深度学习报告

July 5, 2025

摘要

本项目旨在介绍商业智能选址预测的深度学习方法。通过分析品牌历史门店的地理分布数据，建立能够预测品牌下一家门店最优选址的模型。项目以网格（Grid）为基本地理单元，利用品牌在城市中的历史开店网格序列及每个网格的地理属性特征，挖掘品牌扩张的空间模式和内在偏好。通过对历史数据的建模与学习，预测品牌未来最有可能选址的网格位置。模型由序列编码器、空间编码器和环境编码器组成，采用多模态融合设计。实验结果表明，该模型在选址预测任务中表现优异，能够为商业决策提供科学依据。

关键词

选址预测, 深度学习, 网格模型, 多模态融合, 序列编码器, 空间编码器, 环境编码器

目录

1 引言

在现代商业环境中，选址决策是品牌扩张和市场布局过程中至关重要的一环。一个优质的门店位置不仅能够提升品牌曝光度和客流量，还直接影响企业的经济效益和市场竞争力。传统的选址方法主要依赖于专家经验和简单的人口统计分析，存在主观性强、难以量化、适应性差等局限。随着大数据、机器学习和深度学习技术的快速发展，基于数据驱动的选址预测方法为选址预测提供更多的思路启发，其能够通过对历史数据的深入分析，挖掘出潜在的空间模式和内在偏好，从而实现更科学、精准、有效的选址决策。

本课题聚焦于“商业智能选址预测”，旨在通过分析品牌历史门店的地理分布数据，建立能够预测品牌下一家门店最优选址的模型。具体而言，课题以网格（Grid）为基本地理单元，利用品牌在城市中的历史开店网格序列及每个网格的地理属性特征，挖掘品牌扩张的空间模式和内在偏好。通过对历史数据的建模与学习，预测品牌未来最有可能选址的网格位置，为企业提供科学、量化的决策依据。

本项目的数据集包含多个品牌的历史门店分布（训练集和测试集），以及覆盖研究区域的网格地理坐标信息。提供数据已划分为训练集和测试集，网格划分保证了空间分析的精度和一致性。研究区域的经纬度范围明确，便于空间特征的提取和建模。

2 相关工作

项目提供了多种基线（Baseline）模型或方法，包括简单模型和深度模型。在项目文档中，对比了多个基线方案的实验结果，展示了不同方法在选址预测任务中的表现，作为模型实现方案的参考和对比。

随机猜测（Random guess）方法，将所有未出现过的网格作为候选集，随机选取若干作为预测结果，作为最简单的基线 [1]。一般认为，一个模型至少有一点作用，就可以将它和随机猜测进行对比，如果这个模型居然结果还不如随机猜测，那么这个模型就没有意义了，一定存在很明显的缺陷，比如严重的过拟合、特征选择不当、数据标签出现错误、数据分布出现重大偏差等 [2]。

随着空间数据挖掘的发展，基于 Word2vec 的嵌入方法被用到选址任务中 [3]。这种方法，虽然最初被用于自然语言处理，但是其本质是一种时序性的序列预测方法 [4]，因此也可以被利用在这个任务中。该方法通过 skip-gram 模型训练每个网格的 embedding，测试时将品牌历史网格序列的 embedding 取平均，计算与候选网格 embedding 的相似度（如点积、余弦相似度等），并据此排序预测结果。

深度学习方法方面，LSTM+MLP 结构利用嵌入层将输入网格序列编码为向量，随后通过 LSTM 捕捉序列中的时序依赖，最后通过多层感知机（MLP）进行分类预测。Transformer+MLP 则用 Transformer 替代 LSTM，利用自注意力机制更好地建模序列中各网格之间的复杂关系。进一步地，基于注意力融合和对比学习的方法，通过对输入序列进行 self-attention，获得全局语义表达，并采用 InfoNCE 损失进行正负样本的对比训练，有效提升了模型的判别能力和泛化性能。

上述方法在本项目数据集上的实验结果如表所示。可以看出，深度学习模型（如 Transformer+MLP、对比学习方法）在准确率和平均排序等指标上均优于传统方法，尤其是对比学习方法在 acc@1 和 acc@10 等指标上表现突出，显示了其在空间选址预测任务中的潜力。此外，模型结构的层级设计（如嵌入层、序列建模层、融合层和输出层）对最终性能有显著影响，合理的结构选择和层级组合能够更好地捕捉空间和序列特征，从而提升预测效果。

3 方法探索

3.1 数据预处理

数据集中有已经分配好的训练集和测试集。数据的内容主要是品牌历史门店的选址信息。具体分析数据结构如表1所示。

字段	类型	描述
brand_name	string	品牌名称
brand_type	string(csv)	品牌类型（如餐饮、零售等）
longtitude_list	list(float)	网格经度列表
latitude_list	list(float)	网格纬度列表
grid_id_list	list(int)	网格 ID 列表

表 1: 数据集字段说明

此外还通过了网格数据，给出对应网格 ID 在真实地理世界中的经纬度坐标。网格数据的结构如表2所示。

字段	类型	描述
grid_id	int	网格 ID
grind_lon_min	float	网格左下角经度
grind_lon_max	float	网格右上角经度
grind_lat_min	float	网格左下角纬度
grind_lat_max	float	网格右上角纬度

表 2: 网格数据字段说明

3.1.1 数据加载

数据加载是数据预处理的第一步。我们需要从提供的训练集和测试集文件中读取品牌历史门店的选址信息，并将其转换为适合模型输入的格式，提交给神经网络。

3.1.2 网格信息加载

“坐标”含有的信息过于的稀少，难以直接被模型“理解”，从而掌握其中内在的一些联系。需要对数据增加有关的信息，以扩展其表达能力，否则模型表达能力的上限就会受到限制 [5]。为了得出更加合理的结论，必须在数据预处理阶段强化并且增加特征。

网格（Grid）是商业选址预测任务中的基本地理单元。在原始数据集里，每个网格由其唯一的网格 ID 和对应的地理坐标（经纬度）组成。这个体现了网格在空间上的位置信息。但是这个信息并不能很有效地描述网格的全部特性。为了增强网格的信息，需要从数据集中重新挖掘出可能存在的信息。兴趣点（POI, Point of Interest）是地理信息系统（GIS）和位置服务中非常核心的概念，它指的是地图上具有特定意义或吸引力的地点。它代表一个具有空间位置和语义信息的地理目标，比如餐厅、医院、公交站、商场、景点等，在经纬度以外，其还通常包括有名称、地址、类别等信息，并经常以向量的形式表示。POI 可以将三维世界中的复杂实体抽象为一个零维点，从而便于管理、分析和计算 [6]。

因此，可以从全部的门店数据中，统计每个网格的相关信息，再将其关联到网格 ID 上，构成更加立体、具有丰富特征的高维网格对象。具体而言，再项目实现上，从网格坐标文件中提取每个网格的地理信息，包括经纬度坐标和 10 种类型的 POI 特征（医疗、住宿、摩托、体育、餐饮、公司、购物、生活、科教、汽车）。这种类型的统计信息，虽然引入了一些和其他若干特征高度相关的特征，但是会在大型复杂模型的训练上甚至有性能提升的作用 [7]。

