房价预测

2019 年 12 月 15 日

目录

第 1 章　任务介绍

Kaggle 是一个数据科学竞赛的平台。本次实验任务就是 House Prices: Advanced Regression Techniques，一个房价预测的任务。参加 kaggle 最简单的流程就是：

第一步：在 Data 里面下载三个数据集，最基本的就是上面提到的三个文件，有些比赛会有附加的数据描述文件等。

第二步：自己在线下分析，建模，调参，把用 test 数据集预测好的结果，按照 sample_submission 的格式输出到 csv 文件中。

第三步：点击蓝色按钮’Submit Predictions’，把 csv 文件拖拽进去，然后系统就会加载并检验结果，稍等片刻后就会在 Leaderboard 上显示当前结果所在的排名位置。

使用 Anaconda 下的 Jupyter notebook，用到了一些流行的第三方包。

第 2 章　数据集概览

导入相关 Python 包：

[1]:
```
#import some necessary librairies

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
%matplotlib inline
import matplotlib.pyplot as plt  # Matlab-style plotting
import seaborn as sns
color = sns.color_palette()
sns.set_style('darkgrid')
import warnings
def ignore_warn(*args, **kwargs):
pass
warnings.warn = ignore_warn #ignore annoying warning (from sklearn
 ↪and seaborn)

from scipy import stats
```

```
from scipy.stats import norm, skew #for some statistics

pd.set_option('display.float_format', lambda x: '{:.3f}'.format(x))␣
 ↪#Limiting floats output to 3 decimal points

from subprocess import check_output
#print(check_output(["ls", "../input"]).decode("utf8")) #check the␣
 ↪files available in the directory
```

读取 csv 文件：

[2]:
```
train = pd.read_csv('house_price_data/train.csv')
test = pd.read_csv('house_price_data/test.csv')
```

查看数据集前 5 行：

[3]:
```
train.head(5)
```

[3]:
```
Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
0   1          60       RL       65.000     8450   Pave   NaN      Reg
1   2          20       RL       80.000     9600   Pave   NaN      Reg
2   3          60       RL       68.000    11250   Pave   NaN      IR1
3   4          70       RL       60.000     9550   Pave   NaN      IR1
4   5          60       RL       84.000    14260   Pave   NaN      IR1

LandContour Utilities  ... PoolArea PoolQC Fence MiscFeature MiscVal␣
 ↪MoSold  \
0         Lvl    AllPub  ...        0    NaN   NaN         NaN      ␣
 ↪0       2
1         Lvl    AllPub  ...        0    NaN   NaN         NaN      ␣
 ↪0       5
2         Lvl    AllPub  ...        0    NaN   NaN         NaN      ␣
 ↪0       9
3         Lvl    AllPub  ...        0    NaN   NaN         NaN      ␣
 ↪0       2
```

```
4          Lvl    AllPub  ...        0    NaN   NaN         NaN      ␣
 ↪0      12

YrSold  SaleType  SaleCondition  SalePrice
0    2008        WD         Normal     208500
1    2007        WD         Normal     181500
2    2008        WD         Normal     223500
3    2006        WD        Abnorml     140000
4    2008        WD         Normal     250000

[5 rows x 81 columns]
```

[4]:
```
test.head(5)
```

[4]:
```
Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
0  1461          20       RH       80.000    11622   Pave   NaN      ␣
 ↪Reg
1  1462          20       RL       81.000    14267   Pave   NaN      ␣
 ↪IR1
2  1463          60       RL       74.000    13830   Pave   NaN      ␣
 ↪IR1
3  1464          60       RL       78.000     9978   Pave   NaN      ␣
 ↪IR1
4  1465         120       RL       43.000     5005   Pave   NaN      ␣
 ↪IR1

LandContour Utilities  ... ScreenPorch PoolArea PoolQC  Fence␣
 ↪MiscFeature  \
0          Lvl    AllPub  ...         120        0    NaN  MnPrv      ␣
 ↪   NaN
1          Lvl    AllPub  ...           0        0    NaN    NaN      ␣
 ↪ Gar2
2          Lvl    AllPub  ...           0        0    NaN  MnPrv      ␣
 ↪   NaN
```

```
3          Lvl    AllPub  ...           0        0    NaN    NaN      ␣
 ↪  NaN
4          HLS    AllPub  ...         144        0    NaN    NaN      ␣
 ↪  NaN

MiscVal MoSold  YrSold  SaleType  SaleCondition
0       0       6    2010        WD         Normal
1   12500       6    2010        WD         Normal
2       0       3    2010        WD         Normal
3       0       6    2010        WD         Normal
4       0       1    2010        WD         Normal

[5 rows x 80 columns]
```

查看训练、测试集的大小：

[5]:
```
#check the numbers of samples and features
print("The train data size before dropping Id feature is : {} ".
 ↪format(train.shape))
print("The test data size before dropping Id feature is : {} ".
 ↪format(test.shape))

#Save the 'Id' column
train_ID = train['Id']
test_ID = test['Id']

#Now drop the  'Id' colum since it's unnecessary for  the prediction␣
 ↪process.
train.drop("Id", axis = 1, inplace = True)
test.drop("Id", axis = 1, inplace = True)

#check again the data size after dropping the 'Id' variable
print("\nThe train data size after dropping Id feature is : {} ".
 ↪format(train.shape))
```

```
print("The test data size after dropping Id feature is : {} ".
 ↪format(test.shape))
```

