# 中山大学计算机学院

# 人工智能

# 本科生实验报告

课程名称：Artificial Intelligence

| 学号 | 23336266 | 姓名 | 熊彦钧 |
|---|---|---|---|

## 一、实验题目: 感知机算法

房价预测任务 data.csv 数据集包含六个属性，共 10000 条数据。其中 longitude 和 latitude 表示房子经纬度， housing_age 表示房子年龄， homeowner_income 表示房主的收入（单位：十万元）。请根据用戶的年龄以及估计的薪水，利用感知机算法预测房价，并画出数据可视化图、loss 曲线图。

提示最后提交的代码只需包含性能最好的实现方法和参数设置，只需提交一个代码文件 ，请不要提交其他文件。

2. 本次作业可以使用 numpy 库、 matplotlib 库以及 python 标准库.

3. 数据集可以在 Github 上下载。

## 二、实验内容

### 1.1: 数据收集与理解:

本实验首先对包含 20,000 条记录的房价数据集进行了详细分析，该数据集包含经度、纬度、房龄、房主收入四个特征以及房价这一目标变量。在数据预处理阶段，我们实现了 IQR 和 Z-score 两种异常值检测方法，通过可视化手段清晰展示了异常值分布及处理效果，并对处理前后的数据分布变化进行了系统对比。针对地理特征的特殊性，我们创新性地使用 K-means 聚类算法将经纬度转换为到五个聚类中心的距离特征，这一转换不仅保留了地理位置信息，还增强了模型对空间关系的理解能力。

### 1.2: 数据预处理流程:

在特征工程环节，我们对所有数值特征进行了 Z-score 标准化处理，确保不同尺度的特征能够被公平对待，同时保存了标准化参数以便后续预测结果的还原。模型架构设计方面，我们构建了一个具有两层隐藏层的 MLP 网络，输入层包含 8 个节点（原始特征转换后），第一隐藏层采用 20 个 tanh 激活节点捕捉初始非线性特征，第二隐藏层使用 10 个 ReLU 激活节点缓解梯度消失问题，输出层则采用线性激活函数适配回归

任务需求。

1.3 模型构建与训练：

为了优化模型性能，我们设计了系统的参数实验，测试了 0.05、0.02、0.01 三种学习率与 1000、2000、5000 三种迭代次数的组合，共形成 9 种不同的训练配置。在模型评估阶段，我们不仅计算了标准化 MSE 和原始尺度 MSE，还引入了 $R^2$决定系数来全面评估模型性能。通过精心设计的可视化方案，包括损失曲线对比、预测值与真实值散点图以及各特征与房价关系图等多角度展示，使模型表现和特征影响得以直观呈现，为后续模型优化提供了清晰的方向指引。

# 三、关键代码展示

（1）数据读取与处理：

load_data 函数：使用 Python 的 csv 模块读取文件，跳过表 头，将每行数据转换为浮点数，最后返回 NumPy 数组。

detect_and_remove_outliers 函数：采用 IQR 方法和 Z-score 方法清洗异常值，返回清洗后的特征数据。

normalize_data 函数：对每个特征计算均值和标准差，然后使用公式(X-mean)/std 进行标准化。

```python
# 读取数据函数
def load_data(filename):
    data = []
    with open(filename, 'r', encoding='utf-8') as f:
        reader = csv.reader(f)
        next(reader)  # 跳过表头
        for row in reader:
            data.append([float(x) for x in row])  # 转换为浮点数
    return np.array(data)

# 异常值检测与处理
def detect_and_remove_outliers(X, y, method='iqr', threshold=1.5):
    data = np.hstack((X, y.reshape(-1, 1)))
    n_samples, n_features = data.shape
    outliers_mask = np.zeros(n_samples, dtype=bool)

    if method == 'iqr':
        for i in range(n_features):
            q1 = np.percentile(data[:, i], 25)
            q3 = np.percentile(data[:, i], 75)
            iqr = q3 - q1
            lower_bound = q1 - threshold * iqr
            upper_bound = q3 + threshold * iqr
            column_outliers = (data[:, i] < lower_bound) | (data[:, i] > upper_bound)
            outliers_mask = outliers_mask | column_outliers
    elif method == 'zscore':
        for i in range(n_features):
            z_scores = np.abs(stats.zscore(data[:, i]))
            column_outliers = z_scores > threshold
            outliers_mask = outliers_mask | column_outliers

    return X[~outliers_mask], y[~outliers_mask], outliers_mask
```

