《机器学习与预测》

课程结课报告

基于机器学习的房屋价格预测

专 业 名 称 ：管理科学

课 程 名 称 ：机器学习与预测

指 导 教 师 ：陈植元

学 生 学 号 ：2021301052056

学 生 姓 名 ：陈凯强

二○二四年五月

目录

摘要

本文旨在探索基于机器学习的房屋价格预测方法。通过对 Kaggle 平台提供的数据集进行特征分析和特征工程，处理了包括缺失值、离群值和偏态等问题。数据集包含 81 个变量，主要描述了爱荷华州艾姆斯地区房屋的各种特征。我们筛选了偏度大于 0.75 的变量并进行了 Box-Cox 变换，优化了数值型变量到分类变量的转换。

在机器学习模型的选择和优化过程中，我们采用了套索回归、核岭回归、弹性网络、线性回归和 SVM 等 5 种基本模型，并通过五折交叉验证评估其性能。结果显示，套索回归和弹性网络表现最佳。接下来，我们尝试了 Boosting 方法，包括梯度提升回归树（GBRT）、XGBoost 和 LightGBM，其中 LightGBM 效果最佳。为了进一步提升模型性能，我们对部分模型应用了 Bagging 算法，并尝试了随机森林，但 bagging 对模型效果提升并不显著。

最后，通过将效果较好的模型进行 Stacking 集成（ENet、KRR 和 GBoost 作为第一层学习器，Lasso 作为第二层学习器），成功将 RMSE 从 0.1226 降低至 0.1183。该成绩在 Kaggle 排名中可进入前 2%。

本文展示了特征工程和模型优化在提高房屋价格预测准确度中的关键作用，为实际应用提供了有价值的参考。

1. 研究背景

1.1 研究目的

房屋价格预测在房地产市场中具有重要意义，对于购房者、投资者和房地产开发商来说，准确预测房屋价格可以帮助他们做出更明智的决策。本文旨在通过应用先进的机器学习技术，对房屋价格进行准确预测，并探讨不同机器学习算法在这一问题上的表现和优化方法。

1.2 研究意义

本研究具有重要的实践和理论意义：

(1) 实践意义

a. 提高决策效率：通过准确的房价预测，购房者可以更合理地规划购房预算，投资者可以更有效地选择投资项目，开发商可以更科学地定价和营销策略。

b. 降低风险：精确的价格预测有助于识别市场趋势和波动，降低投资和经营风险，为各类市场参与者提供有价值的参考。

c. 推动智能化应用：将机器学习技术应用于房价预测领域，推动房地产行业向智能化、数据驱动的方向发展，提高行业整体效率和竞争力。

(2) 理论意义

a. 丰富机器学习应用场景：将多种机器学习算法应用于房价预测问题，探索其在实际问题中的表现和优化方法，为机器学习技术在其他领域的应用提供参考。

b. 推动特征工程研究：通过系统的特征工程实践，总结出有效的特征处理和选择方法，为后续研究提供借鉴。

c. 提升模型集成技术：研究并验证了多种模型集成技术的有效性，特别是在复杂预测问题中的应用，丰富了模型集成的理论和实践经验。

通过本研究，希望能够为房价预测提供有效的技术手段和方法，提高预测精度和可靠性，推动房地产行业的智能化和数据化转型。

2. 数据集来源和特征分析

2.1 数据来源

本文数据来源于 Kaggle 平台 (https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques)。该项目要求我们预测每套房屋最终的销售价格。

2.2 数据集和特征向量介绍

数据集共有 81 个变量，1460 个样本。共有 1 个被解释变量和 79 个解释变量。被解释变量为 SalePrice，即房屋价格。被解释变量描述了爱荷华州艾姆斯（Ames, Iowa）地区房屋的众多特征。

表 1 数据集中所有的变量及其含义

变量	变量含义	变量	变量含义
SalePrice	房屋销售价格（被解释变量）	HeatingQC	加热质量和条件
MSSubClass	住宅标识	CentralAir	是否有中央空调
MSZoning	住宅分区	Electrical	电气系统
LotFronitage	到街道距离	1stFlrSF	一层面积
LotArea	地段面积	2ndFlrSF	二层面积
Street	通往住宅 Street 类型	LowQualFinSF	所有楼层低质量面积
Alley	通往住宅 Alley 类型	GrLivArea	地上生活面积
LotShape	地段形状	BsmtFulllBath	地下室全浴室数量
. LandContour	平坦度	BsmtHalfBath	地下室半浴室数量

Utilities	设备可用性	FullBath	地上全浴室数量
LotConfig	地段配置	HalfBath	地上半浴室数量
LandSlope	住宅坡度	Bedroom	地上卧室数量
Neighborhood	街区位置	Kitchen	厨房数量
Condition1	靠近主要道路或铁路(条件 1)	KitchenQual	厨房质量
Condition2	靠近主要道路或铁路(条件 2)	TotRmsAbvGrd	地.上总房间数(不包括浴室)
BldgType	住宅类型	Functional	居家功能性
HouseStyle	住宅风格	Fireplaces	壁炉数量
OverallQual	整体材料和装饰综合质量评级	FireplaceQu	壁炉质量
OverallCond	整体情况评级	GarageType	车库位置
YearBuilt	建设时间	GarageYrBlt	车库建造年份
YearRemodAdd	改造时间	GarageFinish	车库完成情况
RoofStyle:	屋顶类型	GarageCars	车库容量
RoofMatl	屋顶材料	GarageArea	车库面积
Exterior1st	房子的外墙覆盖物	GarageQual	车库质量
Exterior2nd	房屋的外墙覆盖物（如果有第二种材料）	GarageCond	车库状况
: MasVnrType	砌面贴面类型	PavedDrive	车道情况
MasVnrArea	砌面贴面面积	WoodDeckSF	木制甲板面积
ExterQual	外部材料质量评级	OpenPorchSF	开放式阳台面积

ExterCond	外部材料当前状态评级	EnclosedPorch	封闭式门廊面积
Foundation	基础类型	3SsnPorch	三季门廊面积
BsmtQual	地下室高度评估	ScreenPorch	屏幕门廊面积
BsmtCond	地下室状况评估	PoolAreai	泳池面积
BsmtExposure	地下室曝光	PoolQC	泳池质量
BsmtFinType1	第一个地下室完成区域的等级	Fence	围栏质量
BsmtFinSF1	类型 1 地下室完成区域面积	MiscFeature	其他未涵盖项
BsmtFinType2	第二个地下室完成区域的等级(如果存在)	MiscVal	其他未涵盖项的值
. BsmtFinSF2	类型 2 地下室完成区域面积	MoSold	开售月份
BsmtUnfSF	地下室未完成面积	YrSold	开售年份
TotalBsmtSF	地下室总面积	SaleType	销售类型
Heating	加热类型	SaleCondition	销售条件

