)
```

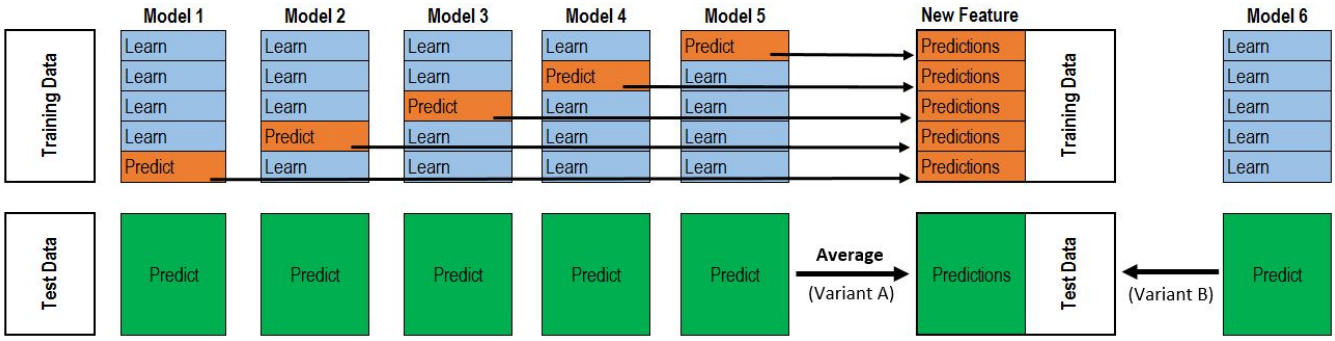
通过这种融合策略，我们目前暂时达到了最低的误差。

- Add a Meta-model 这部分，我们尝试在基础模型上引入一个 Meta model（元模型），训练过程如下：

1. 将总数据集分为两部分，train 和 holdout
2. 使用 train 训练许多个基模型
3. 使用 holdout 来测试基模型
4. 使用 3）中基模型的预测值作为输入和对应的真实标签来训练一个高级的元模型。

反复迭代 1）-3），当测试时，我们使用基模型对测试集的预测作为（meta-features）元特征，然后用元模型输出最终的预测结果。

Reference：Kaggle 机器学习之模型融合（stacking）心得



这里，我们定义元模型对象

```
class StackingAveragedModels(BaseEstimator, RegressorMixin, TransformerMixin):
  def __init__(self, base_models, meta_model, n_folds=5):
    self.base_models = base_models
    self.meta_model = meta_model
    self.n_folds = n_folds

  # We again fit the data on clones of the original models
  def fit(self, X, y):
    self.base_models_ = [list() for x in self.base_models]
    self.meta_model_ = clone(self.meta_model)
    kfold = KFold(n_splits=self.n_folds, shuffle=True, random_state=156)

    # Train cloned base models then create out-of-fold predictions
    # that are needed to train the cloned meta-model
    out_of_fold_predictions = np.zeros((X.shape[0], len(self.base_models)))
    for i, model in enumerate(self.base_models):
      for train_index, holdout_index in kfold.split(X, y):
        instance = clone(model)
        self.base_models_[i].append(instance)
        instance.fit(X[train_index], y[train_index])
        y_pred = instance.predict(X[holdout_index])
        out_of_fold_predictions[holdout_index, i] = y_pred

    # Now train the cloned  meta-model using the out-of-fold predictions as new feature
    self.meta_model_.fit(out_of_fold_predictions, y)
    return self

  #Do the predictions of all base models on the test data and use the averaged
                                                  predictions as
  #meta-features for the final prediction which is done by the meta-model
  def predict(self, X):
```

```
    meta_features = np.column_stack([
      np.column_stack([model.predict(X) for model in base_models]).mean(axis=1)
      for base_models in self.base_models_ ])
    return self.meta_model_.predict(meta_features)
```

为了和前面的平均策略形成对比，我们使用相同 ENet，GBoost 和 KRR 作为基模型，并使用 Lasso 作为元模型。

```
stacked_averaged_models = StackingAveragedModels(base_models = (ENet, GBoost, KRR),
                                                 meta_model = lasso)

score = rmsle_cv(stacked_averaged_models)
print("Stacking Averaged models score: {:.4f} ({:.4f})".format(score.mean(), score.std()
                                                              ))
# Stacking Averaged models score: 0.1085 (0.0074)
```

这里，我们还可以将 Stacking 后的模型作为单模型，再此基础上和 XGBoost 和 LightGBM 再进行一次加权 Stacking。我们定义我们的损失函数