（2）地理特征转换（经纬度联合处理）

transform_geo_features 函数：使用 K-means 聚类将地理位置分为 5 个区域，计算每个样本点到各区域中心的距离作为新特征，并将得到的 5 个距离特征与原有的房龄、收入特征合并。

```python
def transform_geo_features(X, y=None, cluster_centers=None, n_clusters=5):
    geo_coords = X[:, :2].copy()

    if cluster_centers is None:
        kmeans = KMeans(n_clusters=n_clusters, random_state=42)
        kmeans.fit(geo_coords)
        cluster_centers = kmeans.cluster_centers_
    else:
        kmeans = KMeans(n_clusters=len(cluster_centers), random_state=42)
        kmeans.cluster_centers_ = cluster_centers

    dist_to_clusters = np.zeros((len(X), n_clusters))
    for i in range(n_clusters):
        center = cluster_centers[i]
        dist_to_clusters[:, i] = np.sqrt(np.sum((geo_coords - center) ** 2, axis=1))

    other_features = X[:, 2:].copy()
    X_transformed = np.hstack((other_features, dist_to_clusters))

    return X_transformed, cluster_centers
```

（3）MLP 模型

定义了一个 `MLP` 类。

1.声明和初始化：根据指定的网络层级大小（`layer_sizes`）初始化权重和偏置。

```python
class MLP:
    def __init__(self, layer_sizes, learning_rate=0.001, epochs=1000):
        self.layer_sizes = layer_sizes
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.num_layers = len(layer_sizes)
        self.weights = []
        self.biases = []
        self.loss_history = []

        for i in range(1, self.num_layers):
            w = np.random.randn(self.layer_sizes[i-1], self.layer_sizes[i]) * np.sqrt(2.0 / self.layer_sizes[i-1])
            b = np.zeros((1, self.layer_sizes[i]))
            self.weights.append(w)
            self.biases.append(b)
```

2.激活函数：实现了一系列激活函数。

```python
    def tanh(self, x):
        """双曲正切激活函数"""
        return np.tanh(x)

    def tanh_derivative(self, x):
        """双曲正切激活函数的导数"""
        return 1.0 - np.tanh(x)**2

    def relu(self, x):
        """ReLU激活函数"""
        return np.maximum(0, x)

    def relu_derivative(self, x):
        """ReLU激活函数的导数"""
        return np.where(x > 0, 1.0, 0.0)

    def leaky_relu(self, x, alpha=0.01):
        """Leaky ReLU激活函数"""
        return np.maximum(alpha * x, x)

    def leaky_relu_derivative(self, x, alpha=0.01):
        """Leaky ReLU激活函数的导数"""
        return np.where(x > 0, 1.0, alpha)

    def linear(self, x):
        """线性激活函数，直接返回输入"""
        return x

    def linear_derivative(self, x):
        """线性激活函数的导数"""
        return np.ones_like(x)
```

3.向前和向后传输

Forward（向前传播）：实现了混合激活函数策略：根据层的位置选择不同的激活函数（例如，第一隐藏层用 tanh，第二隐藏层用 ReLU，输出层用 linear）。

Backward（向后传播）：根据链式法则和激活函数的导数，逐层反向计算梯度，并使用梯度下降法更新权重和偏置。

```python
    def forward(self, X):
        activations = [X]
        layer_inputs = []

        for i in range(self.num_layers - 1):
            layer_input = np.dot(activations[-1], self.weights[i]) + self.biases[i]
            layer_inputs.append(layer_input)

            if i == 0:
                activation = self.tanh(layer_input)
            elif i == self.num_layers - 2:
                activation = self.linear(layer_input)
            elif i == 1:
                activation = self.relu(layer_input)
            elif i == 2 and self.num_layers > 4:
                activation = self.leaky_relu(layer_input)
            elif i == 3 and self.num_layers > 5:
                activation = self.tanh(layer_input)
            else:
                activation = self.relu(layer_input)
            activations.append(activation)

        return activations, layer_inputs
```

```python
def backward(self, X, y, activations, layer_inputs):
    """
    反向传播，计算梯度并更新权重
    参数:
    X: 输入特征矩阵
    y: 目标向量
    activations: 前向传播中每层的激活值
    layer_inputs: 前向传播中每层的输入值
    """
    n_samples = X.shape[0]

    # 计算输出层误差 (y_pred - y_true)
    output_error = activations[-1] - y.reshape(-1, 1)

    # 初始化当前误差为输出层误差
    delta = output_error

    # 从输出层向前反向传播
    for i in range(self.num_layers - 2, -1, -1):
        # 计算当前层权重的梯度
        dw = np.dot(activations[i].T, delta) / n_samples
        db = np.sum(delta, axis=0, keepdims=True) / n_samples

        # 更新当前层的权重和偏置
        self.weights[i] -= self.learning_rate * dw
        self.biases[i] -= self.learning_rate * db

        # 如果不是第一层，则计算前一层的误差
        if i > 0:
            # 确定当前层使用的激活函数的导数
            if i == 1:  # 第一隐藏层使用tanh
                derivative = self.tanh_derivative(layer_inputs[i-1])
            elif i == 2 and self.num_layers > 4:  # 第三隐藏层使用Leaky ReLU
                derivative = self.leaky_relu_derivative(layer_inputs[i-1])
            elif i == 3 and self.num_layers > 5:  # 第四隐藏层(新增)使用tanh
                derivative = self.tanh_derivative(layer_inputs[i-1])
            else:  # 其他隐藏层使用ReLU
                derivative = self.relu_derivative(layer_inputs[i-1])

            # 计算前一层的误差
            delta = np.dot(delta, self.weights[i].T) * derivative
```