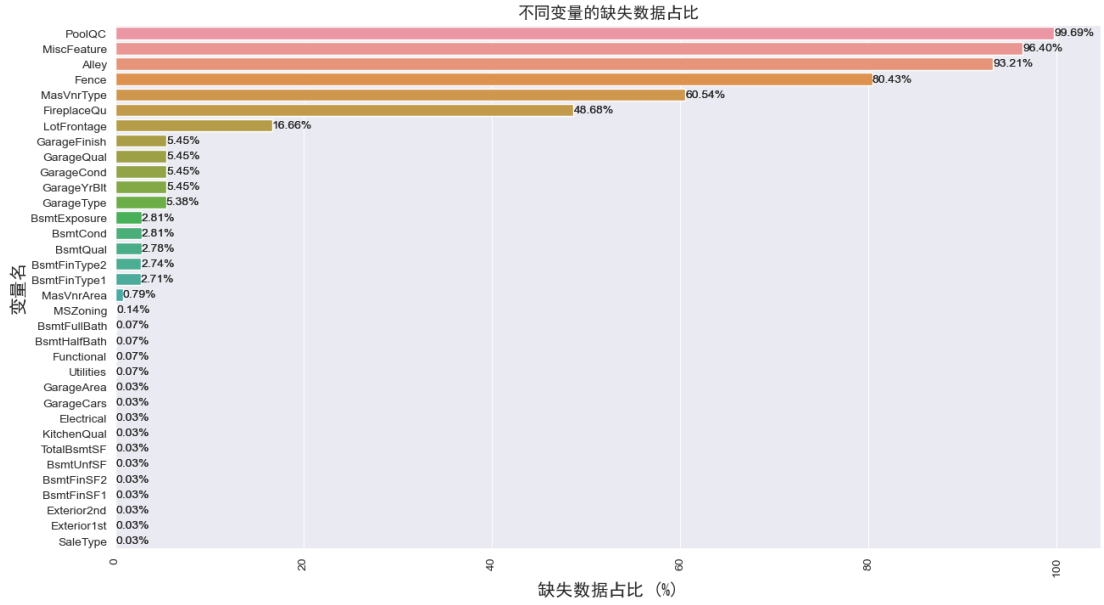
2.3 数据预处理

接下来进行数据预处理以方便进一步的特征工程和建模。主要包括：查看并处理缺失值、离群值、对数变换、数值型特征变换、分类型特征变换。

2.3.1 缺失值

首先对数据集数据完备情况进行探索。笔者画出了不同变量缺失数据占比的柱状图。从图中可以看出，共有 25 个含缺失值的特征，其中前五大特征的缺失数据占比均超过 50%。

图 1 不同变量缺失数据占比柱状图



经过数据探索可以发现，前五大特征是所以缺失率如此之高，是因为该数据集中用缺失值来表示该特征的一种情况。如数据描述提及对于 PoolQC 变量，NA 表示“没有泳池”，1 表示“有泳池”。这符合大多数房子没有游泳池这一常识，因而缺失值占比高达 99.69%。对于此，直接用 0 填充缺失值即可。

同样地，笔者对其他所有缺失值的缺失原因进行了研究，将缺失值类型分类如下，并针对不同类型采取了不同的缺失值处理策略。

表 2 对不同缺失值的分类和处理策略

缺失值种类和举例	处理策略
缺失是因为房子没有的特征（前五大缺失特征）	用 0 替换缺失值
缺失数量较少的离散型特征	用众数填补缺失值
方差很小的特征：Utilities（设备可用性），所有记录除了一个“NoSeWa”和 2 个 NA ，其余值都是“AllPub”	删除该特征
数据描述中有说明的特征：Functional（居家功能性）	数据描述提及 NA 表示类型“Typ”，因此用这个填充
其他：LotFrontage （到街道的距离）	通过该社区的中位数来填充缺失值

2.3.2 离群值处理

笔者挑选了几个变量绘制了散点图，发现 GrLivArea 有比较明显的离群值。总体趋势是 SalePrice（房屋售价）随着 GrLivArea（地上生活面积）的增加而增加，这也符合常识。而右下角的两个离群值显然不符合该趋势。

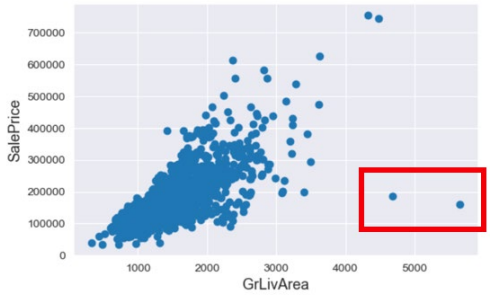


图 2 GrLivArea 的分布图（红框为离群值）

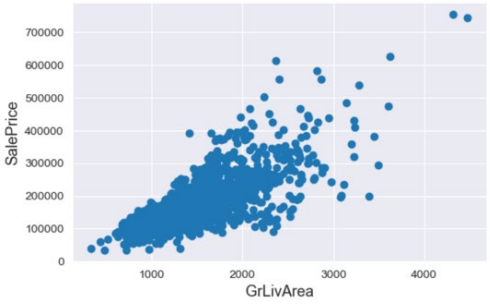


图 3 删除离群值后的分布图

然而，由于变量种类过多，我们无法发现并删除所有的离群值。同时，笔者认为保留一定的离群值有利于提高训练结果的鲁棒性，因此对于离群值并不做过多的处理，而是保留一定的噪声。

2.3.3 目标变量偏态处理

对于许多机器学习算法，目标变量的正态分布可以提高模型的性能和稳定性。正态分布的目标变量可以使模型更容易拟合，减少过拟合的风险，并且在某些情况下，可以提升模型的预测精度。同时，对于线性回归模型来说，需要服从正态性假设。因此，我们首先查看目标变量 SalePrice 的分布，发现偏态较为明显，因此对其常识进行对数变换，发现变换后基本服从正态分布，因此将变换后的 SalePrice 作为目标变量。

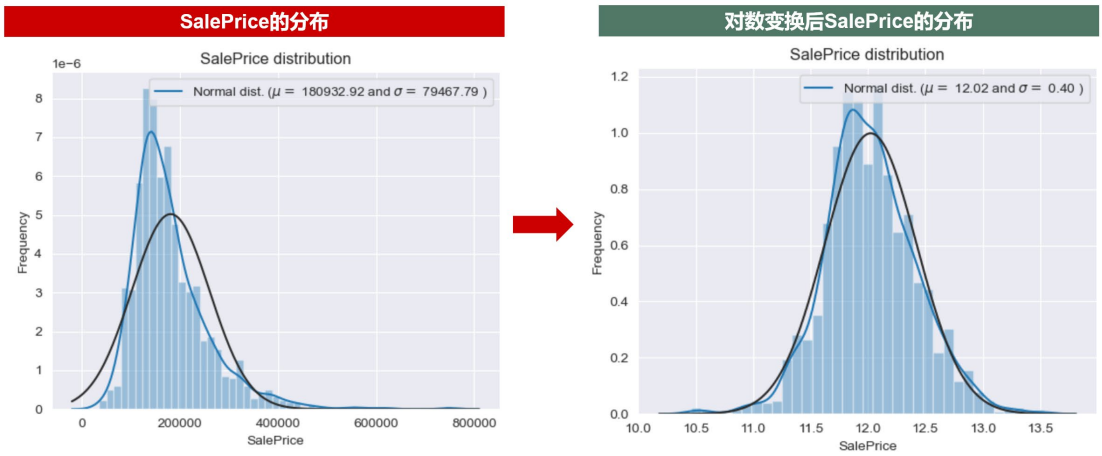


图 4 对 SalePrice 分布进行对数变换

2.4 特征工程

2.4.1 特征相关性探索

考虑到减少多重共线性、简化模型、进行特征组合和变换等需要，笔者利用 python 中的 seaborn、matplotlib 等库对各个变量的相关性进行了探索，初步得到了变量相关性热力图。

由于变量过多，直接画出的热力图难以辨别，因此我们简化情形，只考虑相关性大于 0.8 的变量之间的强相关性，因此绘制出了第二个变量热力图。

从图中我们可以发现，YearBuilt & GarageYrBlt, GarageCars &GarageArea 等 4 组变量的相关性非常高。通过变量含义来看，YearBuilt 和 GarageYrBlt 分别表示房屋建筑年龄和地下车库建筑年龄，两者显然具有高度的相关性；同理 GarageCars 表示地下车库容量，CarageArea 表示地下车库面积，两者显然也具有高度的相关性。因此我们考虑在后续建模的过程中可以删掉一些高相关性的变量中的一个。

此外，笔者还探究了哪些变量与目标变量高度相关，输出了相关性最高的前五大特征。可见，OverallQual（整体材料和装饰综合质量评级）、GrLivArea（地上生活面积）、GarageCarsArea（车库容量）、GarageArea（车库面积）和 TotalBsmtSF（地下室总面积）是和目标变量最相关的五大特征。**因此，房屋总体质量、地上面积、车库面积和地下室面积是影响房屋价格最重要的特征。**