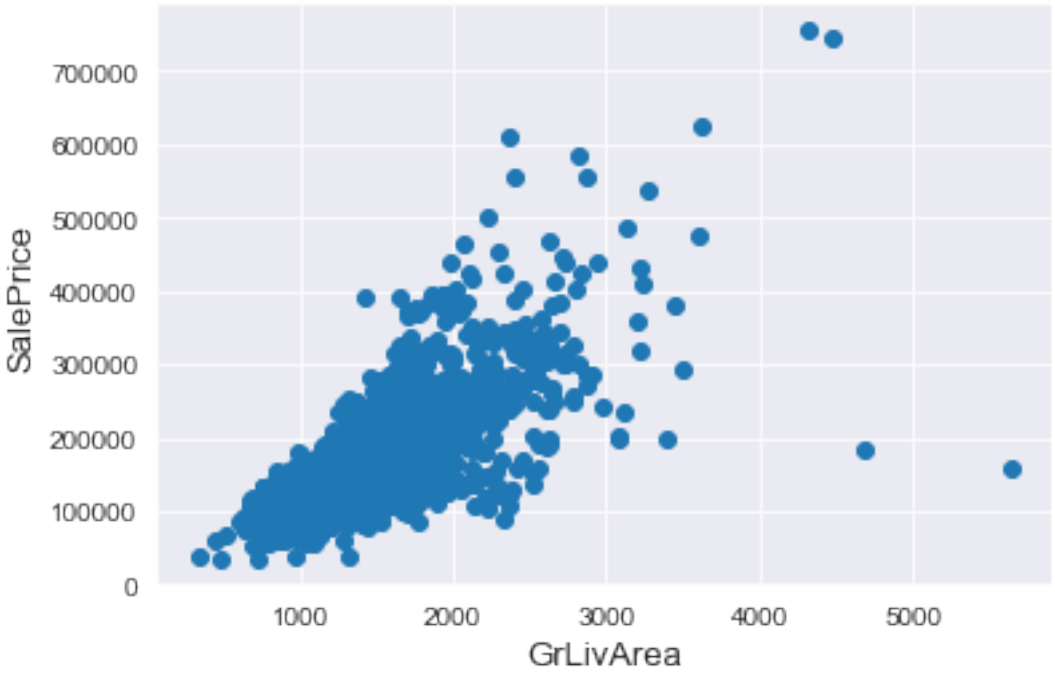
```
The train data size before dropping Id feature is : (1460, 81)
The test data size before dropping Id feature is : (1459, 80)

The train data size after dropping Id feature is : (1460, 80)
The test data size after dropping Id feature is : (1459, 79)
```

第 3 章　特征工程

3.1 离群值处理

通过绘制散点图可以直观地看出特征是否有离群值，这里以 GrLivArea 为例。

[6]:
```
fig, ax = plt.subplots()
ax.scatter(x = train['GrLivArea'], y = train['SalePrice'])
plt.ylabel('SalePrice', fontsize=13)
plt.xlabel('GrLivArea', fontsize=13)
plt.show()
```



[7]:
```
#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()
```

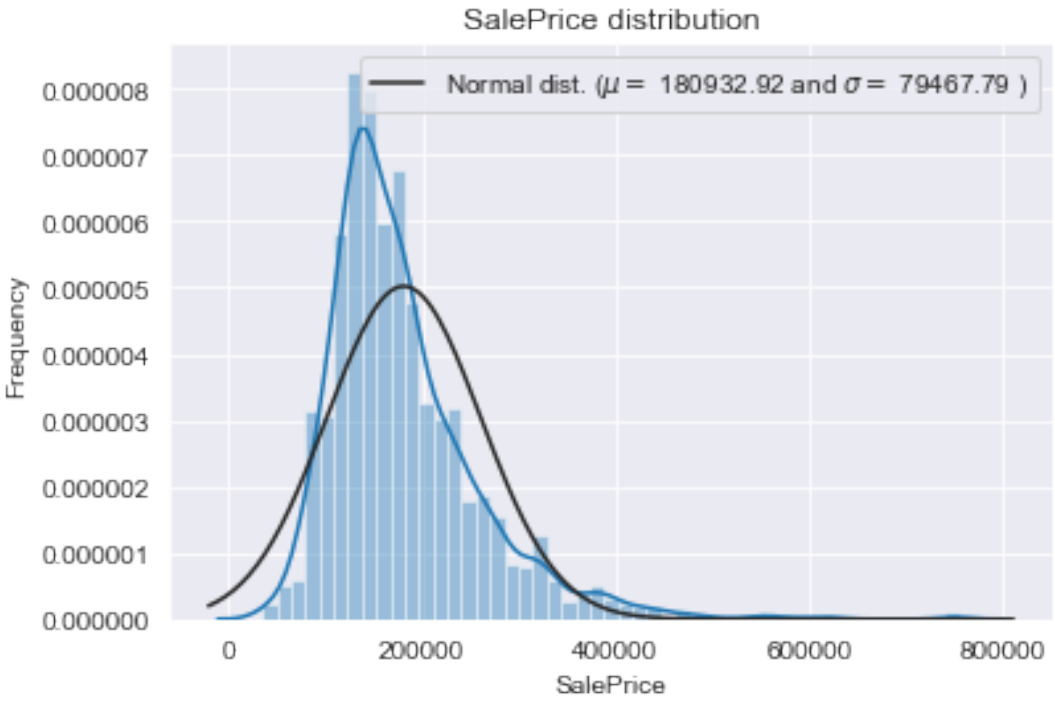
值得一提的是，删除离群值并不总是安全的。不能也不必将所有的离群值全部剔除，因为测试集中依然会有一些离群值。用带有一定噪声的数据训练出的模型会具有更高的鲁棒性，从而在测试集中表现得更好。