3.1.3 特征归一化

为了消除不同特征间的尺度差异，我们对坐标和 POI 特征分别进行归一化处理：

对于坐标特征，我们计算网格的中心点坐标，然后将其归一化到 [0,1] 区间：

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min} + \epsilon} \tag{1}$$

对于 POI 特征，我们同样采用归一化方法，确保不同类型 POI 特征的数值在同一尺度上。

3.1.4 基于密度的序列排序

考虑到商业扩张通常遵循” 从高密度区域向外扩展” 的规律，我们设计了基于密度的序列排序算法。该算法通过计算每个网格到其 K 个最近邻网格的平均距离来衡量密度，距离越小表示密度越大：

$$density\_score_i = \frac{1}{K}\sum_{j=1}^{K} d(grid_i, neighbor_j) \tag{2}$$

其中 $d(grid_i, neighbor_j)$ 表示网格 i 到其第 j 个最近邻的欧氏距离。通过这种排序方式，我们能够构建符合商业扩张规律的序列，提高模型的学习效果。

3.1.5 滑动窗口样本生成

为了最大化利用有限的训练数据，我们采用滑动窗口方法从每个品牌的店铺序列中生成多个训练样本。例如，对于序列 [A,B,C,D]，我们生成以下样本：

- 已有 [A]，预测 B
- 已有 [A,B]，预测 C
- 已有 [A,B,C]，预测 D

这种处理方法不仅增加了训练样本数量，还使模型能够学习不同长度序列的预测模式，更好地模拟品牌实际扩张过程中的决策链。

3.2 对于标签的处理尝试

3.3 数据增强和特征增强

3.4 多模态特征融合设计

针对商业选址预测任务的特点，我们设计了一个多模态融合的神经网络架构，该架构能够同时处理和融合三种不同类型的信息：

3.4.1 序列特征编码

历史选址序列包含了品牌扩张的时序信息和空间模式。我们使用嵌入层将网格 ID 映射到低维向量空间，然后通过 LSTM 网络捕获序列中的时序依赖关系：

$$h_t = LSTM(embed(grid\_id_t), h_{t-1}) \tag{3}$$

LSTM 的长短期记忆特性使其能够有效学习品牌在不同时期的选址偏好变化。

3.4.2 空间特征编码

地理坐标信息反映了选址的空间连续性。我们通过多层感知机 (MLP) 对归一化后的坐标特征进行编码，提取空间分布模式。

3.4.3 环境特征编码

POI 特征反映了网格周边的商业环境和设施分布。同样使用 MLP 对 POI 特征向量进行编码，捕获环境因素对选址决策的影响。

3.4.4 特征融合策略

我们采用特征拼接的方式将三种编码后的特征向量组合，然后通过多层融合网络进一步提取联合特征：

$$f_{fused} = MLP_{fusion}(concat(f_{seq}, f_{coord}, f_{poi})) \tag{4}$$

最终通过线性分类器预测下一个网格的概率分布。

3.5 模型最终设计

考虑到数据集中特征的特点，模型采用时序的方案进行建模。整体架构采用多模态融合设计，具体包括：

- 序列编码器：基于嵌入层和 LSTM 的序列特征提取
- 空间编码器：基于 MLP 的坐标特征编码
- 环境编码器：基于 MLP 的 POI 特征编码

- 融合网络：多层感知机进行特征融合
- 分类器：线性层输出概率分布

该设计充分考虑了商业选址的多因素特性，通过端到端的学习方式自动发现不同模态特征间的内在关联，为精准的选址预测提供了有力支撑。

参考文献

[1] T. G. Dietterich, “Approximate statistical tests for comparing supervised classification learning algorithms,” *Neural Computation*, vol. 10, no. 7, pp. 1895–1923, 1998.

[2] P. Domingos, “A few useful things to know about machine learning,” *Commun. ACM*, vol. 55, no. 10, p. 78–87, Oct. 2012. [Online]. Available: https://doi.org/10.1145/2347736.2347755

[3] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” 2013. [Online]. Available: https://arxiv.org/abs/1301.3781

[4] A. Mohan and K. V. Pramod, “Link prediction in dynamic networks using time-aware network embedding and time series forecasting,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 2, pp. 1981–1993, Feb 2021. [Online]. Available: https://doi.org/10.1007/s12652-020-02289-0

[5] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. USA: Cambridge University Press, 2014. [Online]. Available: https://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/understanding-machine-learning-theory-algorithms.pdf

[6] A. Psyllidis, S. Gao, Y. Hu, E.-K. Kim, G. McKenzie, R. Purves, M. Yuan, and C. Andris, “Points of interest (POI): a commentary on the state of the art, challenges, and prospects for the future,” *Computational Urban Science*, vol. 2, no. 1, p. 20, 2022. [Online]. Available: https://doi.org/10.1007/s43762-022-00047-w

[7] J. Heaton, “An empirical analysis of feature engineering for predictive modeling,” in *SoutheastCon 2016*. IEEE, Mar. 2016, p. 1 – 6. [Online]. Available: http://dx.doi.org/10.1109/SECON.2016.7506650