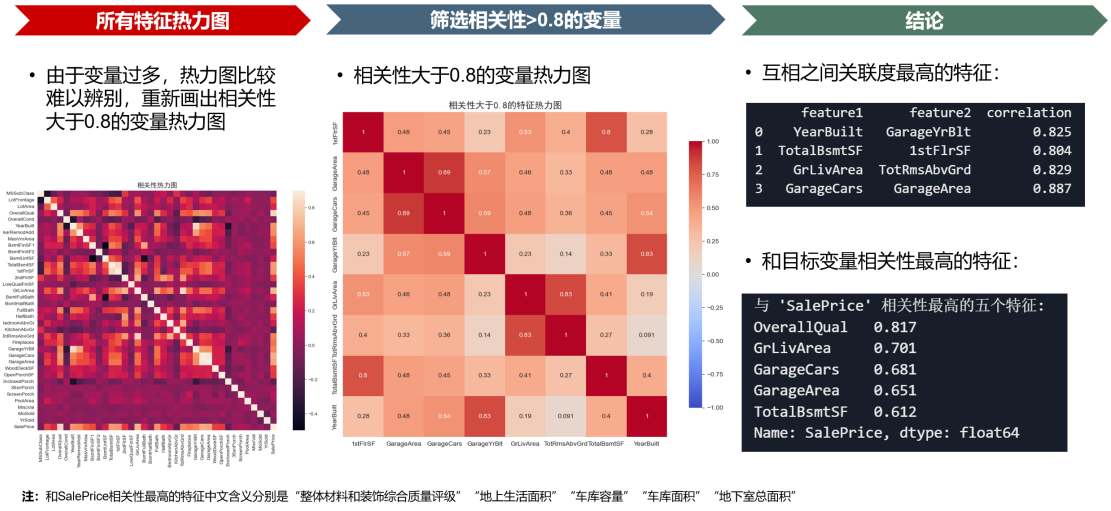


图 5 变量相关性热力图

2.4.2 数值型变量的处理

（1）数值型变量到分类型变量的转换

笔者发现，MSSubClas（住宅标识）、YrSold（开售年份）、MoSold（开售月份）等特征虽然是数值特征，但其实只表示类别，数值大小不代表实际意义。如 MSSubClass（住宅标识）虽然是数值型，但这些数值代表的是不同类型的住宅。直接将这些数值作为连续变量使用会使模型认为它们有顺序和大小关系，但实际上它们只是不同的类别。YrSold（开售年份）和 MoSold（开售月份）更合适被看作是时间类别。例如，YrSold 代表的只是不同的年份，并没有顺序上的意义；MoSold 代表的是月份，也没有大小之分。因此，为了减少噪声，使模型更清晰地理解每个变量，提高模型的准确度和可解释性，有必要对这种数值型特征进行变换，将其转换为没有数值大小之分的分类型特征。

分类特征可以采用不同的编码方式，如独热编码（One-Hot Encoding）、目标编码（Target Encoding）等。这里笔者将部分数值型变量用 astype 转换为字符串。

（2）Box-Cox 变换

为了满足使数据更加对称，提高拟合效果，符合线性假设，笔者对高偏态的数据进行了 Box-Cox 变换。

首先，笔者利用以下的偏度计算公式，调用 Python 中的 scipy 中 boxcox1p() 对各个变量的偏度进行了计算：

$$S=\frac{n}{(n-1)(n-2)}\Sigma_{i=1}^{n}\left[\left(\frac{X_i-\mu}{\sigma}\right)^3\right] \tag{1}$$

其中，n 是样本个数，μ 是均值，σ 是标准差。

得到偏度>0.75 的变量如下：

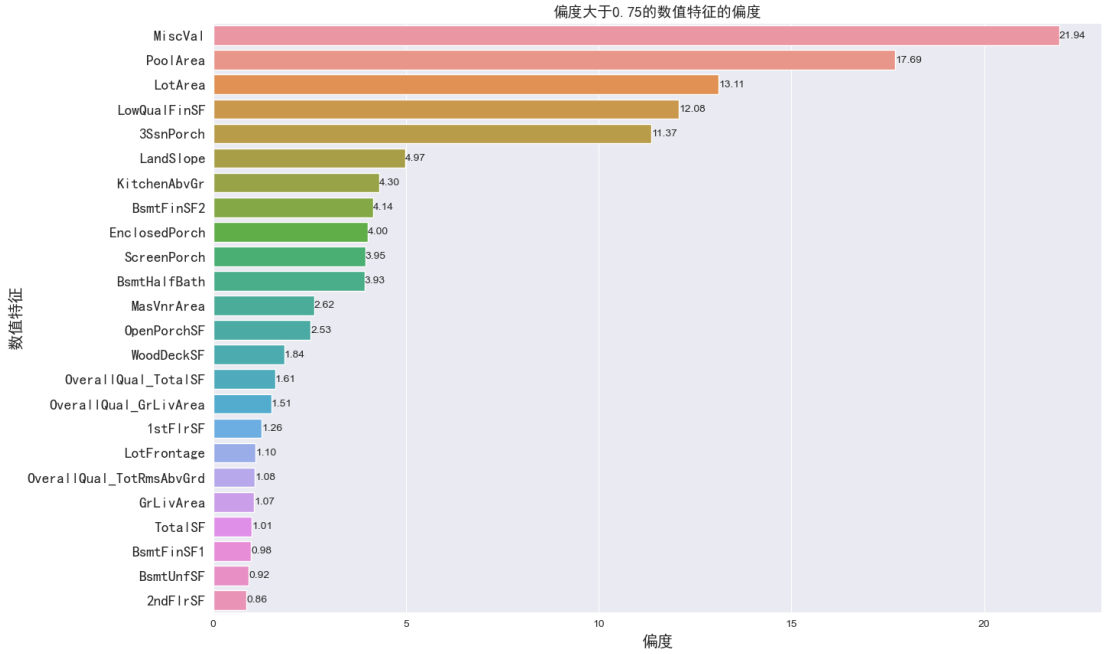


图 6 偏度大于 0.75 的数值特征及其偏度

接着将大于 0.75 的偏度平滑至 0.15。

```
new_skewness = skewness[skewness.abs() > 0.75]
print("有{}个高偏度特征被Box-Cox变换".format(new_skewness.shape[0]))
```

✓ 0.0s

```
有63个高偏度特征被Box-Cox变换
```

图 7 Box-Cox 变换代码

2.4.3 分类型变量的处理

（1）数值型变量到分类型变量的转换

与数值型变量的情况相对应，有的分类型变量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，如 PoolQC（泳池质量），有 Ex （Excellent 优秀）Gd（Good 良好） 等等，因此，需要将特征的类别映射

成数值大小，来表现这种潜在的偏序关系。这里笔者用标签编码 (LabelEncoder) 来将数值型变量准换成标签型变量。

（2）对属性数据进行独热编码

对于类别特征，我们将其转化为独热编码，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。利用 pandas 中的 get_dummies 函数。

2.4.4 新特征的构造

利用原有特征合理构造新特征。如利用地下室面积、地上生活面积、车库面积构造房屋总面积这一新特征。

```
# 构造更多的特征
all_data['TotalSF'] = all_data['TotalBsmtSF'] + all_data['1stFlrSF'] + all_data['2ndFlrSF'] # 房屋总面积

all_data['OverallQual_TotalSF'] = all_data['OverallQual'] * all_data['TotalSF']  # 整体质量与房屋总面积交互项
all_data['OverallQual_GrLivArea'] = all_data['OverallQual'] * all_data['GrLivArea'] # 整体质量与地上总房间数交互项
all_data['OverallQual_TotRmsAbvGrd'] = all_data['OverallQual'] * all_data['TotRmsAbvGrd'] # 整体质量与地上生活面积交互项
all_data['GarageArea_YearBuilt'] = all_data['GarageArea'] * all_data['YearBuilt'] # 车库面积与建造时间交互项
```