```
def rmsle(y, y_pred):
  return np.sqrt(mean_squared_error(y, y_pred))
```

然后分别使用 StackingRegressor，XGBoost 和 LightGBM 来拟合我们的模型。

```
stacked_averaged_models.fit(train.values, y_train)
stacked_train_pred = stacked_averaged_models.predict(train.values)
stacked_pred = np.expm1(stacked_averaged_models.predict(test.values))
print(rmsle(y_train, stacked_train_pred))
# 0.0781571937916

model_xgb.fit(train, y_train)
xgb_train_pred = model_xgb.predict(train)
xgb_pred = np.expm1(model_xgb.predict(test))
print(rmsle(y_train, xgb_train_pred))
# 0.0785165142425

model_lgb.fit(train, y_train)
lgb_train_pred = model_lgb.predict(train)
lgb_pred = np.expm1(model_lgb.predict(test.values))
print(rmsle(y_train, lgb_train_pred))
# 0.072050888492

'''RMSE on the entire Train data when averaging'''
print('RMSLE score on train data:')
print(rmsle(y_train,stacked_train_pred*0.70 +
xgb_train_pred*0.15 + lgb_train_pred*0.15 ))
# 0.0752374213174
```

因此我们的最终结果为

```
ensemble = stacked_pred*0.70 + xgb_pred*0.15 + lgb_pred*0.15
```

我们的最终集成结果可以用图4.2.3来表示

5 提交结果

最终，我们将其转化为提交所需的格式。

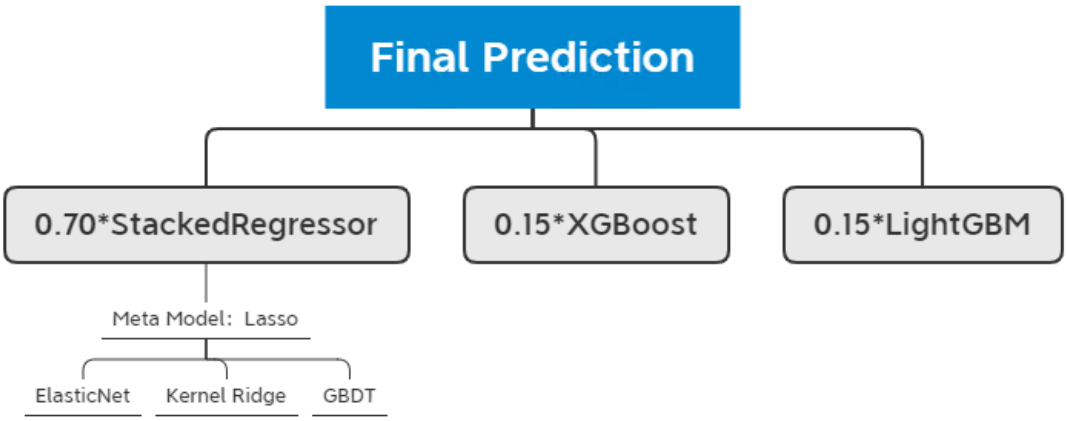


Figure 4: 集成策略