A 附录

A.1 项目信息

GitHub 仓库地址： `https://github.com/szw0407/DL-project-2025`

A.2 核心模型代码

A.2.1 神经网络模型实现 **(model.py)**

```
"""
商业选址预测模型

整体模型架构:
┌─────────────────────────────────────────────────────────┐
│                    NextGridPredictor                    │
└─────────────────────────────────────────────────────────┘
          │               │               │
          ▼               ▼               ▼
┌──────────────┐  ┌──────────────┐  ┌──────────────┐
│ SeqEncoder   │  │ CoordEncoder │  │ POIEncoder   │
│ (LSTM-based) │  │  (MLP-based) │  │  (MLP-based) │
└──────────────┘  └──────────────┘  └──────────────┘
          │               │               │
          └───────────────┼───────────────┘
                          │
                          ▼
                 ┌──────────────┐
                 │   Fusion Layer   │
                 │    (MLP with     │
```

```
                    |     Dropout)     |
                    └──────────────────┘
                             |
                             ▼
                    ┌──────────────────┐
                    |    Classifier    |
                    |  (Linear Layer)  |
                    └──────────────────┘
                             |
                             ▼
                    预测下一个网格的概率分布

该模型采用多模态融合架构，综合利用三种信息进行商业选址预测：
1. 历史选址序列信息 (通过LSTM编码)
2. 地理坐标信息 (通过MLP编码)
3. 兴趣点(POI)特征信息 (通过MLP编码)

这三种信息经过各自的编码器处理后，在特征层面融合，
再通过多层感知机进行进一步特征提取，最终用于预测下一个最佳选址位置。
"""
import torch
import torch.nn as nn

class SeqEncoder(nn.Module):
    """
    序列编码器：用于编码历史选址序列的特征

    该编码器首先将网格ID映射到低维嵌入空间，然后通过LSTM网络捕获序列中的
    时序依赖关系和选址模式。使用LSTM的优势在于能够记忆长期依赖，
    适合捕捉商业选址中的空间扩张规律。

    架构：
    输入序列 → 嵌入层 → LSTM → 最后时刻的隐藏状态

    参数：
        vocab_size: 词汇表大小，即网格总数
        embed_dim: 嵌入维度
        lstm_hidden: LSTM隐藏层维度
        lstm_layers: LSTM层数
        dropout: Dropout比率，用于防止过拟合
    """
    def __init__(self, vocab_size, embed_dim, lstm_hidden, lstm_layers, dropout):
        super().__init__()
        self.embed = nn.Embedding(vocab_size, embed_dim)
        self.lstm = nn.LSTM(embed_dim, lstm_hidden, lstm_layers,
                            batch_first=True, dropout=dropout if lstm_layers > 1 else 0)

    def forward(self, seq_ids):
        """
        前向传播

        参数：
            seq_ids: 网格ID序列，形状为(batch_size, seq_len)

        返回：
            序列的最终表示，形状为(batch_size, lstm_hidden)
        """
        emb = self.embed(seq_ids)  # (batch_size, seq_len, embed_dim)
        out, (h, _) = self.lstm(emb)  # out: (batch_size, seq_len, lstm_hidden)
        return out[:, -1, :]  # 返回最后一个时间步的输出作为序列表示

class MLPEncoder(nn.Module):
    """
    多层感知机编码器：用于编码坐标和POI特征

    这是一个简单的两层感知机，通过非线性变换将输入特征映射到固定维度的
    输出空间。适用于处理非序列性的数值特征，如坐标和POI统计数据。

    架构：
    输入特征 → 线性层 → ReLU → 线性层 → ReLU

    参数：
        in_dim: 输入特征维度
        out_dim: 输出特征维度
        hidden_dim: 隐藏层维度，默认为32
    """
```

```
    def __init__(self, in_dim, out_dim, hidden_dim=32):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(in_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, out_dim),
            nn.ReLU()
        )

    def forward(self, x):
        """
        前向传播

        参数:
            x: 输入特征, 形状为(batch_size, in_dim)

        返回:
            编码后的特征, 形状为(batch_size, out_dim)
        """
        return self.model(x)

class NextGridPredictor(nn.Module):
    """
    下一个网格预测器: 预测品牌下一个最佳选址位置

    这是整个模型的主体部分, 采用多模态融合方法, 将三种不同类型的信息
    (历史选址序列、地理坐标、POI特征)整合起来, 共同预测下一个最佳选址位置。

    模型处理流程:
    1. 通过各自的编码器分别编码三种信息
    2. 将编码后的特征向量拼接起来
    3. 通过多层感知机进行特征融合和提取
    4. 最终通过线性分类器预测下一个网格的概率分布

    多模态融合的优势:
    - 序列信息: 捕捉品牌扩张的时序模式和依赖关系
    - 坐标信息: 考虑地理位置的连续性和空间分布
    - POI信息: 考虑周边设施和商业环境的影响

    参数:
        num_classes: 类别数量, 即网格总数
        embed_dim: 网格ID的嵌入维度, 默认为32
        lstm_hidden: LSTM隐藏层维度, 默认为64
        lstm_layers: LSTM层数, 默认为1
        coord_dim: 坐标特征维度, 默认为2 (经度、纬度)
        poi_dim: POI特征维度, 默认为10 (10种POI类型)
        coord_out_dim: 坐标编码后的维度, 默认为16
        poi_out_dim: POI编码后的维度, 默认为16
        fusion_dim: 特征融合后的维度, 默认为64
        dropout: Dropout比率, 用于防止过拟合, 默认为0.1
    """
    def __init__(self, num_classes, embed_dim=32, lstm_hidden=64, lstm_layers=1,
                 coord_dim=2, poi_dim=10, coord_out_dim=16, poi_out_dim=16, fusion_dim=64,
     dropout=0.1):
        super().__init__()
        # 序列编码器: 处理历史选址序列
        self.seq_encoder = SeqEncoder(num_classes, embed_dim, lstm_hidden, lstm_layers,
     dropout)
        # 坐标编码器: 处理地理坐标信息
        self.coord_encoder = MLPEncoder(coord_dim, coord_out_dim)
        # POI编码器: 处理兴趣点特征信息
        self.poi_encoder = MLPEncoder(poi_dim, poi_out_dim)
        # 特征融合网络: 整合三种编码后的特征
        self.fusion = nn.Sequential(
            nn.Linear(lstm_hidden + coord_out_dim + poi_out_dim, fusion_dim),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(fusion_dim, fusion_dim),
            nn.ReLU(),
            nn.Dropout(dropout)
        )
        # 分类器: 预测下一个网格的概率分布
        self.classifier = nn.Linear(fusion_dim, num_classes)
    def forward(self, seq_ids, seq_coords, seq_poi):
        """
        前向传播
```

```
        数据流程:
        1. 各编码器独立处理对应的输入特征
        2. 对坐标和POI特征取序列平均值，降维处理
        3. 拼接所有特征向量
        4. 通过融合网络进一步提取联合特征
        5. 通过分类器预测下一个网格的概率分布

        参数:
            seq_ids: 网格ID序列，形状为(batch_size, seq_len)
            seq_coords: 坐标序列，形状为(batch_size, seq_len, 2)
            seq_poi: POI特征序列，形状为(batch_size, seq_len, 10)

        返回:
            logits: 下一个网格的预测概率分布，形状为(batch_size, num_classes)
        """
        # 输入: (batch, seq_len, dim)
        seq_out = self.seq_encoder(seq_ids)               # (batch, lstm_hidden)

        # 取坐标序列的平均值，简化处理同时保留整体分布信息
        coords_out = self.coord_encoder(seq_coords.mean(dim=1)) # (batch, coord_out_dim)

        # 取POI特征序列的平均值，同样简化处理
        poi_out = self.poi_encoder(seq_poi.mean(dim=1))   # (batch, poi_out_dim)

        # 特征拼接：将三种编码后的特征向量拼接起来
        x = torch.cat([seq_out, coords_out, poi_out], dim=-1)

        # 特征融合：通过MLP进一步提取联合特征
        f = self.fusion(x)

        # 分类预测：预测下一个网格的概率分布
        logits = self.classifier(f)
        return logits
```