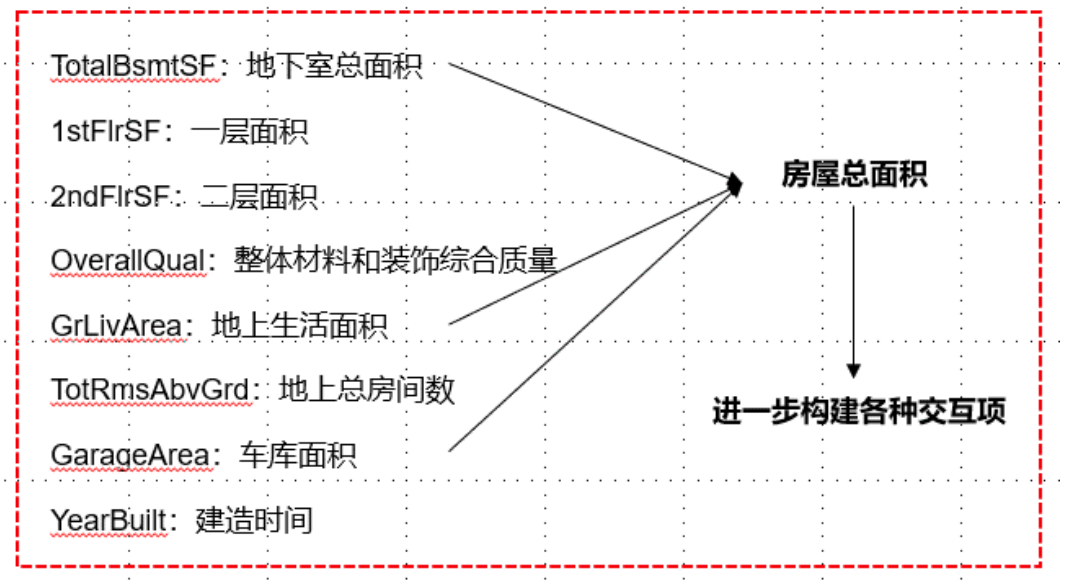


图 8 构造新特征

3. 机器学习

3.1 建立交叉验证函数

首先，建立五折交叉验证函数。根据比赛评价标准，用 Root-Mean-Squared-Error (RMSE)来为每个模型打分。使用 sklearn 中的 cross_val_score()函数，但是这个函数没有 shuffle 属性，所以添加一行代码，在交叉验证之前对数据集进行 shuffle。

- 采用**K折交叉验证 (取K=5)**，将数据集划分成5个大小相同的互斥子集，每次取4份做训练集，剩下一份做测试集，最终选择损失函数评估最优的模型和参数.
  - **优点：**所有数据都参与训练中，增加模型泛化性，避免划分训练集和测试集只进行一次划分的随机性
  - **K的选取：**比较复杂，实验表明，正常情况下可以选择5或者10，能达到不错的效果。 因此这里选择了5

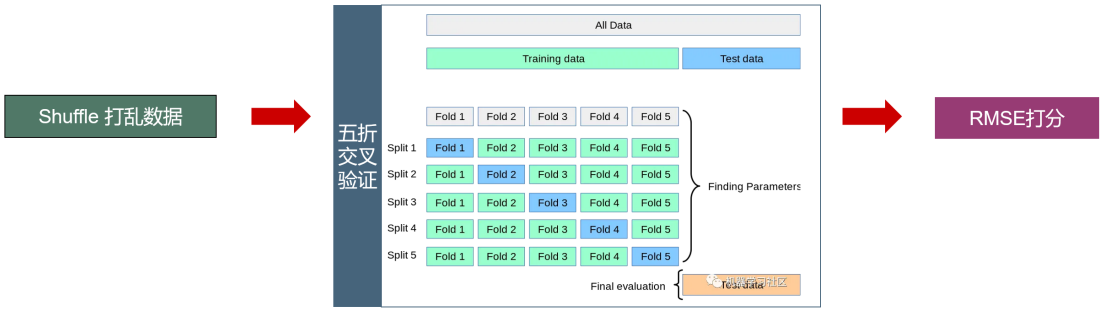


图 9 五折交叉验证示意图

```
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_numerical.values, y_train, scoring="neg_mean_squared_error", cv=kf))
    return(rmse)
✓ 0.0s
```

图 10 五折交叉验证代码

3.2 建立五大基本模型

3.2.1 基本模型（套索回归、核岭回归、弹性网络、线性回归、SVM）

笔者建立了套索回归、核岭回归、弹性网络、线性回归、SVM 这 5 种基本模型，并最终均调用前面建立的五折交叉验证函数。

```python
models = {'LinearRegression':linear,'Lasso': lasso, 'ElasticNet': ENet, 'Kernel Ridge': KRR,'SVM':svm_sigmoid}
results=[]
for model_name, model in models.items():
    score = rmsle_cv(model)
    results.append([model_name, round(score.mean(),4), score.std()])

# 创建 DataFrame
df_results = pd.DataFrame(results, columns=['Model', 'Mean Score', 'Std Dev'])

df_sorted = df_results.sort_values(by='Mean Score', ascending=True)
pd.set_option('display.float_format', '{:.4f}'.format)
# 显示排序后的结果
print("Sorted Results:")
print(df_sorted)
```

图 11 五种基本线性回归调用五折交叉验证代码

得到各个模型 RMSE 得分的均值和标准差汇总如下：

```
Sorted Results:
                Model  Mean Score  Std Dev
1               Lasso      0.1226   0.0067
2          ElasticNet      0.1226   0.0067
0  LinearRegression      0.1232   0.0069
4                 SVM      0.1235   0.0063
3        Kernel Ridge      0.1325   0.0065
```

图 12 基本模型 RMSE 结果汇总

可以看出 Lasso 回归和弹性网络的 RMSE 最低，且标准差也很低，效果最好，核岭回归在这里的效果最差。

3.2.2 模型调参

示例：尝试了三种不同的 svm 参数，得到的效果排序如下：

```
Sorted Results:
          Model  Mean Score  Std Dev
1   SVM_sigmoid      0.1235   0.0063
2    svm_linear      0.1235   0.0063
0       SVM_rbf      0.1407   0.0066
```

图 13 三种 svm 参数比较

可见 sigmoid 参数的效果最好，rbf 的效果最差。因此在模型间比较时将只保留 SVM_rbf 作为 SVM 模型的代表。

还有很多别的调参尝试，由于篇幅这里不一一列举。

4. 集成学习

4.1 Boosting

首先尝试了 boosting，引入梯度提升回归 GBRT、XBGoost、LightGBM

梯度提升回归树(GBRT)算法是以 CART 回归树为基学习器的集成学习模型，通过迭代多棵回归树生成多个弱模型，然后将每个弱模型的预测结果相加，通过不断减小损失函数的值来调整模型。与其他集成学习算法相比，梯度提升回归树对参数设置更为敏感。

极致梯度提升（XGBoost）是基于 GBRT 的一种改进算法，通过引入模型复杂度重新构造目标函数，在迭代每一棵树的过程中都力求最小化，同时最小化了模型的损失函数即损失率和模型的复杂度，从而来提高算法的运算效率。

LightGBM 也是基于 GBRT 算法的一种优化改进，其改进之一是采用了基于直方图的决策层树算法，对每个特征的取值做分段函数，将所有样本在该特征上的取值划分到某一段中，最终把特征取值离散化；改进之二是采用 Leaf-wise 的生长策略，使增益高的节点优先生长，从而减少低增益点浪费计算资源的问题，提高计算效率。

引入这三种 boosting 模型后所有模型排序效果如下：

Sorted Results:

