房价预测代码审查报告

摘要

我们对第 5 组的模型代码进行了审查，代码可以复现，但也发现了一些问题，主要包括：数据泄露、特征工程处理不当、模型选择不当。这些问题导致模型性能不好，并且难以在实际应用中有较好效果。

1. 数据加载：没有充分使用所有数据集

代码加载了四个数据集，但实际上只使用了训练集和测试集，df_housing 和 df_rent 两个数据集被完全忽略，小区信息和租房信息中有用的数据没有充分使用。

```
df_housing = pd.read_csv(url+"/ruc_Class25Q1_details.csv")
df_rent = pd.read_csv(url+"/ruc_Class25Q1_rent.csv")
df_train = pd.read_csv(url+"/ruc_Class25Q1_train.csv")
df_test = pd.read_csv(url+"/ruc_Class25Q1_test.csv")
```

2. 数据处理与特征工程

2.1 数据泄露

代码中存在数据泄露问题，在填充缺失值的时候，计算了整个数据集上的统计量，应用在训练集和测试集：

```
# 在全部训练集上计算统计量
huanxian = []
for i in range(7):
    huanxian.append(df_train.loc[df_train["城市信息"].astype(int) == i, "环线
ti_25 = df_train["梯"].quantile(0.25)
hu_75 = df_train["户"].quantile(0.75)
# ... 更多统计量

# 然后在测试集上应用这些统计量
df_train = zym_fillna(df_train)
df_test = zym_fillna(df_test)

...
# 最终训练结果展示
print(results_df)
          Model   In-Sample MAE   Out-Sample MAE   6-Fold CV MAE
0           OLS   515460.634136     8.759804e+06    9.247794e+05
1         LASSO   975849.750884     1.008257e+06    9.761699e+05
2         Ridge   570160.538975     1.563713e+07    1.318913e+06
3    ElasticNet   848424.756653     3.253918e+07    9.066031e+05
```

这种做法导致测试集与训练集都含有彼此的信息，可能导致模型在验证集表现较好、在测试集表现不好，模型泛化能力较差。就最终 MAE 结果来看，验证集 MAE 确实远大于训练集，表现很差。

2.2 异常值处理缺失

代码中没有针对异常值的处理。唯一过滤的数据仅用于可视化，没有用在最终模型里：

```
df_filtered = df_train[(df_train["建筑面积"] > 100) & (df_train["建筑面积"] < 5
```

但房价数据通常包含异常值，比如说下列的建筑面积的最大值和最小值都有异常（由我们检查发现）。忽略异常值处理可能导致模型受极端值影响较大。

```
X_train['建筑面积'].describe()
```

```
count      67306.000000
mean          55.887945
std           47.337856
min            1.000000
25%           21.000000
50%           54.000000
75%           83.000000
max         5108.000000
Name: 建筑面积, dtype: float64
```

2.3 分类变量处理不当

代码将分类变量转换为数值时，直接赋予了数值，但这些赋值缺乏明确的依据：

```
mappings = {
    '产权所属': {
        '共有': 1, '非共有': 0
    },
    '房屋年限': {
        '满五年': 0, '满两年': 2, '未满两年': 7
    },
    '房屋用途': {
        '车库': 0, '商业': 1, '商业办公类': 1, '写字楼': 1,
        '底商': 2, '商住两用': 2,
        '老公寓': 3, '平房': 3,
        # ... 更多映射
    }
      '环线': {
        '内环内': 1, '一环内': 1, '二环内': 2, '外环外': 5,
        '四环外': 5, '三环外': 4, '六环外': 7,
        '一至二环': 1.5, '二至三环': 2.5, '三至四环': 3.5,
        '四至五环': 4.5, '五至六环': 5.5,
        '内环至外环': 3, '内环至中环': 2, '中环至外环': 4
    }
    # ... 更多类别变量映射
}
```

这种处理方式隐含地假设了类别之间存在顺序关系，例如将"车库"赋值为 0，"商业"赋值为 1，这暗示"商业"与"车库"的价值有线性提升关系，但实际上它们可能是完全不同的类别，应该使用独热编码或其他更合适的编码方法。

此外，环线在不同城市中有不同的定义体系，直接统一用一套数值映射来处理，可能会引入逻辑错误和噪声。比如说，城市 a 的“四环至五环”代表相对偏远的位置，而城市 b 的“内环至外环”可能对应实际类似的地理位置，但映射方式是把所有城市的“环线”都用统一数值映射，这样看起来是“有序”，但它在不同城市中实际代表的距离含义不同，模型会误以为这两个地方相近，从而学不到真正的空间规律。

2.4 多重共线性问题

代码创建了大量的交互项和高阶项，但没有进行适当的共线性检查：

```
df['城市 1 交叉'] = df['城市_1'] * df['环线']
df['城市 1 交叉 1'] = df['城市_1'] * df['lon2']
df['城市 1 交叉 2'] = df['城市_1'] * df['lat2']
# ... 大量类似的交互项
```