```
sub = pd.DataFrame()
sub['Id'] = test_ID
sub['SalePrice'] = ensemble
sub.to_csv('submission.csv',index=False)
```

然后我们上传该文件

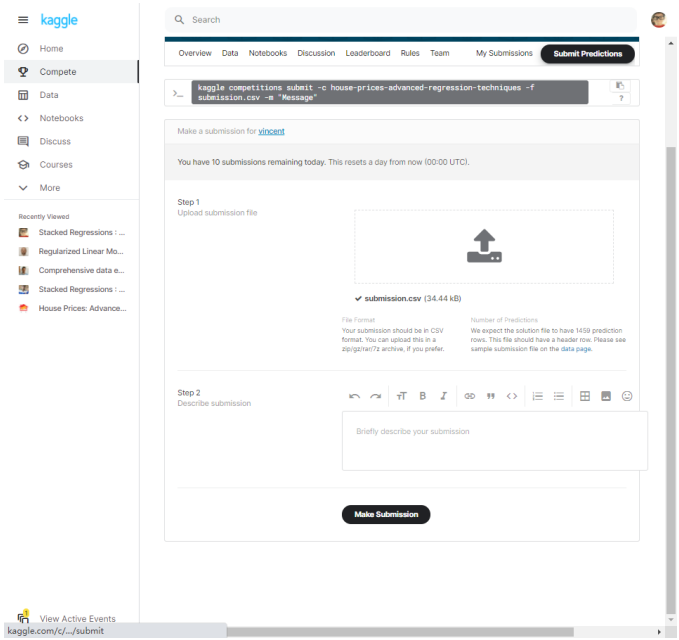


Figure 5: Kaggle 上传结果

Kaggle 即会返回我们的最终结果，可以看到我们的最终方案在测试集上的对应的误差为 0.12010。此外，我们还可以看到我们在所有参赛者中的名次。

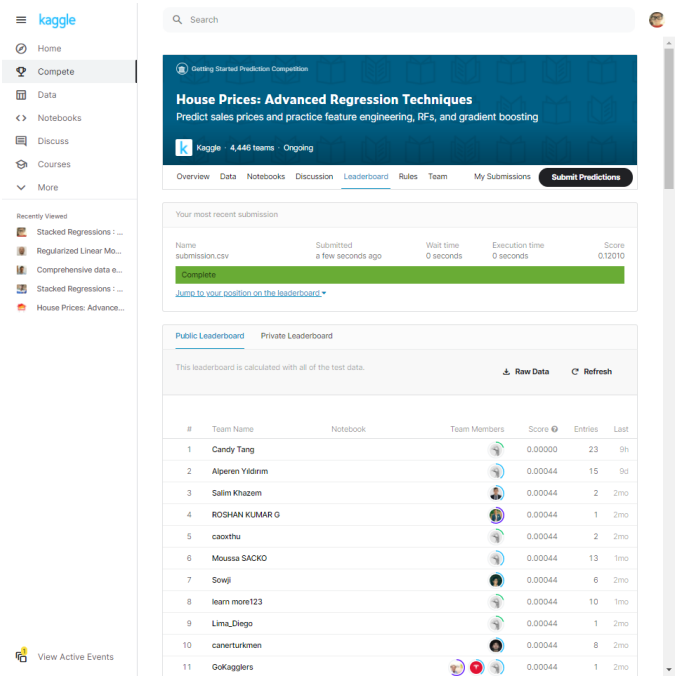


Figure 6: 测试集得分

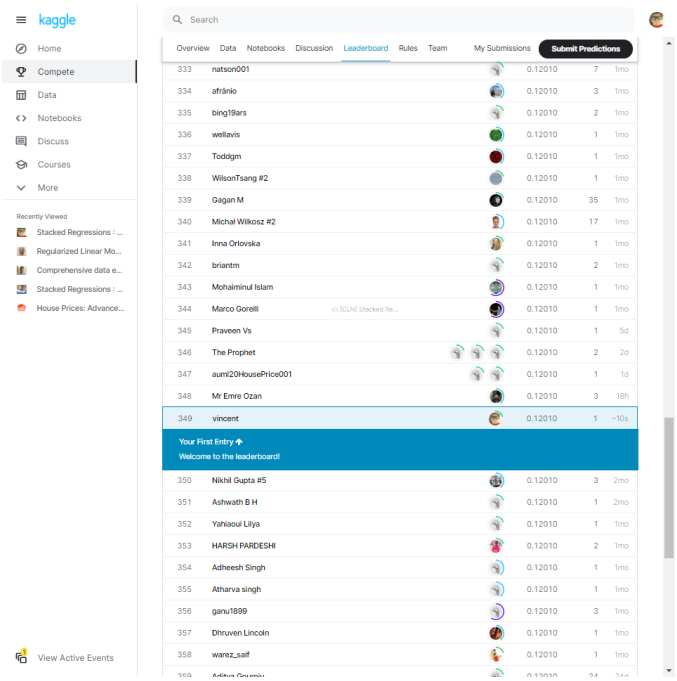


Figure 7: 成绩排名

6 参考资源

本文主要参考了【Stacked Regressions : Top 4% on LeaderBoard】的方案，Kaggle 平台上还提供了其他优秀的思路，以下的资源有兴趣的同学可以进一步学习。

- Comprehensive data exploration with Python
- Regularized Linear Models
- Kaggle 竞赛——房价预测 (House Prices)
- 【干货】Kaggle 实战记录——回归篇之房价预测
- Kaggle 竞赛-房价预测（House Prices）小结