（4）模型训练和评估：我们在这里定义 了 MLP 的网络结构（输入层-20 节点隐层-10 节点隐层-输出层），并实例化 `MLP` 模型，设置学习率和迭代次数。

随后我们使用训练集训练模型，并使用测试集评估模型性能，计算标准化 MSE、原始尺度 MSE 和 $R^2$ (决定系数)。

（5）可视化训练结果：这里生成了以下可视化结果：

数据清理可视化：对比清理前后各特征与房价的关系；
地理聚类可视化：展示 K-means 聚类的区域划分；
损失曲线可视化：展示模型训练过程中损失的变化片；
MSE 比较可视化：比较不同参数组合的模型性能；
预测结果可视化：展示真实值与预测值的对比，以及特征与房价的关系；
特征重要性可视化：展示各特征对预测结果的影响程度。

将数据可视化处理部分的代码在此省略，具体请见附件

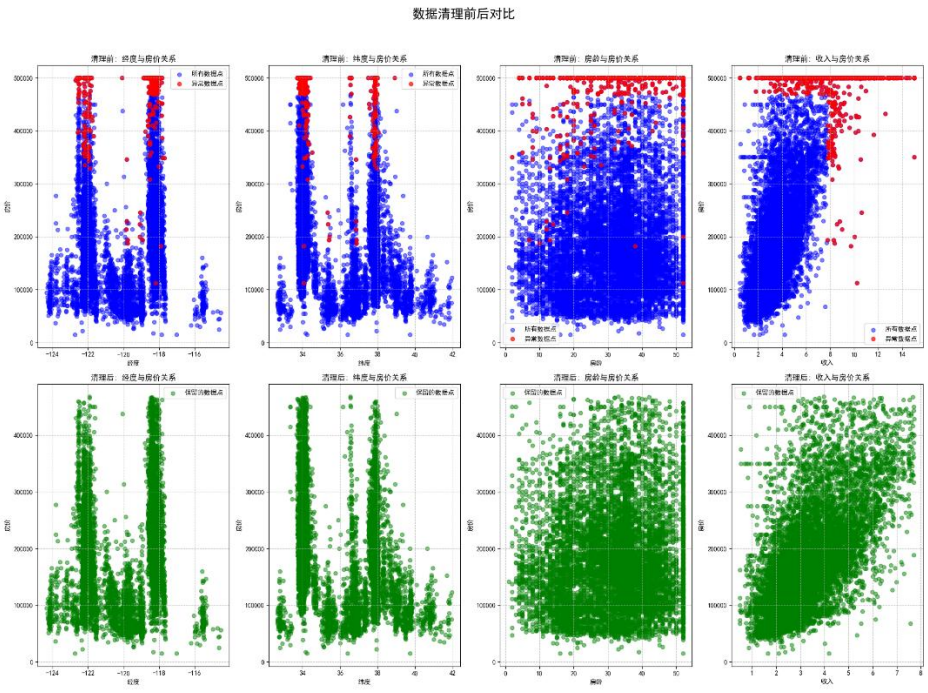
# 四、创新点&优化

1. **混合使用多种激活函数策略**:
   第一隐藏层使用 tanh 激活函数，适合捕捉初始非线性特征;
   中间层使用 ReLU 激活函数，缓解梯度消失问题;
   输出层使用线性激活函数，适合回归任务;
2. **经纬度联合考量**:
   采用 K-means 聚类将经纬度转换为到聚类中心的距离特征;
   有效捕捉地理位置的非线性影响;
3. **异常值提前处理优化**:
   实现 IQR 和 Z-score 两种异常值检测方法;
   可视化展示异常值处理前后的数据分布对比;
4. **训练过程有完整的可视化过程**:
   绘制不同参数组合的损失曲线;
   实现神经网络架构可视化功能;