同时，还使用了线性相关的变量，例如：

- "上次年份"、"交易年份"、"交易时长"（这三者之间存在直接的数学关系）
- "建筑面积交叉"和"建筑面积"（高度相关）

多重共线性会导致线性模型不稳定，系数难以解释，并可能引发过拟合。

2.5 缺失值填充缺乏依据

代码中的缺失值填充策略缺乏依据或数据支持，比如套内面积的填充：

```
df.loc[df["套内面积"] == 0, "套内面积"] = df.apply(
    lambda row: 0.8 * row["建筑面积"] + 0.38 if row["建筑面积"] > 100 else 8
    axis=1
)
```

没有解释计算公式中系数（0.8 和 0.38）的来源；同样，针对建筑面积小于 100 的情况使用固定值 80.65 也缺乏明确的依据。

2.6 文本数据处理不充分

文本处理较简单，仅进行了基本的关键词匹配：

```
df["周边配套"] = df["周边配套"].fillna("")
df["周边配套"] = df["周边配套"].apply(lambda x: re.sub(r"[，、。 ； ：]", ",", x))
df["周边配套"] = df["周边配套"].apply(lambda x: re.sub(r"\s+", "", x))  # 去除
categories = ["医院", "公园", "超市", "商场", "银行", "学校", "地铁", "公交"]
for cat in categories:
    df[cat] = df["周边配套"].apply(lambda x: 1 if cat in x else 0)
```

这种处理方式存在的主要问题为：

1. 关键词选择没有明确依据，比如没有统计高频词、利用 NLP 技术等等，纯手动创建字典，存在优化空间；
2. 忽略了其他文本信息，"房屋优势"、"核心卖点"、"户型介绍"等文本字段的内容分析；

2.7 不合理的特征创建

代码创建了一些不合理的特征：

```
df['建筑面积正常'] = (df['建筑面积'] >= 100) & (df['建筑面积'] <= 600)
df['建筑面积交叉'] = df['建筑面积正常'] * df['建筑面积']
```

这个"建筑面积交叉"特征存在几个问题：

1. 与原始的"建筑面积"高度相关，引入共线性
2. 创建逻辑不清晰，难以解释其业务含义，本质上只是对建筑面积进行了分段处理，将建筑面积过大或过小的样本暴力地赋值为 0，其它则保持不变，缺乏连续性。它既不是“是否超大面积”，也不是“标准面积”，而是一个**人为截断的原始值**，可能反而引入了噪声。

3. 模型训练与评估问题

3.1 随机种子设置不符合要求

代码中使用了随机种子 42，而非指定的 111：

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42  # 应为 111
)
```

3.2 缺乏超参数调优

所有模型都使用了默认参数或手动设置的固定参数，没有进行任何超参数搜索，可能导致模型性能有更大优化空间：

```
models = {
    'OLS': LinearRegression(),
    'LASSO': Lasso(alpha=0.1, tol=1e-4, max_iter=100000),
    'Ridge': Ridge(alpha=1),
    'ElasticNet': ElasticNet(alpha=0.1, l1_ratio=0.5, max_iter=100000)
}
```

3.3 模型选择逻辑错误

代码的模型选择逻辑有问题，比如一开始选择最佳的模型是依据六重交叉验证的 MAE：

```
best_model_name = results_df.loc[results_df['6-Fold CV MAE'].idxmin(), 'Mo
best_model = models[best_model_name]
```

但在最终生成预测时选择了 OLS 模型（最终分数较高）：

```
test_predictions = models["OLS"].predict(X_real_scaled)
```

后续代码又选择了弹性网络：

```
test_predictions = best_model.predict(X_real_scaled)
```

这种不一致可能导致实际使用的并非最佳模型。另外，结果表明 ElasticNet 模型的交叉验证 MAE 最低，但这可能是由于过拟合，而非真正的模型性能较好。

3.4 目标变量处理不当

就代码块顺序而言，是先对价格进行对数变换，然后才绘制直方图，不符合正常的思维顺序：

```
df_train["price1"] = np.log(df_train["价格"])
# ...后续才进行可视化
df_train['价格'].hist(bins=30)
```

应该先可视化原始数据，确认分布特性，再决定是否需要变换。

4. 代码结构与效率问题

4.1 代码冗余

代码中存在多处可以统一打包为函数的部分，比如说周边配套和交通出行的处理：

```
# 周边配套处理
df["周边配套"] = df["周边配套"].fillna("")
df["周边配套"] = df["周边配套"].apply(lambda x: re.sub(r"[，、。 ； ：]", ",", x))
df["周边配套"] = df["周边配套"].apply(lambda x: re.sub(r"\s+", "", x))
categories = ["医院", "公园", "超市", "商场", "银行", "学校", "地铁", "公交"]
for cat in categories:
    df[cat] = df["周边配套"].apply(lambda x: 1 if cat in x else 0)