Listing 1: 多模态神经网络模型实现

A.2.2 主程序入口 (main.py)

```
import torch
from data_preprocessing import load_all_data
from model import NextGridPredictor
from train import train_model
from evaluate import evaluate_model

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

train_csv = 'data/train_data.csv'
test_csv = 'data/test_data.csv'
grid_csv = 'data/grid_coordinates-2.csv'
if __name__ == '__main__':
    print("加载并处理数据...")
    train_set, val_set, test_set, num_classes, grid2idx = load_all_data(
        train_csv, test_csv, grid_csv, val_size=0.2
    )

    print(f"训练样本数: {len(train_set)}, 验证样本数: {len(val_set)}, 测试样本数: {len(
     test_set)}")
    model = NextGridPredictor(num_classes=num_classes)

    print("开始训练...")
    model = train_model(model, train_set, val_set, device, num_epochs=40, batch_size=32, lr
     =1e-3, patience=5)

    print("在测试集上评估...")
    acc_k, mrr = evaluate_model(model, test_set, device)
    print(f"Test_MRR: {mrr:.4f}")
    for k in [1, 5, 10]:
        print(f"Test_Acc@{k}: {acc_k[k]:.3f}")
```

Listing 2: 主程序实现

A.2.3 训练逻辑实现 (train.py)

```python
"""
模型训练模块
该模块提供了模型训练的相关函数，包括训练数据批处理和模型训练流程。
实现了带有早停机制的模型训练，以及基于验证集性能的模型选择。
"""
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
from evaluate import evaluate_model

def batch_iter(samples, batch_size=32, shuffle=True):
    """
    训练数据批次迭代器

    与evaluate模块中的batcher函数类似，但增加了数据打乱功能，
    适用于训练过程中需要随机打乱数据的场景。

    参数:
        samples: 样本列表，每个样本是(前缀索引，前缀坐标，前缀POI特征，目标索引)的元组
        batch_size: 批次大小，默认为32
        shuffle: 是否打乱数据顺序，默认为True

    生成:
        每次生成一个批次的数据，包含:
        - seq_ids: 序列ID张量，形状为(batch_size, max_seq_len)
        - seq_coords: 序列坐标张量，形状为(batch_size, max_seq_len, 2)
        - seq_poi: 序列POI特征张量，形状为(batch_size, max_seq_len, 10)
        - targets: 目标索引张量，形状为(batch_size,)
    """
    idxs = np.arange(len(samples))
    if shuffle:
        np.random.shuffle(idxs)  # 随机打乱索引，增加训练随机性
    for i in range(0, len(samples), batch_size):
        batch = [samples[j] for j in idxs[i:i+batch_size]]
        maxlen = max(len(x[0]) for x in batch)
        # pad to maxlen
        seq_ids = np.zeros((len(batch), maxlen), dtype=np.int64)
        seq_coords = np.zeros((len(batch), maxlen, 2), dtype=np.float32)
        seq_poi = np.zeros((len(batch), maxlen, 10), dtype=np.float32)
        for i, (pidx, pcoord, ppoi, _) in enumerate(batch):
            L = len(pidx)
            # 右对齐填充：将数据放在数组的右侧，左侧用0填充
            seq_ids[i, -L:] = pidx
            seq_coords[i, -L:, :] = pcoord
            seq_poi[i, -L:, :] = ppoi
        targets = np.array([x[3] for x in batch], dtype=np.int64)
        yield (
            torch.from_numpy(seq_ids),
            torch.from_numpy(seq_coords),
            torch.from_numpy(seq_poi),
            torch.from_numpy(targets)
        )

def train_model(model, train_set, val_set, device, num_epochs=40, batch_size=32, lr=1e-3,
    patience=5):
    """
    训练模型的主函数

    实现了完整的模型训练流程，包括:
    1. 模型训练与参数优化
    2. 验证集评估
    3. 早停机制
    4. 最佳模型保存

    使用MRR(平均倒数排名)作为模型选择的指标，
    当验证集上的MRR不再提升时，触发早停机制。

    参数:
        model: 待训练的模型
        train_set: 训练数据集
        val_set: 验证数据集
        device: 计算设备(CPU/GPU)
```

```
        num_epochs: 最大训练轮数，默认为40
        batch_size: 批次大小，默认为32
        lr: 学习率，默认为1e-3
        patience: 早停耐心值，当验证指标连续多少轮未改善时停止训练，默认为5

    返回:
        训练完成的模型（已加载最佳状态）
    """
    model = model.to(device)  # 将模型移至指定设备
    opt = optim.Adam(model.parameters(), lr=lr)  # Adam优化器
    criterion = nn.CrossEntropyLoss()  # 交叉熵损失函数，适用于多分类问题
    best_mrr = -1  # 最佳MRR值
    best_state = None  # 最佳模型状态
    no_improve = 0  # 未改善轮数计数器

    for epoch in range(1, num_epochs+1):
        # 训练阶段
        model.train()  # 设置模型为训练模式
        losses = []  # 记录每个批次的损失

        for seq_ids, seq_coords, seq_poi, targets in batch_iter(train_set, batch_size):
            # 将数据移至指定设备
            seq_ids, seq_coords, seq_poi, targets = (
                seq_ids.to(device), seq_coords.to(device), seq_poi.to(device), targets.to(
 device)
            )
            opt.zero_grad()  # 梯度清零
            logits = model(seq_ids, seq_coords, seq_poi)  # 前向传播
            loss = criterion(logits, targets)  # 计算损失
            loss.backward()  # 反向传播
            opt.step()  # 参数更新
            losses.append(loss.item())  # 记录损失值

        # 验证阶段
        val_acc_k, val_mrr = evaluate_model(model, val_set, device)

        # 打印当前训练状态
        print(f"Epoch {epoch} | loss={np.mean(losses):.4f} | Val_MRR={val_mrr:.4f} | Acc@1={
 val_acc_k[1]:.3f} Acc@5={val_acc_k[5]:.3f} Acc@10={val_acc_k[10]:.3f}")

        # 模型选择与早停机制
        if val_mrr > best_mrr:  # 如果验证集MRR有提升
            best_mrr = val_mrr  # 更新最佳MRR
            best_state = model.state_dict()  # 保存当前模型状态
            no_improve = 0  # 重置未改善计数器
        else:
            no_improve += 1  # 未改善计数器加1
            if no_improve >= patience:  # 如果连续patience轮未改善
                print("Early stop triggered.")  # 触发早停
                break

    # 加载最佳模型状态
    if best_state is not None:
        model.load_state_dict(best_state)

    return model
```