# 五、实验结果及分析

## （一）实验结果展示

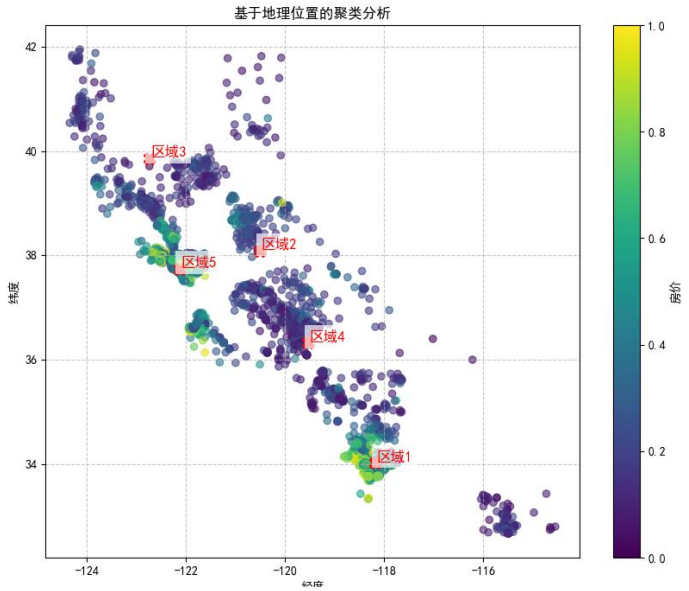
### （1）实验数据预处理结果




```
加载数据...
检测并处理异常值...
原始数据样本数：10000
处理后数据样本数：9320
被移除的异常样本数：680（6.80%）
```