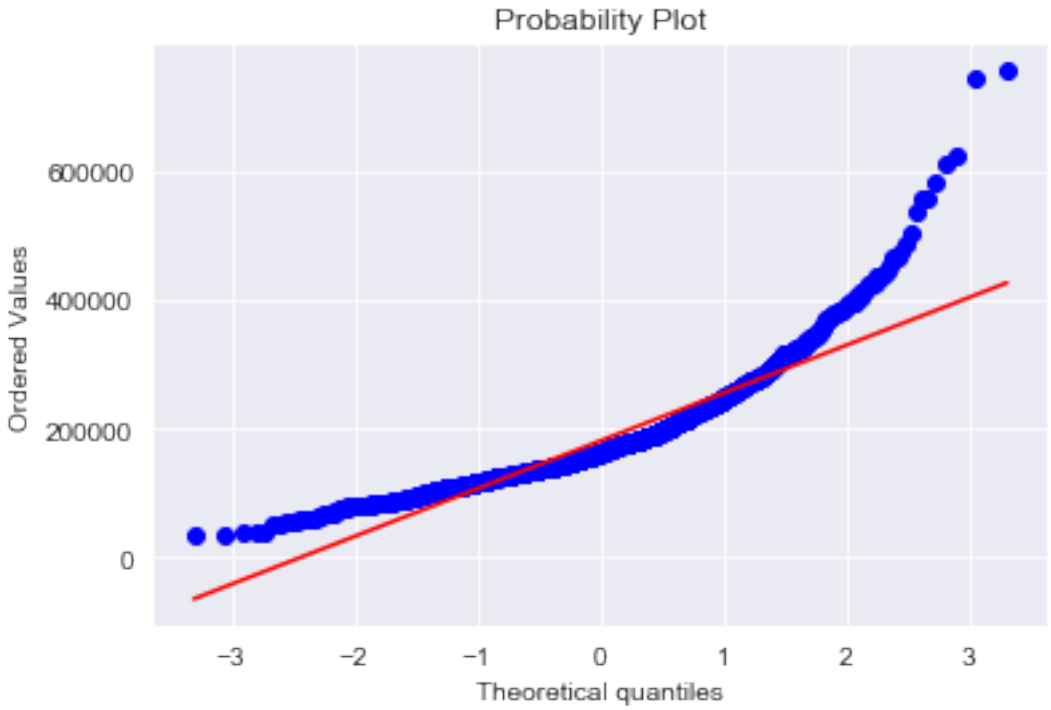
3.2 目标值分析

画出 SalePrice 的分布图和 QQ 图（Quantile Quantile Plot）。QQ 图，它是由标准正态分布的分位数为横坐标，样本值为纵坐标的散点图。如果 QQ 图上的点在一条直线附近，则说明数据近似于正态分布，且该直线的斜率为标准差，截距为均值。

[8]:
```
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))

#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()
```

```
mu = 180932.92 and sigma = 79467.79
```



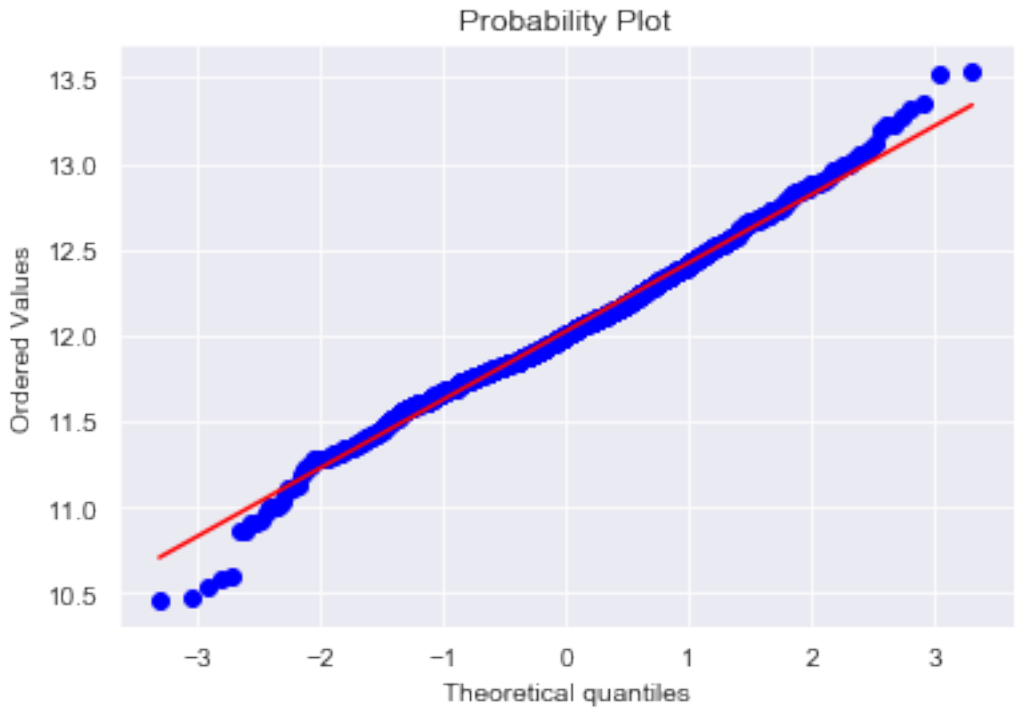


SalePrice 的分布呈正偏态，而线性回归模型要求因变量服从正态分布。对其做**对数变**

换，让数据接近正态分布。

[9]:
```
#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))

#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()
```

```
mu = 12.02 and sigma = 0.40
```



正态分布的数据有很多好的性质，使得后续的模型训练有更好的效果。另一方面，由于

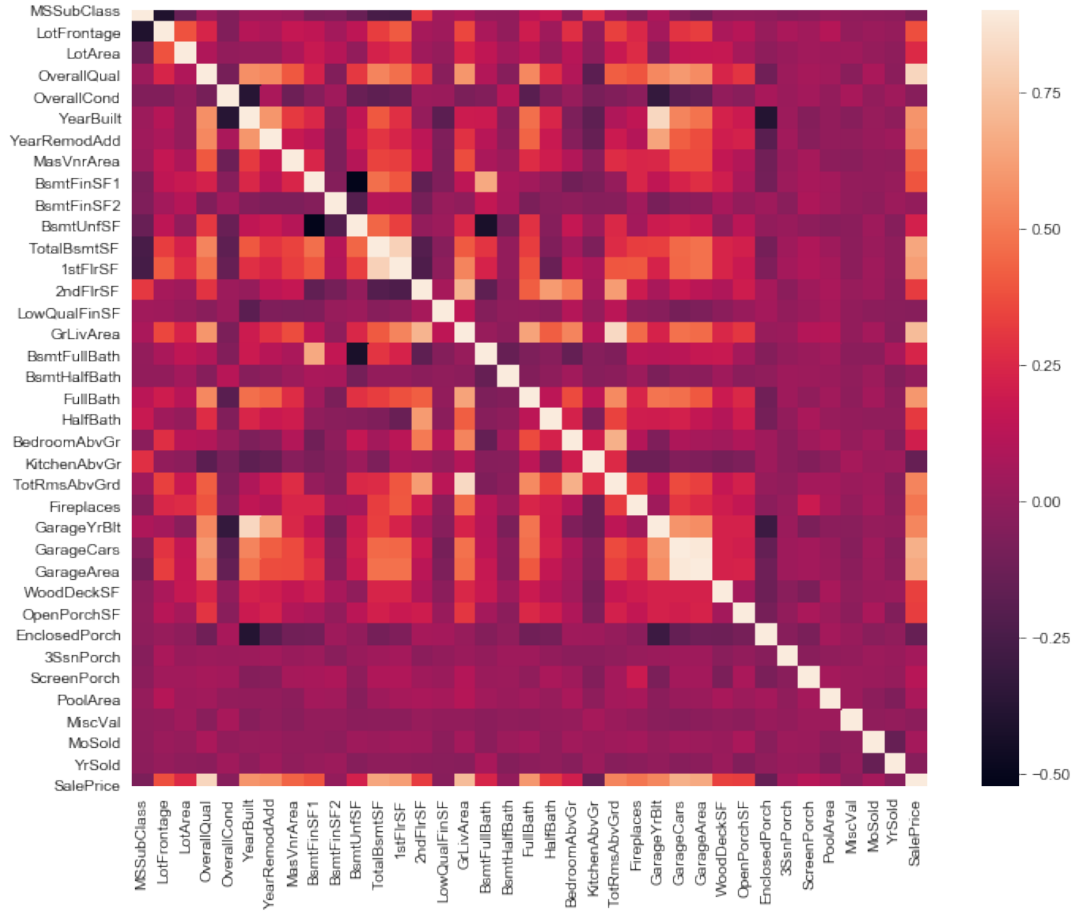
这次比赛最终是对预测值的对数的误差进行评估，所以在本地测试的时候也应该用同样的标准。