Listing 3: 模型训练实现

A.2.4 数据预处理模块 **(data_preprocessing.py)**

```
"""
数据预处理模块
该模块用于处理位置预测任务的数据，包括网格信息加载、样本生成和数据集划分。
主要实现了基于密度的店铺序列排序和序列到预测样本的转换。
"""
import pandas as pd
import numpy as np
import ast
from sklearn.model_selection import train_test_split

def parse_list(s):
    """
```

```
    将字符串形式的列表解析为Python列表对象

    参数:
        s: 字符串表示的列表，如 "[1, 2, 3]"

    返回:
        解析后的Python列表，若解析失败则返回空列表
    """
    try:
        return ast.literal_eval(s)
    except Exception:
        return []

def load_grid_info(grid_csv_path):
    """
    加载网格信息，包括坐标和POI特征，并进行归一化处理

    归一化的目的是将不同尺度的特征转换到同一尺度，提高模型训练效果。
    坐标归一化到[0,1]区间，使模型对不同区域的预测更加公平。
    POI特征归一化避免某些数值较大的特征主导模型训练。

    参数:
        grid_csv_path: 网格信息CSV文件路径

    返回:
        coords_map: 网格ID到归一化坐标的映射字典
        poi_feat_map: 网格ID到归一化POI特征的映射字典
    """
    grid_df = pd.read_csv(grid_csv_path, encoding='gbk')
    coords_map = {}
    poi_feat_map = {}
    poi_columns = ['医疗', '住宿', '摩托', '体育', '餐饮', '公司', '购物', '生活', '科教', '
     汽车']
    for _, row in grid_df.iterrows():
        gid = int(row["grid_id"])
        x = (row["grid_lon_min"] + row["grid_lon_max"]) / 2.0
        y = (row["grid_lat_min"] + row["grid_lat_max"]) / 2.0
        coords_map[gid] = (x, y)
        poi_feat_map[gid] = row[poi_columns].values.astype(float)
    # 归一化空间和poi
    xs, ys = zip(*coords_map.values())
    x_min, x_max = min(xs), max(xs)
    y_min, y_max = min(ys), max(ys)
    for gid in coords_map:
        x, y = coords_map[gid]
        coords_map[gid] = [(x - x_min) / (x_max - x_min + 1e-8), (y - y_min) / (y_max -
     y_min + 1e-8)]
    all_poi = np.stack(list(poi_feat_map.values()))
    poi_min, poi_max = all_poi.min(axis=0), all_poi.max(axis=0)
    for gid in poi_feat_map:
        poi_feat_map[gid] = (poi_feat_map[gid] - poi_min) / (poi_max - poi_min + 1e-8)
    return coords_map, poi_feat_map

def sort_by_density(gid_list, coords_map):
    """
    基于密度对网格ID进行排序

    算法原理:
    计算每个点到其他K个最近邻点的平均距离，距离越小表示该点周围密度越大。
    通过这种排序方式，我们可以从高密度区域到低密度区域构建序列，
    这种顺序更符合商业扩张的实际规律（先在核心区域布局，再扩展到周边）。

    参数:
        gid_list: 网格ID列表
        coords_map: 网格ID到坐标的映射字典

    返回:
        按照密度排序后的网格ID列表
    """
    if len(gid_list) <= 1: return gid_list[:]
    k = 3  # 考虑最近的3个邻居点
    locs = [coords_map[g] for g in gid_list]
    scores = []
    for i, g in enumerate(gid_list):
        xi, yi = locs[i]
```

```
        dists = [float(np.linalg.norm([xi-xj, yi-yj])) for j, (xj, yj) in enumerate(locs) if i != j]
        # 计算到K个最近邻点的平均距离，如果点不足K个，则取所有点
        # 若没有其他点，则设置一个极大值表示最低密度
        avg = np.mean(sorted(dists)[:min(k, len(dists))]) if dists else 1e5
        scores.append((avg, g))
    scores.sort()  # 按平均距离排序，距离小的（密度大的）排前面
    return [g for _, g in scores]

def make_samples(data_csv_path, coords_map, poi_feat_map, grid2idx, max_seq_len=10):
    """
    构建序列预测样本

    采用滑动窗口方法，从每个品牌的店铺序列中生成多个训练样本：
    例如序列[A,B,C,D]会生成样本：
    - 已有[A]，预测B
    - 已有[A,B]，预测C
    - 已有[A,B,C]，预测D

    这种处理方法能够：
    1. 最大化利用有限的数据
    2. 学习不同长度序列的预测模式
    3. 模拟品牌实际扩张过程中的决策链

    参数：
        data_csv_path: 数据CSV文件路径
        coords_map: 网格ID到坐标的映射字典
        poi_feat_map: 网格ID到POI特征的映射字典
        grid2idx: 网格ID到索引的映射字典
        max_seq_len: 最大序列长度，若超过则截断

    返回：
        brand_samples: 样本列表，每个样本是(前缀索引，前缀坐标，前缀POI特征，目标索引)的元组
    """
    data_df = pd.read_csv(data_csv_path)
    brand_samples = []
    for _, row in data_df.iterrows():
        brand = row['brand_name']
        gid_list = parse_list(row['grid_id_list'])
        seq = sort_by_density(gid_list, coords_map)
        if len(seq) < 2: continue  # 至少需要两个点才能形成序列预测样本
        for l in range(1, len(seq)):
            prefix = seq[:l]
            target = seq[l]
            if len(prefix) > max_seq_len:
                prefix = prefix[-max_seq_len:]  # 保留最近的max_seq_len个点
            prefix_idx = [grid2idx[g] for g in prefix]
            prefix_coords = [coords_map[g] for g in prefix]
            prefix_poi = [poi_feat_map[g] for g in prefix]
            target_idx = grid2idx[target]
            brand_samples.append((prefix_idx, prefix_coords, prefix_poi, target_idx))
    return brand_samples

def load_all_data(train_csv, test_csv, grid_csv, val_size=0.2):
    """
    加载并处理所有数据，划分为训练、验证和测试集

    该函数是数据处理的主入口，完成以下步骤：
    1. 加载网格信息（坐标和POI特征）
    2. 构建全局网格ID到索引的映射
    3. 生成训练和测试样本
    4. 从训练样本中再划分出验证集

    数据划分的合理性：
    - 使用固定的随机种子确保实验可复现
    - 验证集从训练集中划分，保证测试集的纯净性
    - 按品牌-网格对划分，避免信息泄露

    参数：
        train_csv: 训练数据CSV文件路径
        test_csv: 测试数据CSV文件路径
        grid_csv: 网格信息CSV文件路径
        val_size: 验证集比例，默认为0.2

    返回：
        train_set: 训练集样本
```

```
        val_set: 验证集样本
        test_samples: 测试集样本
        num_classes: 类别数量（网格总数）
        grid2idx: 网格ID到索引的映射字典
    """
    coords_map, poi_feat_map = load_grid_info(grid_csv)
    # 构造全网格字典
    all_grids = set()
    for csvf in [train_csv, test_csv]:
        df = pd.read_csv(csvf)
        for _, row in df.iterrows():
            all_grids.update(parse_list(row['grid_id_list']))
    grid2idx = {gid: idx for idx, gid in enumerate(sorted(all_grids))}
    num_classes = len(grid2idx)
    train_samples = make_samples(train_csv, coords_map, poi_feat_map, grid2idx)
    test_samples = make_samples(test_csv, coords_map, poi_feat_map, grid2idx)
    # 再在train_samples中拆分出val
    train_idx, val_idx = train_test_split(np.arange(len(train_samples)), test_size=val_size,
      random_state=42)
    train_set = [train_samples[i] for i in train_idx]
    val_set = [train_samples[i] for i in val_idx]
    return train_set, val_set, test_samples, num_classes, grid2idx
```