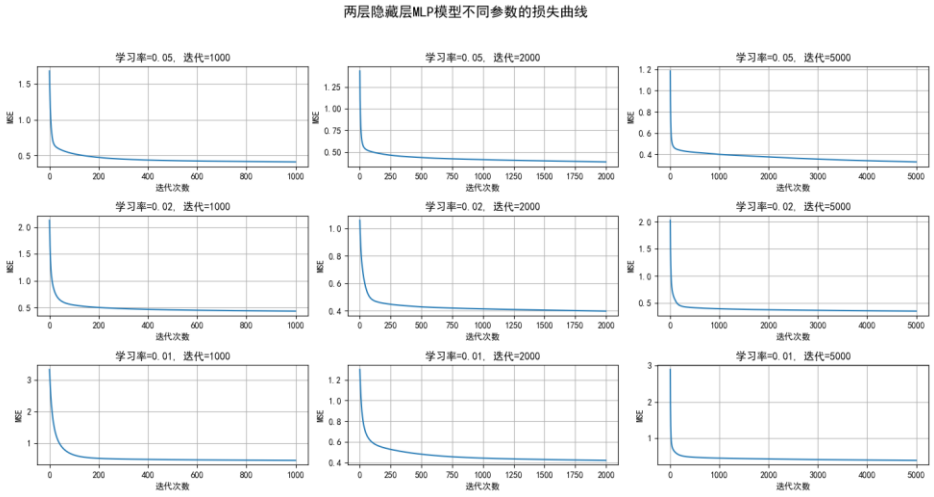
如上图，红色的点为被清除的异常数据，一共清理了 9320 个样本。

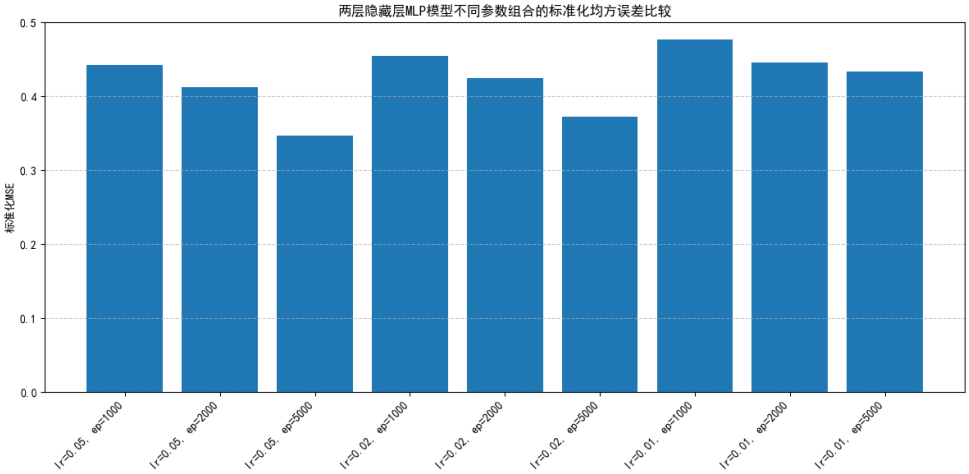
（2）经纬度处理结果




如上图，采用 Kmeans 聚类方法划分了五个区域。

（3）不同参数组合的比较选择



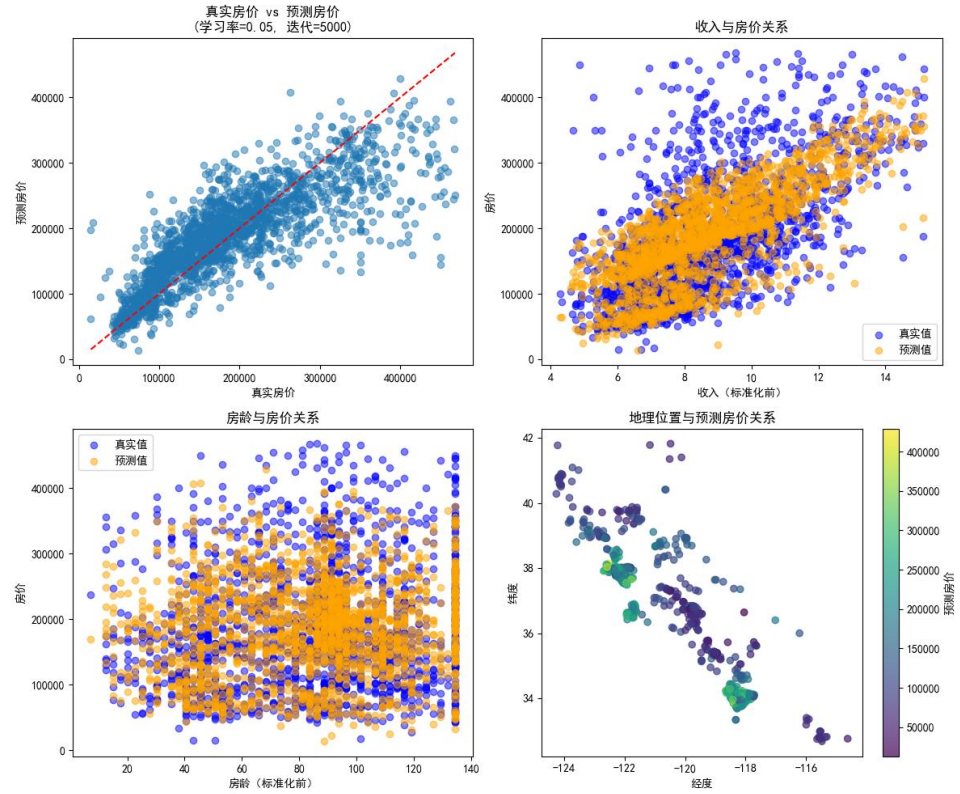
经过实践，学习率过高会导致数据不收敛，而学习率过低会导致曲线下降过快，可能会错过最优解。

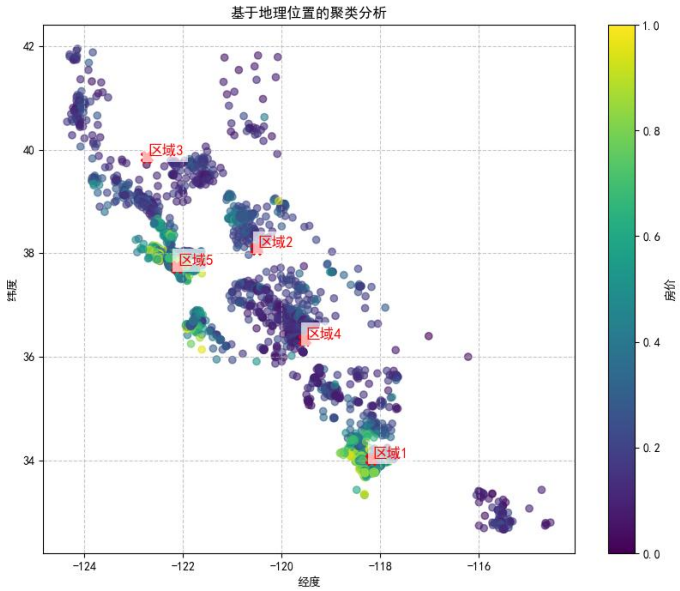
迭代次数过低，曲线可能未达到稳定，继续迭代会出现更好的结果，而迭代次数过高，迭代收益大幅下降的同时计算成本显著增加。