	Model	Mean Score	Std Dev
1	Lasso	0.1226	0.0067
2	ElasticNet	0.1226	0.0067
0	LinearRegression	0.1232	0.0069
4	SVM	0.1235	0.0063
7	LightGBM	0.1236	0.0077
5	Gradient Boosting	0.1239	0.0079
6	XGBoost	0.1247	0.0059
3	Kernel Ridge	0.1325	0.0065
8	RandomForest	0.1389	0.0051

图 14 基本模型+boosing 结果汇总

可见 boosting 模型中 LightGBM 的效果最好，XGBoost 的效果最差。

4.2 Bagging

对 lasso 和 XBGoost 尝试 bagging 算法，并且添加了随机森林算法，但最终效果并不明。此时最小 RMSE 为 0.1226。

```
from sklearn import linear_model
from sklearn.ensemble import RandomForestRegressor

from sklearn.ensemble import BaggingRegressor
linear=linear_model.LinearRegression()


from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor(n_neighbors=5)

rf = RandomForestRegressor(n_estimators=100,  # 决策树的数量
                           max_depth=None,    # 树的最大深度
                           min_samples_split=2, # 拆分内部节点所需的最小样本数
                           random_state=42)   # 随机种子以确保重现性

bagging_lasso = BaggingRegressor(base_estimator=lasso,
                          n_estimators=25,  # 使用25个lasso模型
                          max_samples=0.5,  # 每个模型使用50%的数据
                          max_features=0.8,  # 每个模型使用80%的特征
                          random_state=42)   # 随机种子以确保重现性

bagging_xgb = BaggingRegressor(base_estimator=model_xgb,
                          n_estimators=25,  # 使用25个XGBoost模型
                          max_samples=0.5,  # 每个模型使用50%的数据
                          max_features=0.8,  # 每个模型使用80%的特征
                          random_state=42)   # 随机种子以确保重现性
```

图 15 bagging 部分代码

Sorted Results:

	Model	Mean Score	Std Dev
1	Lasso	0.1226	0.0067
2	ElasticNet	0.1226	0.0067
0	LinearRegression	0.1232	0.0069
11	BaggingLasso	0.1232	0.0070
6	svm_sigmoid	0.1235	0.0063
7	svm_linear	0.1235	0.0063
12	LightGBM	0.1236	0.0077
8	Gradient Boosting	0.1239	0.0079
9	XGBoost	0.1247	0.0059
10	BaggingXGB	0.1273	0.0058
3	Kernel Ridge	0.1325	0.0065
13	RandomForest	0.1389	0.0051
5	svm_rbf	0.1407	0.0066
4	KNN	0.2082	0.0092

图 16 基本模型及 bagging、boosting 后效果汇总

4.3 Stacking 模型组合优化

将效果较好的模型进行 stacking 集成。用 ENet、KRR 和 GBoost 作为第一层学习器，用 Lasso 作为第二层学习器。Stacking 后的交叉验证评分如下。RMSE 成功降低至 0.1183。

```
1  stacked_averaged_models = StackingAveragedModels(base_models=(ENet, linear, model_lgb), meta_model=lasso)
2
3  score = rmsle_cv(stacked_averaged_models)
4  print("Stacking Averaged models score: {:.4f} ({:.4f})".format(score.mean(), score.std()))
[163] ✓ 5.1s
... Stacking Averaged models score: 0.1183 (0.0071)
```

图 17 stacking 集成部分代码和结果

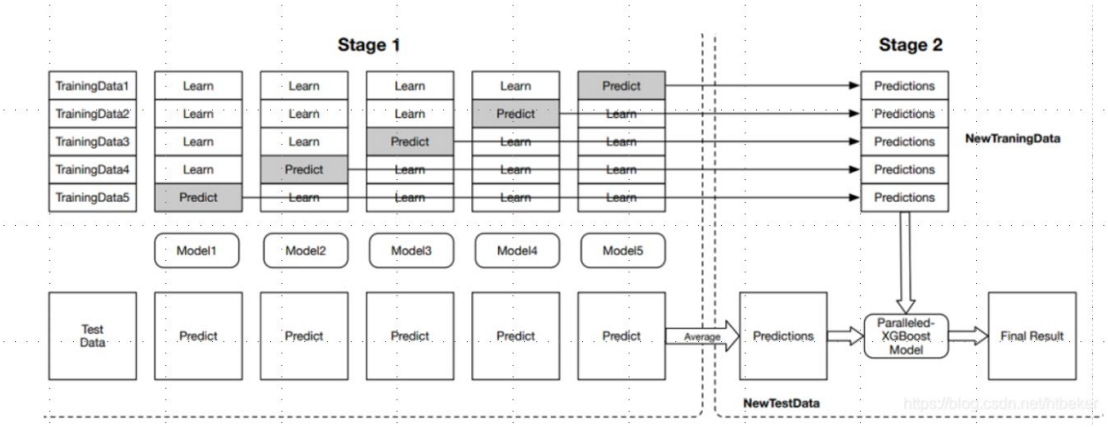


图 18 Stack 流程示意图

对比 Kaggle 网站上的排名，这一成绩可以进入前 2%。

5. 总结与展望

本文通过系统的特征工程和多种机器学习算法的应用，实现了对房屋价格的有效预测。在研究过程中，我们对数据进行了全面的预处理，包括处理缺失值、离群值和对数变换等操作，确保了数据的质量和一致性。通过对特征的深入分析和处理，我们选择了对预测最有意义的特征，并采用多种编码和变换方法提升了模型的表现。

在模型构建和优化方面，我们尝试了多种基本模型，如套索回归、核岭回归、弹性网络、线性回归和支持向量机，并通过五折交叉验证评估了它们的性能。在此基础上，我们引入了 Boosting（GBRT、XGBoost、LightGBM）和 Bagging 算法进行模型集成，并进一步优化模型效果。最终，通过 Stacking 方法将效果较好的模型进行集成，取得了显著的性能提升，成功降低了 RMSE，达到了 0.1183。这一成绩在 Kaggle 比赛中表现优异，排名进入前 2%。

尽管本研究取得了一定的成果，但仍存在一些可以进一步改进和探索的方向：

1. **数据集扩展和多样化**：本研究主要基于 Kaggle 平台提供的数据集，未来可以引入更多的多样化数据源，如不同地区和不同时间段的房屋数据，以验证模型的泛化能力和适用性。
2. **特征工程的深入研究**：特征工程在机器学习中起到了关键作用，未来可以探索更多的特征选择和生成方法，如自动特征工程工具和更复杂的特征组合策略，以提升模型的预测能力。
3. **实时预测和在线学习**：随着房地产市场的动态变化，构建能够实时更新和预测的在线学习模型，捕捉市场的最新趋势和变化，提高预测的时效性和准确性。
4. **跨领域应用**：机器学习在房价预测中的成功应用可以推广到其他领域，如股票价格预测、市场需求预测和经济指标预测等，验证和扩展机器学习技术的适用范围。

总的来说，本研究通过系统的特征工程和模型优化，实现了对房屋价格的高精度预测，为房地产行业的智能化和数据化转型提供了技术支持。未来，我们将继续探索和优化这一研究领域，推动机器学习技术在更多实际问题中的应用和发展。