3.3 特征相关性

用相关性矩阵热图表现特征与目标值之间以及两两特征之间的相关程度，对特征的处理有指导意义。

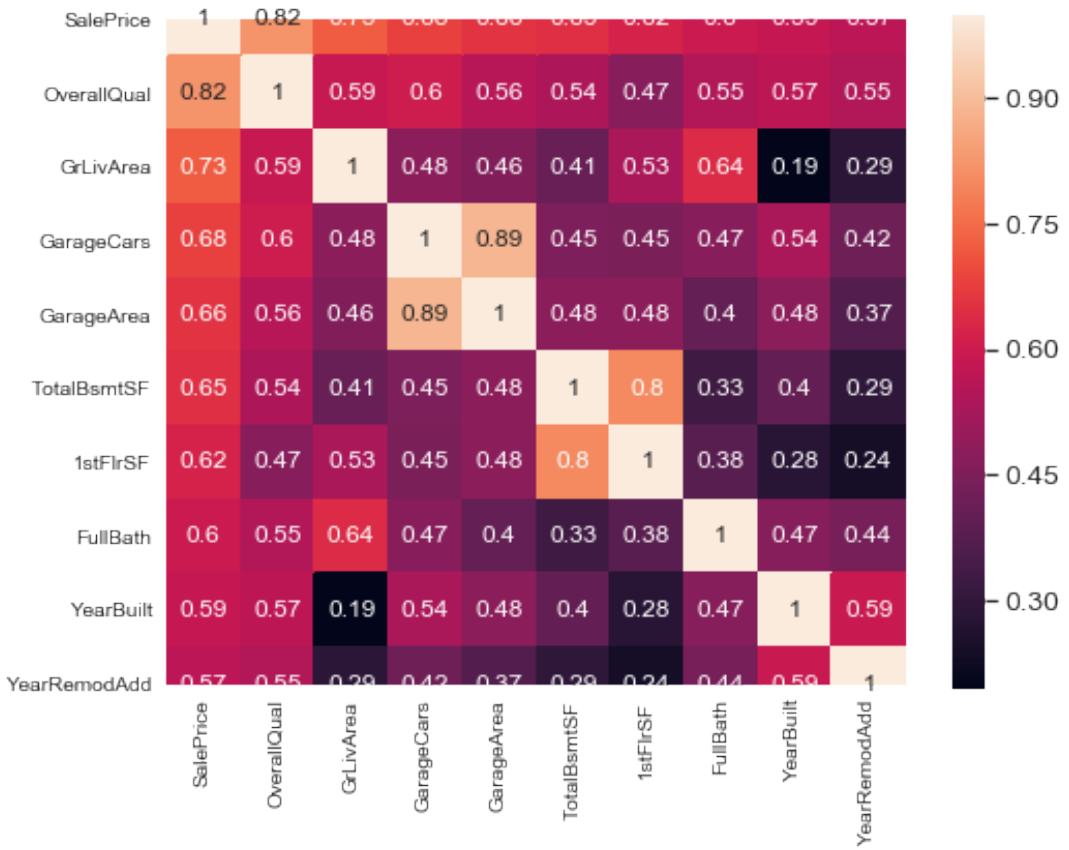
[10]:
```
#Correlation map to see how features are correlated with SalePrice
corrmat = train.corr()
plt.subplots(figsize=(12,9))
sns.heatmap(corrmat, vmax=0.9, square=True)
```