Listing 4: 数据预处理实现

A.2.5 数据特征增强 (测试数据文件.py)

```
import pandas as pd
import ast
from collections import defaultdict

# 读取 train_data.csv 文件
df_train = pd.read_csv('data/train_data.csv')

# 初始化一个默认字典来存储 grid_id 和 brand_type 的计数
grid_brand_counts = defaultdict(lambda: defaultdict(int))

# 已知的 brand_type 前两个字符集合
brand_types = {'住宿', '摩托', '公司', '餐饮', '体育', '购物', '生活', '汽车', '医疗', '科教
    '}

# 遍历 train_data.csv 的每一行
for _, row in df_train.iterrows():
    brand = row['brand_type'][:2]  # 取 brand_type 的前两个字符
    # 将 grid_id_list 从字符串解析为列表
    grid_ids = ast.literal_eval(row['grid_id_list'])

    # 为每个 grid_id 统计 brand_type 的数量
    for grid_id in grid_ids:
        grid_brand_counts[grid_id][brand] += 1

df_grid = pd.read_csv('data/grid_coordinates.csv')

# 为每个 brand_type 添加一列，初始化为 0
for brand in brand_types:
    df_grid[brand] = 0

# 更新 df_grid 中每个 grid_id 对应的 brand_type 数量
for grid_id in grid_brand_counts:
    # 确保 grid_id 存在于 df_grid 中
    if grid_id in df_grid['grid_id'].values:
        for brand, count in grid_brand_counts[grid_id].items():
            df_grid.loc[df_grid['grid_id'] == grid_id, brand] = count

# 保存更新后的表格到新的 Excel 文件
df_grid.to_csv(
    "data/grid_coordinates-2.csv", encoding='gbk'
)
```

Listing 5: 数据特征增强实现

A.2.6 模型评估模块 (evaluate.py)

```python
"""
模型评估模块
该模块提供了评估模型性能的函数和用于批量处理数据的工具函数。
主要实现了模型在验证集/测试集上的评估指标计算，包括Top-K准确率和MRR。
"""

import torch
import numpy as np

@torch.no_grad()
def evaluate_model(model, dataset, device, k_list=[1, 5, 10]):
    """
    评估模型在给定数据集上的性能

    该函数计算两个关键指标：
    1. Top-K准确率：预测结果的前K个选项中包含正确答案的比例
    2. 平均倒数排名(MRR)：正确答案排名的倒数的平均值

    使用@torch.no_grad()装饰器可以在评估时禁用梯度计算，节省内存并加速评估过程

    参数：
        model: 要评估的模型
        dataset: 评估数据集
        device: 计算设备(CPU/GPU)
        k_list: Top-K准确率中K值的列表，默认为[1, 5, 10]

    返回：
        acc_k: 包含各个K值对应准确率的字典
        mrr: 平均倒数排名(Mean Reciprocal Rank)
    """
    model.eval()  # 设置模型为评估模式
    acc_k = {k: 0 for k in k_list}  # 初始化各K值的准确计数
    mrr_sum = 0  # MRR累加值
    total = 0  # 样本总数
    for seq_ids, seq_coords, seq_poi, targets in batcher(dataset, 64):
        # 将数据移至指定设备
        seq_ids, seq_coords, seq_poi, targets = (
            seq_ids.to(device), seq_coords.to(device), seq_poi.to(device), targets.to(device
    )
        )
        # 获取模型输出（预测分数）
        logits = model(seq_ids, seq_coords, seq_poi)
        # 获取预测分数最高的K个位置的索引
        topk = torch.topk(logits, max(k_list), dim=1).indices.cpu().numpy()
        targets_np = targets.cpu().numpy()

        for i, target in enumerate(targets_np):
            # 找出目标位置在预测的topk结果中的排名
            rank = np.where(topk[i] == target)[0]
            if len(rank) > 0:  # 如果目标在topk中
                rank = rank[0] + 1  # 将索引转换为排名（从1开始）
                mrr_sum += 1.0 / rank  # 计算倒数排名并累加
                for k in k_list:
                    if rank <= k:  # 如果排名在k以内，对应的Top-K准确率计数加1
                        acc_k[k] += 1
            total += 1

    # 计算平均值
    mrr = mrr_sum / total if total else 0
    acc_k = {k: acc_k[k]/total for k in k_list}
    return acc_k, mrr

def batcher(samples, batch_size=64):
    """
    将样本数据转换为批次形式的生成器函数

    该函数实现了以下功能：
    1. 将样本数据按批次划分
    2. 对每个批次内的序列进行填充，使其长度一致
    3. 构建张量格式的数据，用于模型输入

    填充策略是从序列右侧对齐，这种方式在处理序列数据时更为合理，
    因为序列的最新部分（右侧）通常包含更重要的信息。
```

```
    参数:
        samples: 样本列表，每个样本是(前缀索引, 前缀坐标, 前缀POI特征, 目标索引)的元组
        batch_size: 批次大小，默认为64

    生成:
        每次生成一个批次的数据，包含:
        - seq_ids: 序列ID张量，形状为(batch_size, max_seq_len)
        - seq_coords: 序列坐标张量，形状为(batch_size, max_seq_len, 2)
        - seq_poi: 序列POI特征张量，形状为(batch_size, max_seq_len, 10)
        - targets: 目标索引张量，形状为(batch_size,)
    """
    for i in range(0, len(samples), batch_size):
        batch = samples[i:i+batch_size]
        # 找出当前批次中最长的序列长度
        maxlen = max(len(x[0]) for x in batch)
        # 初始化批次数据数组
        seq_ids = np.zeros((len(batch), maxlen), dtype=np.int64)
        seq_coords = np.zeros((len(batch), maxlen, 2), dtype=np.float32)
        seq_poi = np.zeros((len(batch), maxlen, 10), dtype=np.float32)

        for j, (pidx, pcoord, ppoi, _) in enumerate(batch):
            L = len(pidx)
            # 右对齐填充：将数据放在数组的右侧，左侧用0填充
            seq_ids[j, -L:] = pidx
            seq_coords[j, -L:, :] = pcoord
            seq_poi[j, -L:, :] = ppoi

        # 提取目标值
        targets = np.array([x[3] for x in batch], dtype=np.int64)

        # 将NumPy数组转换为PyTorch张量并返回
        yield (
            torch.from_numpy(seq_ids),
            torch.from_numpy(seq_coords),
            torch.from_numpy(seq_poi),
            torch.from_numpy(targets)
        )
```

