Final exam

张雷臻

数据处理

- **异常值处理:** 四分位距（IQR）方法
- **类别变量：** 标签编码、独热编码、类别编码
- **正则表达式：** 清洗标点符号、数字、提取文本（建筑面积、梯户、户型）
- **文本分析：** 提取高频词，自定义词库，构建新的特征变量
- **聚类方法：** 使用K Means聚类创建环线特征、使用KNN创建区域价值特征

数据处理

- **正则表达式：** 清洗标点符号、数字、提取文本（建筑面积、梯户、户型）

```
ori_pre_data['建筑面积'] = (
    ori_pre_data['建筑面积']
    .astype(str)  # 确保是字符串类型
    .str.replace(r'\s*m²\s*', '', regex=True)  # 删除“m²”及其前后空格
    .str.replace(r'[^0-9.]+', '', regex=True)  # 删除非数字字符（如“平方米”）
)
```

```
# 处理阿拉伯数字格式（如"2梯3户"或"2T3户"）
arabic_pattern = r'(\d+)(?:梯|T)(\d+)户'
arabic_df = df['梯户比例'].str.extract(arabic_pattern)
arabic_df.columns = ['梯数', '户数']
arabic_df = arabic_df.astype(float)
```

```
words = [
    # 分词并过滤标点符号
    word for word in jieba.lcut(text_data)
    # 使用正则表达式过滤非文本字符（保留中文、英文、数字）
    if len(word) > 1 and re.search(r'[\u4e00-\u9fa5a-zA-Z0-9]{2,}', word)
]
```

数据处理

- **聚类方法：**使用K Means聚类创建环线特征、使用KNN创建区域价值特征

```python
def create_ring_features(df, n_clusters=5):
    """为每个城市创建环线特征"""
    df['环线'] = 0  # 初始化环线特征
    for city in df['城市'].unique():
        city_df = df[df['城市'] == city]
        # 使用KMeans聚类
        kmeans = KMeans(n_clusters=n_clusters, random_state=42)
        clusters = kmeans.fit_predict(city_df[['lon', 'lat']])
        # 获取聚类中心并排序（从市中心到郊区）
        centers = kmeans.cluster_centers_
        main_center = centers.mean(axis=0)  # 主中心点
        # 计算每个中心点到主中心的距离
        distances = np.linalg.norm(centers - main_center, axis=1)
        sorted_indices = np.argsort(distances)
        # 创建环线映射（距离市中心越近，环线值越小）
        ring_mapping = {sorted_indices[i]: i+1 for i in range(n_clusters)}
        # 应用环线分类
        city_rings = [ring_mapping[cluster] for cluster in clusters]
```

```python
# 3. 使用KNN创建区域价值特征
def create_area_value_feature(df, n_neighbors=50):
    """创建基于邻近房屋价值的区域特征"""
    # 只使用训练数据计算（避免数据泄露）
    train_data = df[df['价格'].notnull()]

    # 初始化模型
    knn = NearestNeighbors(n_neighbors=n_neighbors)
    knn.fit(train_data[['lon', 'lat']])

    # 为所有房屋查找邻近房屋
    distances, indices = knn.kneighbors(df[['lon', 'lat']])

    # 计算邻近房屋的平均价格（仅对训练数据）
    neighbor_prices = train_data.iloc[indices.flatten()]['价格'].values.reshape(indices.shape)
    avg_neighbor_price = np.nanmean(neighbor_prices, axis=1)

    df['区域价值指数'] = avg_neighbor_price
    df['区域价值指数'].fillna(df['区域价值指数'].mean(), inplace=True)
```

模型选择

- **神经网络：**添加梯度裁剪、调整学习率调度器、添加完整监控

```python
# 调整模型结构
model1 = Sequential([
    Dense(512, kernel_initializer='he_normal', kernel_regularizer=regularizers.l2(0.001), input_dim=X_train.shape[1]),
    BatchNormalization(),
    LeakyReLU(alpha=0.1),
    Dropout(0.4),

    Dense(256, kernel_initializer='he_normal', kernel_regularizer=regularizers.l2(0.001)),
    BatchNormalization(),
    LeakyReLU(alpha=0.1),
    Dropout(0.3),

    Dense(128, kernel_initializer='he_normal', kernel_regularizer=regularizers.l2(0.001)),
    BatchNormalization(),
    LeakyReLU(alpha=0.1),
    Dropout(0.2),

    Dense(1, activation='linear')
])
```

```python
# 模型编译
model1.compile(
    optimizer=Adam(learning_rate=0.001, clipvalue=0.5),  # 添加梯度裁剪
    loss=Huber(delta=1.5),  # 调整delta参数
    metrics=['mae',
             RootMeanSquaredError(name='rmse')]
)
# 调整学习率调度器
callbacks = [
    EarlyStopping(monitor='val_rmse', patience=15, mode='min'),
    ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=1e-6),
    ModelCheckpoint('best_model.h5', monitor='val_loss', save_best_only=True)
]

# 启动训练（添加完整监控）
history = model1.fit(
    X_train_scaled, y_train,
    epochs=200,
    batch_size=64,
    validation_split=0.25,
    callbacks=callbacks,
    verbose=2
)
```

模型选择

- XG Boost：扩大最大迭代次数、延长早停观察窗口、更频繁的进度输出

```
model2 = xgb.train(params, dtrain, num_boost_round=5000,
                   evals=[(dtrain, 'train'), (dvalid, 'valid')],
                   early_stopping_rounds=50, verbose_eval=100)

print(f"Best iteration: {model2.best_iteration}, Best MAE: {model2.best_score:.4f}")
model3 = xgb.train(
    params,
    dtrain,
    num_boost_round=5000,          # 扩大最大迭代次数
    evals=[(dvalid, 'valid')],    # 专注验证集表现
    early_stopping_rounds=100,    # 延长早停观察窗口
    verbose_eval=50               # 更频繁的进度输出
)
```

```
print(performance_table_sj)
print(performance_table_xg)
```

	指标	值
0	MAE	1.609084e+05
1	RMSE	6.654770e+10
2	R²	9.204757e-01
3	MAPE	1.355798e+01

	指标	值
0	MAE	1.157187e+05
1	RMSE	3.214545e+10
2	R²	9.615863e-01
3	MAPE	1.071735e+01

submission_1.csv	2025/06/16 04:05	张雷臻	已评审	62.085	62.085