[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x193196132c8>
```



可以选出与 'SalePrice' 相关系数最高的 10 个特征查看其相关情况

[11]:
```
k = 10
top10_attr = corrmat.nlargest(k, 'SalePrice').index
top10_mat = corrmat.loc[top10_attr, top10_attr]
fig,ax = plt.subplots(figsize=(8,6))
sns.set(font_scale=1.25)
sns.heatmap(top10_mat, annot=True, annot_kws={'size':12}, square=True)
# annot;  number, annot_kws : number style
plt.show()
```



3.4 缺失值处理

首先将训练集和测试集合并在一起：

[12]:
```
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)
```

统计各个特征的缺失情况：

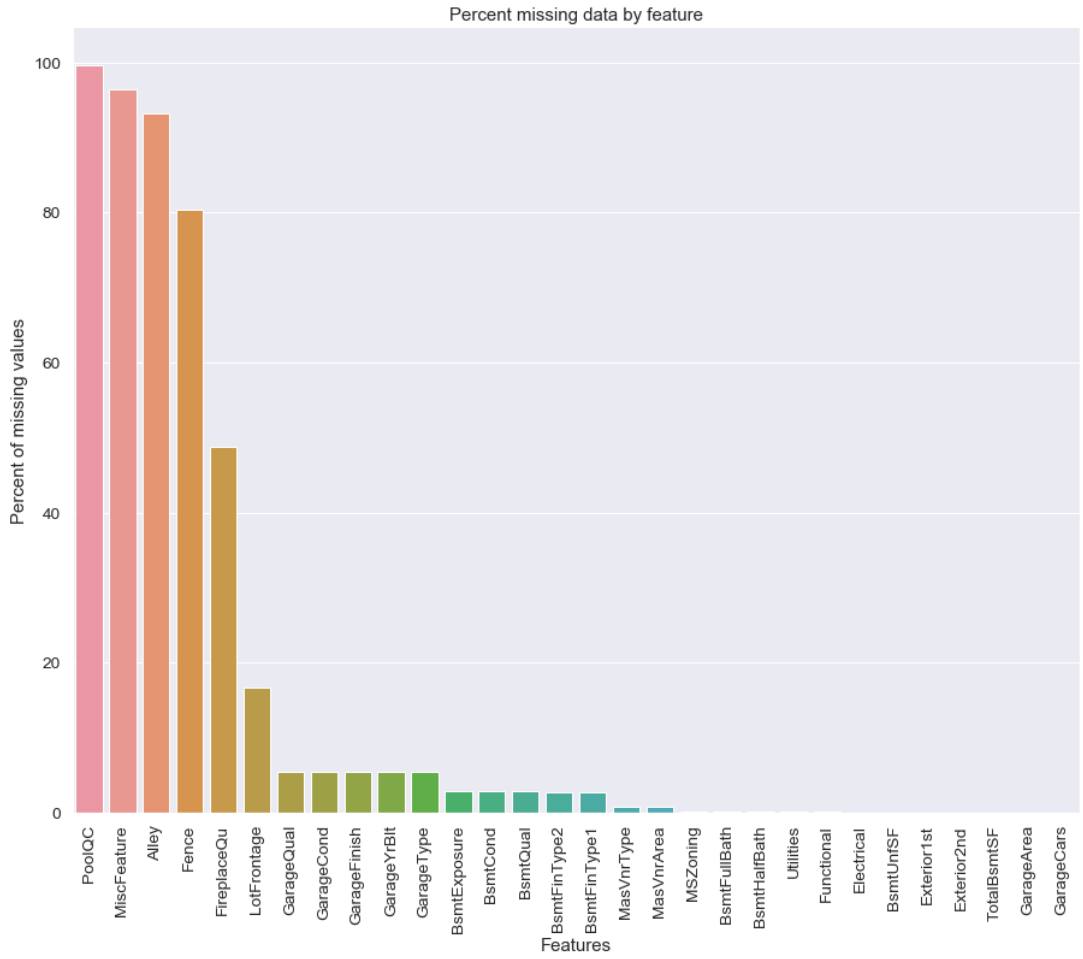
[13]:
```
all_data_na = (all_data.isnull().sum() / len(all_data)) * 100
all_data_na = all_data_na.drop(all_data_na[all_data_na == 0].index).
 ↪sort_values(ascending=False)[:30]
missing_data = pd.DataFrame({'Missing Ratio' :all_data_na})
missing_data.head(20)
```

[13]:

	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

[14]:
```
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)
```

[14]:
```
Text(0.5, 1.0, 'Percent missing data by feature')
```



在 data_description.txt 中已有说明，一部分特征值的缺失是因为这些房子根本没有该项特征，对于这种情况统一用 “None’’ 或者 “0’’ 来填充。

[15]:
```
all_data["PoolQC"] = all_data["PoolQC"].fillna("None")
all_data["MiscFeature"] = all_data["MiscFeature"].fillna("None")
all_data["Alley"] = all_data["Alley"].fillna("None")
```

```
all_data["Fence"] = all_data["Fence"].fillna("None")
all_data["FireplaceQu"] = all_data["FireplaceQu"].fillna("None")
all_data["MasVnrType"] = all_data["MasVnrType"].fillna("None")
all_data["MasVnrArea"] = all_data["MasVnrArea"].fillna(0)
for col in ('GarageType', 'GarageFinish', 'GarageQual', 'GarageCond'):
all_data[col] = all_data[col].fillna('None')
for col in ('GarageYrBlt', 'GarageArea', 'GarageCars'):
all_data[col] = all_data[col].fillna(0)
for col in ('BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF','TotalBsmtSF',␣
 ↪'BsmtFullBath', 'BsmtHalfBath'):
all_data[col] = all_data[col].fillna(0)
for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1',␣
 ↪'BsmtFinType2'):
all_data[col] = all_data[col].fillna('None')
```

对于缺失较少的离散型特征，可以用众数填补缺失值。

[16]:
```
all_data['MSZoning'] = all_data['MSZoning'].
 ↪fillna(all_data['MSZoning'].mode()[0])
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])
```

对于 LotFrontage 项，由于每个 Neighborhood 的房子的 LotFrontage 很可能是比较相近的，所以可以用各个房子所在 Neighborhood 的 LotFrontage 的中位数作为填充值。

[17]:
```
#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()))
```

data_description.txt 中还提到过，Functional 默认是 “Typ’’。

[18]:
```
all_data["Functional"] = all_data["Functional"].fillna("Typ")
```

Utilities 特征有两个缺失值，且只有一个样本是 “NoSeWa’’，除此之外全部都是 “AllPub’’，因此该项特征的方差非常小，可以直接将其删去。

[19]:
```
all_data = all_data.drop(['Utilities'], axis=1)
```

最后确认缺失值是否已全部处理完毕：

[20]:
```
all_data.isnull().sum().max()
```

[20]:
```
0
```

3.5 进一步挖掘特征

注意到有些特征虽然是数值型的，但其实表征的只是不同类别，其数值的大小并没有实际意义，因此将其转化为类别特征。

[21]:
```
all_data['MSSubClass'] = all_data['MSSubClass'].astype(str)
all_data['YrSold'] = all_data['YrSold'].astype(str)
all_data['MoSold'] = all_data['MoSold'].astype(str)
```

反过来，有些类别特征实际上有高低好坏之分，这些特征的质量越高，就可能在一定程度导致房价越高。将这些特征的类别映射成有大小的数字，以此来表征这种潜在的偏序关系。

[22]:
```
all_data['FireplaceQu'] = all_data['FireplaceQu'].map({'Ex': 5, 'Gd':␣
 ↪4, 'TA': 3, 'Fa': 2, 'Po': 1, 'None': 0})
all_data['GarageQual'] = all_data['GarageQual'].map({'Ex': 5, 'Gd':␣
 ↪4, 'TA': 3, 'Fa': 2, 'Po': 1, 'None': 0})
all_data['GarageCond'] = all_data['GarageCond'].map({'Ex': 5, 'Gd':␣
 ↪4, 'TA': 3, 'Fa': 2, 'Po': 1, 'None': 0})
```