6. 附：Python 代码

```
import warnings
warnings.filterwarnings("ignore")     # 忽略警告信息

from scipy.stats import norm, skew    # 获取统计信息
from scipy import stats
import numpy as np
import pandas as pd       # 数据处理包
import seaborn as sns    # 绘图包
color = sns.color_palette()
sns.set_style('darkgrid')
import matplotlib.pyplot as plt
from matplotlib.font_manager import FontProperties

# 定义字体
font = FontProperties(fname=r"C:\Windows\Fonts\SimHei.ttf", size=14)    # Windows
下的字体路径


%matplotlib inline


pd.set_option('display.float_format', lambda x: '{:.3f}'.format(x))    # 限制浮点输出
到小数点后 3 位

import os
print('\n'.join(os.listdir('./input')))    # 列出目录中可用的文件

# %% [markdown]
# ### 加载数据集

# %%
# 加载数据
train = pd.read_csv('./input/train.csv')
test = pd.read_csv('./input/test.csv')

# %%
# 查看样本和特征的数量
print(train.shape)
print(test.shape)
```

```
# %%
train.head(5)

# %%
test.head(5)

# %%
#  保存 Id 列
train_ID = train['Id']
test_ID = test['Id']

#  删除原数据集的 Id 列
train.drop("Id", axis=1, inplace=True)
test.drop("Id", axis=1, inplace=True)

# %% [markdown]
# ###  数据预处理和特征工程

# %% [markdown]
#  异常值处理；目标变量分析；缺失值处理；特征相关性；进一步挖掘特征；对特征进行 Box-Cox 变换；独热编码。

# %% [markdown]
# ####  异常值处理

# %% [markdown]
#  通过绘制散点图可以直观地看出训练集特征是否有离群值，这里以地上生活面积  GrLivArea  为例。
#  我们可以看到，图像右下角的两个点有着很大的 GrLivArea，但相应的 SalePrice 却异常地低，我们有理由相信它们是离群值，要将其剔除。

# %%
plt.figure(figsize=(14, 4))

plt.subplot(121)
plt.scatter(x=train['GrLivArea'], y=train['SalePrice'])
plt.ylabel('SalePrice', fontsize=13)
plt.xlabel('GrLivArea', fontsize=13)

train = train.drop(train[(train['GrLivArea'] > 4000) & (train['SalePrice'] < 300000)].index)
plt.subplot(122)
```

```
plt.scatter(train['GrLivArea'], train['SalePrice'])
plt.ylabel('SalePrice', fontsize=13)
plt.xlabel('GrLivArea', fontsize=13)


fig, ax = plt.subplots(nrows=2, figsize=(6, 10))
sns.distplot(train['SalePrice'], fit=norm, ax=ax[0])
(mu, sigma) = norm.fit(train['SalePrice'])
ax[0].legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)],
loc='best')
ax[0].set_ylabel('Frequency')
ax[0].set_title('SalePrice distribution')
# QQ 图
stats.probplot(train['SalePrice'], plot=ax[1])
plt.show()

# %% [markdown]
# 可以看到，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。我们**对其做对数变换**，让数据接近正态分布。

# %%
# 使用 numpy 中函数 log1p()将 log(1+x)应用于列的所有元素
train["SalePrice"] = np.log1p(train["SalePrice"])

# 查看转换后数据分布
# 分布图
fig, ax = plt.subplots(nrows=2, figsize=(6, 10))
sns.distplot(train['SalePrice'], fit=norm, ax=ax[0])
(mu, sigma) = norm.fit(train['SalePrice'])
ax[0].legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(
    mu, sigma)], loc='best')
ax[0].set_ylabel('Frequency')
ax[0].set_title('SalePrice distribution')

# QQ 图
stats.probplot(train['SalePrice'], plot=ax[1])
plt.show()

# %% [markdown]
# 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，所以我们在本地测试的时候也应该用同样的标准。
```

```
# %% [markdown]
# #### 缺失值处理

# %% [markdown]
# 1. 首先将训练集和测试集合并在一起

# %%
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)) # print(f"all_data size is : {all_data.shape}")

# %% [markdown]
# 2. 统计各个特征的缺失情况

# %%
all_data_na = (all_data.isnull().sum() / len(all_data)) * 100
all_data_na


# %%
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(10)

# %%
missing_data.shape[0]

# %%
fig, ax = plt.subplots(figsize=(15, 8))
barplot = sns.barplot(x=all_data_na.values, y=all_data_na.index, ax=ax)
sns.barplot(x=all_data_na, y=all_data_na.index)
plt.xticks(rotation=90)
plt.ylabel('变量名', fontsize=15,fontproperties=font)
plt.xlabel('缺失数据占比 (%)', fontsize=15,fontproperties=font)
plt.title('不同变量的缺失数据占比', fontsize=15,fontproperties=font)
for container in barplot.containers:
    barplot.bar_label(container, fmt='%.2f%%')

# %% [markdown]
# 3. 填补缺失值
```

```
# %% [markdown]
# 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，对于这种情况我们统一用“None”或者“0”来填充。\
# Python 中的 None 是一个特殊常量，不是 0，也不是 False，不是空字符串，更多的是表示一种不存在，是真正的空。它就是一个空的对象，只是没有赋值而已。
#

# %% [markdown]
# - **PoolQC**：数据描述说 NA 表示“没有泳池”。 这是有道理的，因为大多数房子一般都没有游泳池，因而缺失值的比例很高（+99%）。
# - **MiscFeature**：数据描述说 NA 表示“没有杂项”。
# - **Alley**：数据描述 NA 表示“没有小巷通往”。
# - **Fence**：数据描述说 NA 表示“没有围栏”。
# - **FireplaceQu**：数据描述说 NA 的意思是“没有壁炉”。
# .....

# %%
# 用 None 或 0 填补 25 个含缺失值特征
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')

# %% [markdown]
# 对于缺失较少的离散型特征，比如 Electrical，可以用众数填补缺失值。

# %%
# 用众数填补 6 个缺失较少的离散型特征缺失值
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])

# %% [markdown]
# - **LotFrontage**：由于房屋到街道的距离，最有可能与其附近其他房屋到街道的距离相同或相似，因此我们可以通过**该社区的 LotFrontage 中位数**来填充缺失值。

# %%
all_data["LotFrontage"] =
all_data.groupby("Neighborhood")["LotFrontage"].transform(lambda x:
x.fillna(x.median()))

# %% [markdown]
# - **Functional**：居家功能性，数据描述说 NA 表示类型"Typ"。因此，我们用其填充。

# %%
all_data["Functional"] = all_data["Functional"].fillna("Typ")

# %% [markdown]
# - **Utilities**：设备可用性，对于这个特征，所有记录除了一个“NoSeWa”和 2 个 NA ，其余值都是“AllPub”，因此该项特征的方差非常小。 并且带有“NoSewa”的房子在训练集中，也就是说测试集都是“AllPub”,这个特征对预测建模没有帮助。因此，我们可以安全地删除它。

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

# %% [markdown]
# 4. 最后确认缺失值是否已全部处理完毕

# %%
all_data.isnull().sum().max()

# %% [markdown]
# #### 特征相关性
#
# **相关性矩阵热图**可以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。

```
# %%
train

# %%
# seaborn 中函数 heatmap()可以查看特征如何与 SalePrice 相关联
# corrmat = train.corr()
numerical_train = train.select_dtypes(include=[np.number])

# 计算相关性矩阵
corrmat = numerical_train.corr()
plt.figure(figsize=(12, 9))
sns.heatmap(corrmat, vmax=0.9, square=True)
plt.title('相关性热力图',fontproperties=font)

# %%
numerical_train.shape

# %%
saleprice_corr = corrmat['SalePrice'].sort_values(ascending=False)

# 打印与 'SalePrice' 相关性最高的五个特征（包括 'SalePrice' 自身）
top_features = saleprice_corr.head(6)  # 选取前六个因为第一个是 'SalePrice' 自身

print("与 'SalePrice' 相关性最高的五个特征:")
print(top_features[1:])  # 排除 'SalePrice' 自身


# %%
high_corr = (corrmat.where(np.triu(np.abs(corrmat) > 0.8, k=1))
                    .stack()
                    .reset_index())
high_corr.columns = ['feature1', 'feature2', 'correlation']
print(high_corr)
# 筛选出相关性高于 0.8 的特征
features = np.unique(high_corr[['feature1', 'feature2']].values)
sub_corrmat = corrmat.loc[features, features]