Listing 6: 模型评估实现

A.3 数据样本展示

A.3.1 训练数据格式

以下直接展示训练数据文件的前 10 行内容：

```
brand_name,brand_type,longitude_list,latitude_list,grid_id_list
四季啤酒屋,体育休闲服务;娱乐场所;酒吧,"[116.928274525344, 117.043320008084, 116.9422963244636, 117.0778221078006]","[36.69774403270743, 36.69430636057923, 36.69940378336072, 36.65480559942927]","[71, 77, 72, 25]"
携程旅游,生活服务;生活服务场所;生活服务场所,"[117.0099652551294, 117.0083278100292, 116.9029874347601, 117.1258233788835, 117.1318264911793, 117.130692205233, 117.02760045368049, 116.9643505011395, 117.0700358356303, 116.9762270603227, 117.0415532841417]","[36.61515271727929, 36.64534980906663, 36.67805564190155, 36.65345336291342, 36.66021335891504, 36.676643613408, 36.65479701387262, 36.65681312305596, 36.64246444489299, 36.61303461884545, 36.68037991342308]","[4, 21, 51, 28, 45, 64, 22, 19, 25, 3, 59]"
幸福时光KTV,体育休闲服务;娱乐场所;KTV,"[116.9786056408031, 116.9167728180969, 117.0098096139525, 116.9590453043534, 117.04950909468291]","[36.66161180954602, 36.65290255903322, 36.6710547488655, 36.67043566438539, 36.66060909605174]","[37, 16, 38, 36, 41]"
小电,生活服务;共享设备;充电宝,"[116.946966962845, 117.0165404020267, 117.0442775933112, 117.0251578749105, 117.0251360192659, 117.0703663502994, 116.9679460276465, 117.1241885471427, 117.0454510621802, 116.9908858098853, 117.0164047908445, 117.0595249194539, 117.1492552466137, 117.0387305279752, 117.0026990920988, 116.9833750721236, 117.0259239931365, 117.1239399242847, 116.9688828178708, 117.0047431014325, 116.9807610358491, 117.0817309886335]","[36.70479258447448, 36.66309812601193, 36.64531065644391, 36.663965842217707, 36.66344022296101, 36.67629346501127, 36.72221597591807, 36.65502138143246, 36.67928044322549, 36.66917117386832, 36.65574600192463, 36.68491232846421, 36.70537523707548, 36.65513173060003, 36.61779391991816, 36.68115049015188, 36.66329622415145, 36.69306744935307, 36.6628680099412, 36.61928189507721, 36.66546105430051, 36.70016712172751]","[72, 39, 23, 39, 39, 61, 89, 28, 59, 37, 22, 60, 83, 23, 4, 56, 39, 64, 36, 4, 37, 79]"
康明眼镜,购物服务;专卖店;眼镜店,"[117.1474925587893, 116.9899725033711, 117.0496668536213, 117.0009462913962, 116.9162896279549, 117.1410070518251]","[36.72060927039779, 36.69860146269658, 36.67619517995269, 36.67480000224907, 36.65303092587671, 36.73127828863626]","[98, 74, 60, 38, 16, 105]"
中财管道,购物服务;家居建材市场;建材五金市场,"[117.0334533368044, 116.9703254004627, 116.9880174777508, 116.9558548711589, 116.992987253108, 116.9364190460066, 117.143078283607, 116.9750794367394, 117.0050266022032, 117.0778553647915, 117.031448155798, 117.0102340663978, 116.9578983721978, 117.0242489307601, 117.039477183328, 117.0215823229305, 116.9612181682283, 116.9550518132942, 116.9431431740838, 116.9606835793541, 117.0478167565158]","[36.68642073855862, 36.7131882069229, 36.66762686110791, 36.71761468549576, 36.68152706900885, 36.68485510759297, 36.7221657010541, 36.716158675651, 36.70174365839906, 36.71288009941338, 36.70789597294158, 36.69540520391379, 36.69043405380769, 36.69456713613525, 36.68870724027503, 36.71320839471499, 36.70611753046003, 36.66261143837973, 36.71354397295313, 36.69362291074891, 36.706768313868941","[59, 89, 37, 88, 56, 53, 98, 89, 75, 94, 77, 75, 55, 76, 59, 91, 73, 35, 88, 73, 78]"
美利达自行车,购物服务;专卖店;自行车专卖店,"[116.9524783955336, 116.9154449123776, 117.0606450771626, 117.2299536114407, 116.9908572113357, 117.0750930872932, 117.2280146835699, 117.1371797256954, 116.956813342363, 117.038657173431]","[36.67667890606948, 36.65417998169556, 36.73888578931914, 36.68444825883368, 36.65774427341513, 36.68036777295877, 36.72384491010683, 36.65061320002995, 36.64512170736571, 36.67791786120331]","[54, 16, 102, 68, 37, 61, 101, 29, 18, 59]"
酒快到,购物服务;专卖店;烟酒专卖店,"[117.1316969616346, 117.3016293700698, 117.0343191810293, 116.9759775287286, 117.10136898348041","[36.65471352253594, 36.67943315359312, 36.67821879365496, 36.64535937215268, 36.685721602042]","[28, 70, 59, 20, 63]"
望京烧烤,餐饮服务;中餐厅;特色/地方风味餐厅,"[116.9464233558445, 117.0335236445724, 117.0455703904242, 117.218120082778, 116.9148782739088, 117.17556094168111","[36.61671256380603, 36.68151265323061, 36.71683623281009, 36.70442907328825, 36.67787789073074, 36.67421657284811]","[1, 59, 92, 85, 52, 48]"
```

Listing 7: 训练数据样本 (train_data.csv)

A.3.2 网格坐标映射

以下直接展示网格坐标映射文件的前 10 行内容：

```
,grid_id,grid_lon_min,grid_lat_min,grid_lon_max,grid_lat_max,e□□,¹²□□,²□□□,¿5□,H□□,¹«_,□□»□,□□³µ,□□□□,□□□
0,1,116.8315,36.71129734099892,116.84946622349985,36.72926356449874,11,47,63,4,0,1,19,3,1,7
1,2,116.8315,36.72926356449874,116.84946622349985,36.74722978799856,5,15,12,0,1,0,8,3,0,0
2,3,116.84946622349985,36.6035,116.86743244699966,36.621466223499816,1,25,25,2,0,0,13,5,1,0
3,4,116.84946622349985,36.621466223499816,116.86743244699966,36.63943244699964,15,76,71,7,3,0,36,3,3,5
4,5,116.84946622349985,36.63943244699964,116.86743244699966,36.657398670499454,6,11,11,3,0,0,10,2,3,0
5,6,116.90336489399928,36.6035,116.92133111749912,36.621466223499816,10,26,34,2,0,0,10,0,2,0
6,7,116.90336489399928,36.63943244699964,116.92133111749912,36.657398670499454,3,19,16,2,0,0,6,3,0,0
7,8,116.90336489399928,36.65739867049946,116.92133111749912,36.67536489399928,16,43,28,1,0,0,14,3,1,1
8,9,116.90336489399928,36.67536489399928,116.92133111749912,36.6933311174991,16,43,34,4,1,0,26,1,2,0
```

Listing 8: 网格坐标映射数据 (grid_coordinates-2.csv)