```
all_data['GarageFinish'] = all_data['GarageFinish'].map({'Fin': 3,␣
↪'RFn': 2, 'Unf': 1, 'None': 0})
all_data['BsmtQual'] = all_data['BsmtQual'].map({'Ex': 5, 'Gd': 4,␣
↪'TA': 3, 'Fa': 2, 'Po': 1, 'None': 0})
all_data['BsmtCond'] = all_data['BsmtCond'].map({'Ex': 5, 'Gd': 4,␣
↪'TA': 3, 'Fa': 2, 'Po': 1, 'None': 0})
all_data['BsmtExposure'] = all_data['BsmtExposure'].map({'Gd': 4,␣
↪'Av': 3, 'Mn': 2, 'No': 1, 'None': 0})
all_data['BsmtFinType1'] = all_data['BsmtFinType1'].map({'GLQ': 6,␣
↪'ALQ': 5, 'BLQ': 4, 'Rec': 3, 'LwQ': 2, 'Unf': 1, 'None': 0})
all_data['BsmtFinType2'] = all_data['BsmtFinType2'].map({'GLQ': 6,␣
↪'ALQ': 5, 'BLQ': 4, 'Rec': 3, 'LwQ': 2, 'Unf': 1, 'None': 0})
all_data['ExterQual'] = all_data['ExterQual'].map({'Ex': 5, 'Gd': 4,␣
↪'TA': 3, 'Fa': 2, 'Po': 1, 'None': 0})
all_data['ExterCond'] = all_data['ExterCond'].map({'Ex': 5, 'Gd': 4,␣
↪'TA': 3, 'Fa': 2, 'Po': 1, 'None': 0})
all_data['HeatingQC'] = all_data['HeatingQC'].map({'Ex': 5, 'Gd': 4,␣
↪'TA': 3, 'Fa': 2, 'Po': 1, 'None': 0})
all_data['PoolQC'] = all_data['PoolQC'].map({'Ex': 5, 'Gd': 4, 'TA':␣
↪3, 'Fa': 2, 'Po': 1, 'None': 0})
all_data['KitchenQual'] = all_data['KitchenQual'].map({'Ex': 5, 'Gd':␣
↪4, 'TA': 3, 'Fa': 2, 'Po': 1, 'None': 0})
all_data['Functional'] = all_data['Functional'].map({'Typ': 8, 'Min1':
↪ 7, 'Min2': 6, 'Mod': 5, 'Maj1': 4, 'Maj2': 3, 'Sev': 2, 'Sal': 1,␣
↪'None': 0})
all_data['Fence'] = all_data['Fence'].map({'GdPrv': 4, 'MnPrv': 3,␣
↪'GdWo': 2, 'MnWw': 1, 'None': 0})
all_data['LandSlope'] = all_data['LandSlope'].map({'Gtl': 3, 'Mod':␣
↪2, 'Sev': 1, 'None': 0})
all_data['LotShape'] = all_data['LotShape'].map({'Reg': 4, 'IR1': 3,␣
↪'IR2': 2, 'IR3': 1, 'None': 0})
all_data['PavedDrive'] = all_data['PavedDrive'].map({'Y': 3, 'P': 2,␣
↪'N': 1, 'None': 0})
```

```
all_data['Street'] = all_data['Street'].map({'Pave': 2, 'Grvl': 1,␣
 ↪'None': 0})
all_data['Alley'] = all_data['Alley'].map({'Pave': 2, 'Grvl': 1,␣
 ↪'None': 0})
all_data['CentralAir'] = all_data['CentralAir'].map({'Y': 1, 'N': 0})
```

利用一些重要的特征构造更多的特征：

[23]:
```
all_data['TotalSF'] = all_data['TotalBsmtSF'] + all_data['1stFlrSF']␣
 ↪+ all_data['2ndFlrSF']
all_data['OverallQual_TotalSF'] = all_data['OverallQual'] *␣
 ↪all_data['TotalSF']
all_data['OverallQual_GrLivArea'] = all_data['OverallQual'] *␣
 ↪all_data['GrLivArea']
all_data['OverallQual_TotRmsAbvGrd'] = all_data['OverallQual'] *␣
 ↪all_data['TotRmsAbvGrd']
all_data['GarageArea_YearBuilt'] = all_data['GarageArea'] +␣
 ↪all_data['YearBuilt']
```

3.6 Box-Cox 变换

对于数值型特征，希望它们尽量服从正态分布，也就是不希望这些特征出现正负偏态。那么先来计算一下各个特征的偏度：

[24]:
```
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)
skewness = pd.DataFrame({'Skew': skewed_feats})
skewness.head(10)
```

[24]:

	Skew
MiscVal	21.940
PoolQC	19.549
PoolArea	17.689

```
LotArea        13.109
LowQualFinSF   12.085
3SsnPorch      11.372
KitchenAbvGr    4.301
BsmtFinSF2      4.145
Alley           4.137
EnclosedPorch   4.002
```

可以看到这些特征的偏度较高，需要做适当的处理。这里对数值型特征做 Box-Cox 变换，以改善数据的正态性、对称性和方差相等性。

[25]:
```
skewness = skewness[abs(skewness['Skew']) > 0.75]
print("There are {} skewed numerical features to Box Cox transform".
 ↪format(skewness.shape[0]))

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

```
There are 41 skewed numerical features to Box Cox transform
```