由上面两个图可以看出：学习率为 0.05，迭代次数为 5000 次时效果最好。

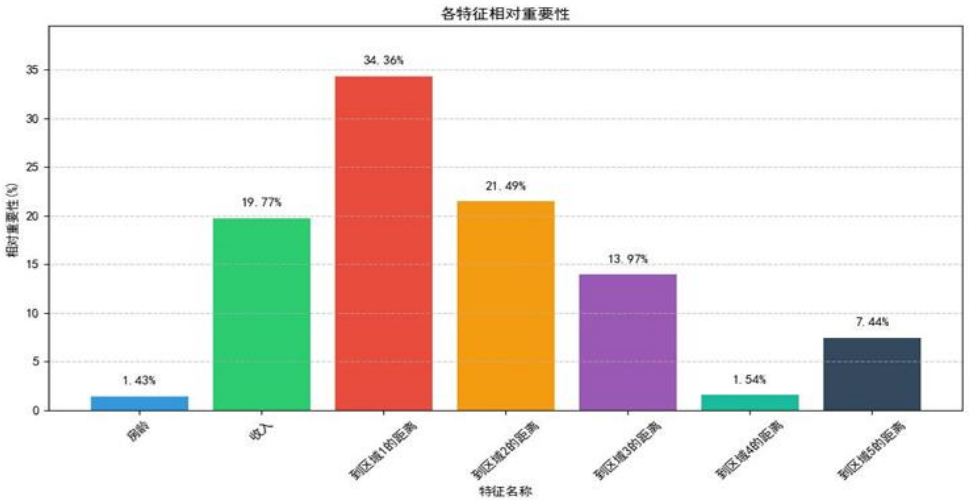
（4）最好的预测结果展示

```
选定的最佳两层隐藏层MLP模型参数 – 学习率：0.05，迭代次数：5000，标准化MSE：0.346765
最佳两层隐藏层MLP模型 R²(决定系数)：0.6621
```



（二）模型性能分析:




(1) 模型性能评估:

标准化 MSE=0.346765，$R^2$=0.6621（决定系数）

模型能够解释目标变量（房价）62.17%的方差，剩余约 38%的波动 未被捕捉。

在房价预测问题中，$R^2$通常期望达到 0.5~0.8（中等以上），考虑到房价还有其他因素影响，且采用的只是简单的线性预测模型，当前结果合理但仍有提升空间。则当前模型有一定预测价值，但需进一步优化。

(2) 参数合理性分析:

学习率：0.05 是中等偏大的学习率，适合快速收敛。 迭代次数：5000 次迭代可能已使模型接近收敛，但需检查训练曲线：训练损失和验证损失均平稳，说明迭代足够;

(3) 变量重要性分析:

（1）到区域 1 的距离（34.36%）：绝对重要性最高（0.9628），可能反映 核心区位价值（如市中心、商业区）。

（2）收入（19.77%）：重要性次之（0.5538），符合经济常识（收入高的 人群购房能力更强）。

（3）到区域 2，3 的距离（合计 35.46%）：可能与区域 1 形成互补（如次 人工智能实验 级商圈或居住区），需分析这些区域的实际功能。

（4）到区域 5 的距离（7.44%）：贡献较低但不可忽略，可能对应郊区或特定设施（如公园）。

（5）房龄（1.43%）和到区域 4 的距离（1.54%）：影响微弱，但需验证是 否因数据分布或模型缺陷导致低估（如房龄与房价实际为非线性关系）。

（4）后续改进建议

升级模型：尝试非线性模型（如 XGBoost、随机森林），自动捕捉变量间复杂关系。

数据增强：补充特征：加入房屋面积、卧室数量、周边设施（地铁、学校）等关键因子。

数据变换：对“距离”类变量尝试分箱或对数变换，解决非线性问题。

# 六、参考资料

详细代码请见附件