# 交通出行处理（几乎相同的代码）
df["交通出行"] = df["交通出行"].fillna("")
df["交通出行"] = df["交通出行"].apply(lambda x: re.sub(r"[，、。 ； ：]", ",", x))
df["交通出行"] = df["交通出行"].apply(lambda x: re.sub(r"\s+", "", x))
categories = ["地铁", "公交", "高速", "高铁", "机场"]
for cat in categories:
    df[cat] = df["交通出行"].apply(lambda x: 1 if cat in x else 0)
```

重复代码应打包为函数，提高可维护性、可读性和代码简洁。

4.2 包导入不必要或使用不当

代码导入了一些未使用的包：

```
from pathlib import Path
import tarfile
import urllib.request
```

同时，中文数字转换可以使用专门的包（如 cn2an），而不是自己实现复杂的函数：

```
def chinese_to_number(chinese_num):
    if pd.isna(chinese_num) == True: return 0
    digit_map = {
        '零':0, '一':1, '二':2, '三':3, '四':4,
        '五':5, '六':6, '七':7, '八':8, '九':9,
        '十':10, '百':100, '两':2
    }
    # ...复杂的转换逻辑
```

4.3 计算效率问题

Lasso 模型设置了非常高的最大迭代次数，但没有进行特征筛选，这可能导致计算非常缓慢：

```
'LASSO': Lasso(alpha=0.1, tol=1e-4, max_iter=100000)
```

此外，模型包含大量可能共线的特征，这种设置会导致算法迭代次数增加，耗时显著增加。

4.4 缺乏代码注释与文档

整个代码缺乏适当的注释，特别是在关键决策点和复杂算法处。例如，下面的特征工程代码没有解释为什么要添加这些特征或它们的含义：

```
df['城市 1 交叉'] = df['城市_1'] * df['环线']
df['城市 2 交叉'] = df['城市_2'] * df['环线']
# ... 更多交叉特征
df['lat2'] = df['lat'] ** 2
df['lon2'] = df['lon'] ** 2
df['time2'] = df['交易年份'] - 2000) ** 2
```

缺乏注释使代码难以理解和维护，也使其他研究者难以重现或改进模型。

5. 结论与建议

本次代码审查发现了多项问题，包括数据泄露、特征工程处理不当、模型选择不当等。这些问题会影响模型的实际性能和适用性。

主要建议：

1. **避免数据泄露**：先进行训练测试集划分，然后仅基于训练集计算用于缺失值填充的数据，避免使用整个数据集的统计信息。
2. **改进特征工程**：
    a. 避免对非有序变量（如房屋用途、房屋类型、环线）使用线性赋值（0/1/2...），这会导致模型产生“强弱关系”的错误假设。
        i. 例如对于环线变量，可以先判断城市，然后使用城市特定的环线映射字典；或者直接对不同城市不同环线进行 one-hot 编码，避免模型默认其有大小关系，引入错误假设。
    b. 系统性地处理异常值和缺失值 :对价格、面积等数值字段应检查其描述性统计情况，分析其分布，使用 IQR 或 z-score 方法剔除或 Winsorize 极端值。
    c. 减少特征间的共线性：进行 VIF（方差膨胀因子）分析筛除冗余变量；考虑用 PCA 降维或仅保留信息量最大的几个交易时间维度 ;相似问题也出现在“建筑面积”、“建筑面积交叉”中，建议择优保留。
    d. 如果想对建筑面积进行分段建模，可以先对建筑面积分箱再进行独热编码。
3. **充分利用所有数据**：分析并整合所有可用数据集
4. **改进模型训练流程**：
    a. 进行超参数调优 :使用 GridSearchCV 或 RandomizedSearchCV 进行交叉验证调参；采用多评价指标（如 MAE、RMSE）平衡评估
    b. 使用特征选择减少特征数量 : 可以采用 L1 正则 (如 Lasso) , 或逐步回归等方法先进行特征筛选
    c. 保持模型选择和应用的一致性
5. **优化代码结构**：
    a. 添加详细注释
    b. 抽象重复逻辑为函数：
        i. 例如：城市 1 交叉 ~ 城市 6 交叉 2 是重复冗长代码，建议用 for 循环批量生成。
        ii. 周边配套 与 交通出行 提取关键词处理也可封装为 parse_keywords(df, col, keyword_list) 类型的函数。
    c. 使用适当的专业库：

i. 自定义的 chinese_to_number 可以被 cn2an 等库替代，提升鲁棒性与精度；
ii. 数据预处理可用 sklearn.pipeline 或 Feature-engine 工具包提高标准化程度和代码整洁性。

通过解决这些问题，可以显著提高模型的预测性能和代码的可维护性。