# 绘制热力图
plt.figure(figsize=(12, 9))
sns.heatmap(sub_corrmat, annot=True, cmap='coolwarm', vmax=1.0, vmin=-1.0,
```

square=True, cbar_kws={"shrink": .82})
font = FontProperties(fname=r"C:\Windows\Fonts\SimHei.ttf", size=14) # 根据实际情况修改字体路径
plt.title('相关性大于 0.8 的特征热力图', fontproperties=font)
plt.show()

%% [markdown]
进一步挖掘特征

%% [markdown]
1. 转换部分数值特征为分类特征
#
我们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，因此我们将其转化为分类特征。
- **MSSubClass** 住宅标识
- **YrSold** 开售年份
- **MoSold** 开售月份

%%
all_data['MSSubClass'] = all_data['MSSubClass'].apply(str) # apply()函数默认对列进行操作
all_data['YrSold'] = all_data['YrSold'].astype(str)
all_data['MoSold'] = all_data['MoSold'].astype(str)

%% [markdown]
2. 转换部分分类特征为数值特征
#
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。\
我们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。
（标签编码 LabelEncoder）
- **PoolQC** 泳池质量，有 Ex Excellent（优秀）Gd Good（良好） 等等

# %%
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')

```
# 处理列，将标签编码应用于分类特征
for c in cols:
        lbl = LabelEncoder()
        lbl.fit(list(all_data[c].values))
        all_data[c] = lbl.transform(list(all_data[c].values))

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

# %% [markdown]
# 3. 利用一些重要的特征构造更多的特征
#
# - TotalBsmtSF：地下室总面积
# - 1stFlrSF：一层面积
# - 2ndFlrSF：二层面积
# - OverallQual：整体材料和装饰综合质量
# - GrLivArea：地上生活面积
# - TotRmsAbvGrd：地上总房间数
# - GarageArea：车库面积
# - YearBuilt：建造时间

# %%
# 构造更多的特征
all_data['TotalSF'] = all_data['TotalBsmtSF'] + all_data['1stFlrSF'] + all_data['2ndFlrSF'] # 房屋总面积

all_data['OverallQual_TotalSF'] = all_data['OverallQual'] * all_data['TotalSF']  # 整体质量与房屋总面积交互项
all_data['OverallQual_GrLivArea'] = all_data['OverallQual'] * all_data['GrLivArea'] # 整体质量与地上总房间数交互项
all_data['OverallQual_TotRmsAbvGrd'] = all_data['OverallQual'] * all_data['TotRmsAbvGrd'] # 整体质量与地上生活面积交互项
all_data['GarageArea_YearBuilt'] = all_data['GarageArea'] * all_data['YearBuilt'] # 车库面积与建造时间交互项

# %% [markdown]
# #### 对特征进行 Box-Cox 变换

# %% [markdown]
# 1. 对于数值型特征，我们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。那么我们先来计算一下各个特征的偏度：

# %%
```

```
# 筛选出所有数值型特征(63 个)
numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index

# 计算特征的偏度
numeric_data = all_data[numeric_feats]
skewed_feats = numeric_data.apply(lambda x: skew(x.dropna())).sort_values(ascending=False)
skewness = pd.DataFrame({'Skew' :skewed_feats})
skewness.head(10)

# 筛选偏度大于 0.75 的变量
high_skew_feats = skewness[skewness['Skew'] > 0.75]

# 绘制偏度大于 0.75 的变量的柱状图
plt.figure(figsize=(15, 10))
ax = sns.barplot(y=high_skew_feats.index, x=high_skew_feats['Skew'])
plt.yticks(rotation=0, fontproperties=font)
plt.xlabel('偏度', fontsize=15, fontproperties=font)
plt.ylabel('数值特征', fontsize=15, fontproperties=font)
plt.title('偏度大于 0.75 的数值特征的偏度', fontsize=15, fontproperties=font)

# 在每个柱子上显示数值
for container in ax.containers:
    ax.bar_label(container, fmt='%.2f')



# %% [markdown]
# 2. 对高偏度的特征进行 Box-Cox 变换

# %%
new_skewness = skewness[skewness.abs() > 0.75]
print("有{}个高偏度特征被 Box-Cox 变换".format(new_skewness.shape[0]))

# %% [markdown]
# 使用 scipy 中 boxcox1p() 函数计算**\\(1 + x\\) 的 Box-Cox 变换** ，当 \\( \lambda = 0 \\) 等效于目标变量的 log1p
#
# 选出偏度绝对值大于 0.75 的数据；找出这些偏度对应的特征；将偏度平滑至 0.15
#

# %%
```

```
from scipy.special import boxcox1p

skewed_features = new_skewness.index
lam = 0.15
for feat in skewed_features:
    all_data[feat] = boxcox1p(all_data[feat], lam)

# %% [markdown]
# #### 独热编码（one-hot encoding)

# %% [markdown]
# 上述主要是在处理数值型特征，接下来处理类别特征。
# 对于类别特征，我们将其转化为独热编码，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。

# %%
df = pd.DataFrame({'color': ['红', '蓝', '黄']})
df

# %%
lbl = LabelEncoder()
df['color'] = lbl.fit(list(df.values)).transform(list(df.values))
df

# %%
df = pd.DataFrame({'color': ['红', '蓝', '黄']})
pd.get_dummies(df)

# %%
all_data = pd.get_dummies(all_data)
print(all_data.shape)

# %% [markdown]
# ### 建立模型
#

# %% [markdown]
# #### 导入算法包

# %%
from sklearn.linear_model import ElasticNet, Lasso
from sklearn.ensemble import 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

# %%
y_train = train.SalePrice.values
train = all_data[:train.shape[0]] #切分训练集和测试集
test = all_data[train.shape[0]:]

# %%
train_numerical = train.select_dtypes(include=[np.number])
train_numerical.shape

# %% [markdown]
# #### 评价函数

# %% [markdown]
# 先定义一个评价函数。采用 5 折交叉验证。根据比赛评价标准，用 Root-Mean-Squared-Error (RMSE)来为每个模型打分。
#
# 使用 sklearn 中的 cross_val_score()函数，但是这个函数没有 shuffle 属性，所以添加一行代码，在交叉验证之前对数据集进行 shuffle。
#
# shuffle=True 就表示对数据打乱，重新洗牌，这样就能设置随机种子。

# %%
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_numerical.values, y_train, scoring="neg_mean_squared_error", cv=kf))
    return(rmse)

# %% [markdown]
# #### 基本模型

# %% [markdown]
# -  **LASSO Regression（套索回归）**
#
```

%% [markdown]
lasso 回归就是在传统的线性回归目标函数后面加上了一个 L1-范数，也称为为 L1 正则项，这样就能使得不重要特征的系数压缩至 0，从而实现较为准确的参数估计效果，但是该模型可能对异常值非常敏感，由于数据集中依然存在一定的离群点，所以我们 sklearn 中的 RobustScaler 对数据进行标准化处理。

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

%% [markdown]
- **Kernel Ridge Regression（核岭回归）**
#
还有一种压缩估计的方法叫岭回归,它通过在目标函数后加上一个 L2 正则项，同样达到防止过拟合的作用。 \
核岭回归是先对数据作核变换以适应各种非线性情形,然后再对数据作岭回归的一种模型。

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

%% [markdown]
- **ElasticNet Regression（弹性网络）**
#
当多个特征存在相关时，Lasso 回归可能只会随机选择其中一个，岭回归则会选择所有的特征。如果将这两种正则化的方法结合起来，就能够集合两种方法的优势，这种正则化后的算法就被称为弹性网络回归。

# %%
ENet = make_pipeline(RobustScaler(), ElasticNet(alpha=0.0005, l1_ratio=.9, random_state=3))

%% [markdown]
- **Gradient Boosting Regression（梯度提升回归 GBRT）**
#

%% [markdown]
梯度提升回归树(GBRT)算法是以 CART 回归树为基学习器的集成学习模型，通过迭代多棵回归树生成多个弱模型，然后将每个弱模型的预测结果相加，通过不断减小损失函数的值来调整模型。与其他集成学习算法相比，梯度提升回归树对参数设置更为敏感。

%%

```
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) #  设置 hue
loss 使其对异常值具有鲁棒性