3.7 独热编码

对于类别特征，将其转化为独热编码，这样既解决了模型不好处理属性数据的问题，在一定程度上也起到了扩充特征的作用。

[26]:
```
all_data = pd.get_dummies(all_data)
print(all_data.shape)
```

```
(2917, 254)
```

经过处理后的训练集和测试集：

[27]:
```
train = all_data[:ntrain]
test = all_data[ntrain:]
```

[28]:
```
train.head(5)
```

[28]:

```
        1stFlrSF  2ndFlrSF  3SsnPorch  Alley  BedroomAbvGr  BsmtCond␣
↪BsmtExposure  \
      0    11.693    11.686      0.000  0.000             3     1.
↪541         0.730
      1    12.792     0.000      0.000  0.000             3     1.
↪541         1.820
      2    11.892    11.725      0.000  0.000             3     1.
↪541         1.194
      3    12.014    11.354      0.000  0.000             3     1.
↪820         0.730
      4    12.511    12.271      0.000  0.000             4     1.
↪541         1.541

      BsmtFinSF1  BsmtFinSF2  BsmtFinType1  ...  SaleType_ConLI␣
↪SaleType_ConLw  \
      0      11.170       0.000             6  ...               0␣
↪             0
      1      12.063       0.000             5  ...               0␣
↪             0
      2      10.200       0.000             6  ...               0␣
↪             0
      3       8.274       0.000             5  ...               0␣
↪             0
      4      10.971       0.000             6  ...               0␣
↪             0

      SaleType_New  SaleType_Oth  SaleType_WD  YrSold_2006␣
↪YrSold_2007  \
      0            0            0            1            0␣
↪     0
      1            0            0            1            0␣
↪     1
      2            0            0            1            0␣
↪     0
```

```
        3              0              0              1              1
↪       0
        4              0              0              1              0
↪       0

        YrSold_2008  YrSold_2009  YrSold_2010
        0            1            0            0
        1            0            0            0
        2            1            0            0
        3            0            0            0
        4            1            0            0

        [5 rows x 254 columns]
```

第 4 章　建立模型

导入算法包

[29]:
```
from sklearn.linear_model import ElasticNet, Lasso
from sklearn.ensemble import RandomForestRegressor,␣
↪GradientBoostingRegressor
from sklearn.kernel_ridge import KernelRidge
from sklearn.preprocessing import RobustScaler
from sklearn.base import BaseEstimator, TransformerMixin,␣
↪RegressorMixin, clone
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import mean_squared_error
import xgboost as xgb
import lightgbm as lgb
```

4.1 标准化

由于数据集中依然存在一定的离群点，首先用 RobustScaler 对数据进行标准化处理。

[30]:
```
scaler = RobustScaler()
train = scaler.fit_transform(train)
test = scaler.transform(test)
```

[31]:
```
train
```

[31]:
```
array([[-0.51169596,  1.03854137,  0.        , ...,  1.        ,
        0.        ,  0.        ],
       [ 0.33252036,  0.        ,  0.        , ...,  0.        ,
        0.        ,  0.        ],
       [-0.35860172,  1.04195475,  0.        , ...,  1.        ,
        0.        ,  0.        ],
       ...,
       [ 0.19783625,  1.1133624 ,  0.        , ...,  0.        ,
        0.        ,  1.        ],
       [-0.01617541,  0.        ,  0.        , ...,  0.        ,
        0.        ,  1.        ],
       [ 0.32185354,  0.        ,  0.        , ...,  1.        ,
        0.        ,  0.        ]])
```

4.2 评价函数

先定义一个评价函数。采用 5 折交叉验证。与比赛的评价标准一致，用 Root-Mean-Squared-Error (RMSE) 来为每个模型打分。

[32]:
```
#Validation function
n_folds = 5

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

4.3 基本模型

- 套索回归

[33]:
```
lasso = Lasso(alpha=0.0005, random_state=1)
```

[34]:
```
lasso
```

[34]:
```
Lasso(alpha=0.0005, copy_X=True, fit_intercept=True, max_iter=1000,
normalize=False, positive=False, precompute=False, random_state=1,
selection='cyclic', tol=0.0001, warm_start=False)
```

- 弹性网络

[35]:
```
ENet = ElasticNet(alpha=0.0005, l1_ratio=.9, random_state=3)
```

- 核岭回归

[36]:
```
KRR = KernelRidge(alpha=0.6, kernel='polynomial', degree=2, coef0=2.5)
```

- 梯度提升回归

[37]:
```
GBoost = GradientBoostingRegressor(n_estimators=1000, learning_rate=0.
↪05,
max_depth=4, max_features='sqrt',
min_samples_leaf=15, min_samples_split=10,
loss='huber', random_state =5)
```

- XGBoost

[38]:
```
model_xgb = xgb.XGBRegressor(colsample_bytree=0.5, gamma=0.05,
learning_rate=0.05, max_depth=3,
min_child_weight=1.8, n_estimators=1000,
reg_alpha=0.5, reg_lambda=0.8,
subsample=0.5, silent=1,
random_state =7, nthread = -1)
```

- LightGBM

[39]:
```
model_lgb = lgb.LGBMRegressor(objective='regression',num_leaves=5,
learning_rate=0.05, n_estimators=1000,
max_bin = 55, bagging_fraction = 0.8,
bagging_freq = 5, feature_fraction = 0.2,
feature_fraction_seed=9, bagging_seed=9,
min_data_in_leaf =6, min_sum_hessian_in_leaf = 11)
```

以上方法的评分

[40]:
```
score = rmsle_cv(lasso)
print("\nLasso score: {:.4f} ({:.4f})\n".format(score.mean(), score.
 ↪std()))
score = rmsle_cv(ENet)
print("ElasticNet score: {:.4f} ({:.4f})\n".format(score.mean(),␣
 ↪score.std()))
score = rmsle_cv(KRR)
print("Kernel Ridge score: {:.4f} ({:.4f})\n".format(score.mean(),␣
 ↪score.std()))
score = rmsle_cv(GBoost)
print("Gradient Boosting score: {:.4f} ({:.4f})\n".format(score.
 ↪mean(), score.std()))
score = rmsle_cv(model_xgb)
print("Xgboost score: {:.4f} ({:.4f})\n".format(score.mean(), score.
 ↪std()))
score = rmsle_cv(model_lgb)
print("LGBM score: {:.4f} ({:.4f})\n" .format(score.mean(), score.
 ↪std()))
```

```
Lasso score: 0.1104 (0.0055)

ElasticNet score: 0.1103 (0.0054)

Kernel Ridge score: 0.1196 (0.0085)

Gradient Boosting score: 0.1143 (0.0077)
```

```
Xgboost score: 0.1175 (0.0069)

LGBM score: 0.1160 (0.0054)
```

4.4 Stacking 方法

集成学习往往能进一步提高模型的准确性，Stacking 是其中一种效果颇好的方法，简单来说就是学习各个基本模型的预测值来预测最终的结果。这里用 ENet、KRR 和 GBoost 作为第一层学习器，用 Lasso 作为第二层学习器：

[41]:
```
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)
```

Stacking 的交叉验证评分：

[42]:
```
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: 0.1072 (0.0054)
```

4.5 建立最终模型

将 XGBoost、LightGBM 和 StackedRegressor 以加权平均的方式融合在一起，建立最终的预测模型。先定义一个评价函数：

[43]:
```
def rmsle(y, y_pred):
return np.sqrt(mean_squared_error(y, y_pred))
```

用整个训练集训练模型，预测测试集的房价，并给出模型在训练集上的评分。

- StackedRegressor

[44]:
```
stacked_averaged_models.fit(train, y_train)
stacked_train_pred = stacked_averaged_models.predict(train)
stacked_pred = np.expm1(stacked_averaged_models.predict(test))
print(rmsle(y_train, stacked_train_pred))
```

```
0.08464516368336122
```

- XGBoost

[45]:
```
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))
```

```
0.08294534243412438
```

- LightGBM

[46]:
```
model_lgb.fit(train, y_train)
lgb_train_pred = model_lgb.predict(train)
lgb_pred = np.expm1(model_lgb.predict(test))
print(rmsle(y_train, lgb_train_pred))
```

```
0.0636622923292498
```

融合模型的评分：

[47]:
```
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))
```

```
RMSLE score on train data:
0.07934565869572287
```

4.6 预测

[48]:
```
ensemble = stacked_pred*0.70 + xgb_pred*0.15 + lgb_pred*0.15
```

4.7 生成提交文件

[49]:
```
sub = pd.DataFrame()
sub['Id'] = test_ID
sub['SalePrice'] = ensemble
sub.to_csv('submission.csv', index=False)
```

第 5 章　预测结果

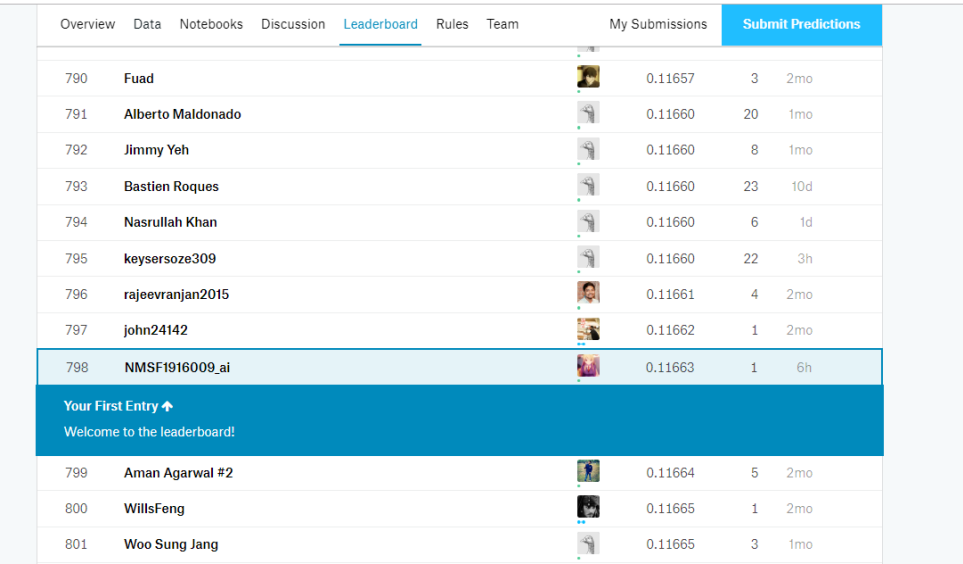


图 **5-1**

Score:0.11663

ID:NMSF1916009_ai

Rank:798

参考文献

[1] dpwang.windows 下使用 conda 安装 xgboost 问题-[EB/OL][2019 - 12 - 14].https://blog.csdn.net/dpengwang/article/details/89433034.

[2] 佚名. Stacking 算法 [EB/OL]([2019-12-15]. https://www.jianshu.com/p/59313f43916f.

[3] 01 编程之路.sklearn 库中数据预处理函数 fit_transform() 和 transform() 的区别 [EB/OL][2019-12-14].https://blog.csdn.net/qq_30638831/article/details/80200684.

[4] LeatherWang 的博客.QQ 图判断一个分布是否为正态分布 [EB/OL][2019 - 12 - 15].https://blog.csdn.net/hzwwpgmwy/article/details/79178485.

[5] 打江南走过一阵.《numpy.ndarray》object has no attribute《values》-[EB/OL][2019-12-15].https://blog.csdn.net/FYZ530357172/article/details/73188039.

[6] 去向前方的博客.Anaconda：包安装以 XGBoost 为例 [EB/OL][2019 - 12 - 14].https://blog.csdn.net/lvsehaiyang1993/article/details/80619495.

[7] 素质云笔记. 结构化数据转换方式之一：box-cox 转换 [EB/OL][2019 - 12 - 15].https://blog.csdn.net/sinat_26917383/article/details/77864582.

[8] 佚名. 从 0 到 1 走进 Kaggle[EB/OL][2019 - 12 - 15].https://zhuanlan.zhihu.com/p/61660061.

[9] COCK D, DEAN. Ames, iowa: alternative to the boston housing data as an end of semester regression project[J]. Journal of Statistics Education, 19(3): 8.