# %% [markdown]
# - **XGBoost（极致梯度回归）**

# %% [markdown]
#  极致梯度提升（XGBoost）是基于 GBRT 的一种改进算法，通过引入模型复杂
度重新构造目标函数，在迭代每一棵树的过程中都力求最小化，同时最小化了模
型的损失函数即损失率和模型的复杂度，从而来提高算法的运算效率。

# %%
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=2200,
                             reg_alpha=0.5, reg_lambda=0.8,
                             subsample=0.5, random_state=7, nthread=-1)

# %% [markdown]
# - **LightGBM**

# %% [markdown]
# LightGBM 也是基于 GBRT 算法的一种优化改进， \
#  其改进之一是采用了基于直方图的决策层树算法，对每个特征的取值做分段函
数，将所有样本在该特征上的取值划分到某一段中，最终把特征取值离散化；\
#  改进之二是采用 Leaf-wise 的生长策略，使增益高的节点优先生长，从而减少
低增益点浪费计算资源的问题，提高计算效率。

# %%
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=0.2,
                              feature_fraction_seed=9, bagging_seed=9,
                              min_data_in_leaf=6,
min_sum_hessian_in_leaf=11, verbose=-1)

# %% [markdown]
# **模型效果评价**
```

```
# %% [markdown]
# 让我们通过评估交叉验证 RMSE 的均值和标准差来看看这些基础模型的表现

# %% [markdown]
# 五种基础模型

# %%
models = {'LinearRegression':linear,'Lasso': lasso, 'ElasticNet': ENet, 'Kernel Ridge': KRR,'SVM':svm_sigmoid}
results=[]
for model_name, model in models.items():
    score = rmsle_cv(model)
    results.append([model_name, round(score.mean(),4), score.std()])

# 创建 DataFrame
df_results = pd.DataFrame(results, columns=['Model', 'Mean Score', 'Std Dev'])

df_sorted = df_results.sort_values(by='Mean Score', ascending=True)
pd.set_option('display.float_format', '{:.4f}'.format)
# 显示排序后的结果
print("Sorted Results:")
print(df_sorted)

# %% [markdown]
# svm 参数比较

# %%
models = {'SVM_rbf':svm_rbf, 'SVM_sigmoid':svm_sigmoid,'svm_linear':svm_linear}
results=[]
for model_name, model in models.items():
    score = rmsle_cv(model)
    results.append([model_name, round(score.mean(),4), score.std()])

# 创建 DataFrame
df_results = pd.DataFrame(results, columns=['Model', 'Mean Score', 'Std Dev'])

df_sorted = df_results.sort_values(by='Mean Score', ascending=True)
pd.set_option('display.float_format', '{:.4f}'.format)
# 显示排序后的结果
print("Sorted Results:")
print(df_sorted)
```

```
# %%


# %%
models = {'LinearRegression':linear,'Lasso': lasso, 'ElasticNet': ENet, 'Kernel Ridge': KRR,  'SVM':svm_sigmoid,
              'Gradient Boosting': GBoost, 'XGBoost': model_xgb, 'LightGBM': model_lgb,'RandomForest':rf}
results=[]
for model_name, model in models.items():
      score = rmsle_cv(model)
      results.append([model_name, round(score.mean(),4), score.std()])

# 创建 DataFrame
df_results = pd.DataFrame(results, columns=['Model', 'Mean Score', 'Std Dev'])

df_sorted = df_results.sort_values(by='Mean Score', ascending=True)
pd.set_option('display.float_format', '{:.4f}'.format)
# 显示排序后的结果
print("Sorted Results:")
print(df_sorted)

# %% [markdown]
# ### Bagging

# %%
from sklearn import linear_model
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import BaggingRegressor
linear=linear_model.LinearRegression()

from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor(n_neighbors=5)

rf=RandomForestRegressor(n_estimators=l00, #决策树的数量
                                 max_depth=None,#树的最大深度
                                 min_samples_split=2,#拆分内部节点所需的最小样本数
                                 random_state=42)#随机种子以确保重现性
bagging_lasso = BaggingRegressor(base_estimator=lasso,
                                          n_estimators=25,#使用 25 个 1asso 模型
                                          max_samples=0.5,#每个模型使用 50%的数
```

```
据,
                                        max_features=0.8,#每个模型使用 8g%的特
征,
                                        random_state=42)
#随机种子以确保重现性
bagging_xgb = BaggingRegressor(base_estimator=model_xgb,
                               n_estimators=25,#使用 25 个 KGBoost 模型
                               max_samples=0.5,#每个模型使用 50%的数据
                               max_features=0.8,#每个模型使用 80%的特征
                               random_state=42)#随机种子以确保重现性

# %%
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)

# 训练模型
rf_regressor.fit(train_numerical.values, y_train)

feature_importances = rf_regressor.feature_importances_

# 创建特征和其重要性的 DataFrame
importances_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': feature_importances
})

# 对特征重要性进行排序
importances_df = importances_df.sort_values(by='Importance', ascending=False)

# 显示特征重要性
print(importances_df)

# %%
results = []

for model_name, model in models.items():
    score = rmsle_cv(model)
    results.append([model_name, round(score.mean(),4), score.std()])

# 创建 DataFrame
df_results = pd.DataFrame(results, columns=['Model', 'Mean Score', 'Std Dev'])

df_sorted = df_results.sort_values(by='Mean Score', ascending=True)
```

```
# 显示排序后的结果
print("Sorted Results:")
print(df_sorted)

# %% [markdown]
# #### 堆叠方法（[Stacking Models][1]）
#
# [1]: https://www.jianshu.com/p/59313f43916f
#

# %% [markdown]
# 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。

# %% [markdown]
# ![image-2.png](attachment:image-2.png)

# %% [markdown]
# 定义一个新类，表示 Stacking 方法

# %%
class StackingAveragedModels(BaseEstimator, RegressorMixin, TransformerMixin):
    def __init__(self, base_models, meta_model, n_folds=5):
        self.base_models = base_models    # 第一层模型
        self.meta_model = meta_model      # 第二层模型
        self.n_folds = n_folds

    # 运用克隆的基本模型拟合数据
    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)

        # 训练克隆的第一层模型
        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
```

```
            # 使用交叉验证预测的结果作为新特征，来训练克隆的第二层模型
            self.meta_model_.fit(out_of_fold_predictions, y)
            return self

    # 在测试数据上做所有基础模型的预测，并使用平均预测作为由元模型完成的最终预测的元特征
    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)

# %% [markdown]
# 这里我们用 ENet、KRR 和 GBoost 作为第一层学习器，用 Lasso 作为第二层学习器。查看以下 Stacking 的交叉验证评分：

# %%
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()))

# %% [markdown]
#

# %% [markdown]
# 我们得到了比单个基学习器更好的分数。

# %% [markdown]
# #### 建立最终模型

# %% [markdown]
# **集成 StackedRegressor, XGBoost 和 LightGBM**

# %% [markdown]
# **我们将 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**

# %% [markdown]
# 1. 首先，定义一个评价函数,表示预测值和实际值之间的均方根误差 RMSE
```

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

# %% [markdown]
#2. 其次，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

# %% [markdown]
# 注：numpy 中的 expm1()函数表示 \\(e^{x}-1\\)，即将预测的对数值转换为原始预测值

# %% [markdown]
# - **StackedRegressor:**

# %%
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))

# %% [markdown]
# - **XGBoost:**

# %%
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))

# %% [markdown]
# - **LightGBM:**

# %%
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))

# %%
print('集成模型的得分:{}'.format(rmsle(y_train, stacked_train_pred*0.70 +
xgb_train_pred*0.15 + lgb_train_pred*0.15)))
```

```
# %% [markdown]
# 3. 生成最终预测结果

# %%
ensemble = stacked_pred*0.70 + xgb_pred*0.15 + lgb_pred*0.15

# %% [markdown]
# ####  提交结果

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