A.3.3 测试数据格式

以下直接展示测试数据文件的前 10 行内容：

```
brand_name,brand_type,longitude_list,latitude_list,grid_id_list
婴贝儿,购物服务;专卖店;儿童用品店,"[117.0996712293962, 117.1379846421835, 116.8932622025578, 117.1499975247358, 117.024877028235,
        116.9530623680675, 116.9175268352235, 116.9058644662969, 116.9726887822101, 116.9864232644485, 117.1984933964415, 117.0789651487876,
        117.0068100044073, 117.0645120634065, 117.1805712919167, 117.1291492015525, 117.1229203417397, 116.9822128030304, 116.9805567733302,
        117.0470645130879, 116.9699514634457, 116.9668837825261, 116.932661722804, 116.8956431338762, 117.0294847274234, 117.0925924533216,
        117.1404898036926, 117.0676784552798, 116.9632200208925, 117.2130256724302, 117.114841357525, 116.9589876072608, 116.9112198636204,
        117.0535553747456, 116.976233074227, 117.1481918016937, 117.0755583820197, 117.0493036526866, 117.0647861401144, 116.9786000817559,
        117.0318934269682, 116.9474375020692, 117.3023863434853, 116.9989403856787, 116.9286838764815, 117.1031537792245, 117.212533931024,
        116.9490482542897, 116.9335158542075, 117.0912068977715, 116.8952493698547, 116.8937014397158, 117.0012938718576, 117.1348022717404,
        116.9934381463388, 116.9248374912076, 117.0338667428631, 117.1245142410434, 116.959728614966, 117.0789611631621, 116.9819612318406,
        116.9761256282631, 117.0663021496076, 116.9598322117709, 116.9543295013623, 117.0760248289039, 116.9373115120875, 116.972206984709,
        116.9759941866322, 117.139236644202, 116.9820209791802, 117.122275277546, 117.006788341595]",”[36.68518634034514, 36.65259926084561,
        36.64789102488911, 36.65282137701404, 36.70738025265271, 36.62171064249292, 36.67760566255552, 36.68758359058992, 36.64926352641689,
        36.69574507118392, 36.68805065032451, 36.64994782878281, 36.6923003368949, 36.69080087192778, 36.71362065379788, 36.68683109709343,
        36.68971435585137, 36.68355946553892, 36.65804020687233, 36.71975535800927, 36.690235964207999, 36.65427378356909, 36.66076342911936,
        36.67534239720015, 36.69639228882006, 36.72149586387541, 36.65705057132519, 36.7013634286597, 36.682964976529779, 36.699040419888929,
        36.64470063327149, 36.65144342757935, 36.64945811374917, 36.699346626688804, 36.612680804298876, 36.71987996978413, 36.68042138556957,
        36.69581050100167, 36.73720243585072, 36.67586502415248, 36.70002192573417, 36.7040490371283, 36.680286280858879, 36.65370106096641,
        36.63952835597043, 36.69587093347233, 36.6752718706838, 36.67571039371824, 36.65750701651433, 36.68988022221125, 36.68434475948285,
        36.66258215735837, 36.67399944541669, 36.73042609089215, 36.62910253209033, 36.70510200590964, 36.66111545846772, 36.68652928405672,
        36.67158120624317, 36.6498678793942, 36.63545258838949, 36.71097399359922, 36.66674974427514, 36.671700865835557, 36.63431322611724,
        36.68791932711586, 36.69440347151026, 36.67442049817143, 36.71091641427032, 36.72927374196909, 36.6354204286713, 36.68513449089427,
        36.61908088107396]",”[62, 29, 15, 29, 76, 8, 52, 52, 19, 74, 66, 25, 57, 60, 99, 64, 64, 56, 37, 92, 55, 19, 34, 32, 77, 95, 29, 79, 55,
        85, 27, 19, 16, 78, 3, 98, 61, 78, 102, 56, 77, 72, 70, 21, 17, 81, 50, 54, 34, 62, 51, 32, 38, 104, 11, 71, 40, 64, 36, 25, 10, 74, 42,
        36, 8, 61, 71, 36, 74, 105, 10, 64, 4]”
易居房友,生活服务;中介机构;中介机构,”[117.0455418760648, 116.9509171744841, 116.9431907340005, 117.1178271095851, 117.0605752418835,
        116.9685808273521, 116.9747873447345, 117.1068710799779, 117.2360849347613, 116.9480072307894, 116.8889581106447, 116.9503722410296,
        116.9435205358656, 117.0798758522983, 117.1183239314952, 116.9211595391622, 116.9208517884814, 117.1028050006402]”,”[36.69592275616423,
        36.62761021751274, 36.64295591436264, 36.63778862425124, 36.704137595514209, 36.68285644843733, 36.64682295253007, 36.64515329401156,
        36.6844896505136, 36.62532311705273, 36.62305080197009, 36.60744311333485, 36.64456028149461, 36.65115957318063, 36.64249232991678,
        36.67326156531449, 36.65803927039991, 36.6938970305318]”,”[77, 8, 18, 13, 78, 55, 19, 27, 68, 8, 6, 1, 18, 25, 27, 33, 33, 81]”
ONLY,购物服务;服装鞋帽皮具店;服装鞋帽皮具店,”[117.0329723390454, 117.126770055059, 117.0069781402267, 116.9158825102825, 116.9680673165032,
        117.1030590123878, 117.0661138170195, 116.9724189265898, 116.9160061067833, 117.0548685526778, 117.0091332642082,
        116.9699238776644]”,”[36.65771270947515, 36.71707256556891, 36.684708631290 98, 36.71590579384737, 36.6609164721 51, 36.66783446422561,
        36.6729341401392, 36.64897846977063, 36.65301619076054, 36.70229353395882, 36.64640278878978, 36.66027968043058]”,”[40, 97, 57, 86, 36, 44,
        42, 19, 16, 78, 21, 36]”
马博士婴幼儿游泳馆,生活服务;婴儿服务场所;婴儿游泳馆,”[117.0329544373787, 117.152660325078, 117.1310958460403, 117.0015420162948, 116.9766451102195,
        116.976173324498, 117.0969491625688, 117.1273725892052, 117.0574113167814, 116.903516194149, 117.2276270738628,
        117.2183742367875]”,”[36.65741593403924, 36.6507404109261, 36.67535198563168, 36.66312840564837, 36.63285286296802, 36.6126849763032,
        36.69106492160663, 36.71812366636688, 36.68266191994277, 36.67560000226136, 36.69304155157505, 36.66937440180349]”,”[40, 29, 45, 38, 10, 3,
        62, 97, 60, 52, 68, 50]”
东润大药房,医疗保健服务;医药保健销售店;药房,”[116.9189725373848, 117.1798187085277, 117.3004779145081, 117.056191930025, 117.178542302356,
        117.0920242146625, 117.0450441177042, 116.9223930707534, 117.1007288455053, 116.9511416845087, 117.0634469596785, 117.0836888352784,
        116.895060399059, 116.9420764131831, 117.0378625217185, 117.1446505331589]”,”[36.64703294263889, 36.64582666385298, 36.69132468468406,
        36.69534881889702, 36.69994979724778, 36.70030056941857, 36.71594188138783, 36.67330211662661, 36.72198908101953, 36.66159833949235,
        36.73747494277003, 36.68309610665886, 36.64780522367574, 36.69557866189321, 36.6809817531441, 36.7241810365305]”,”[16, 31, 70, 78, 84, 80,
        92, 34, 95, 35, 102, 62, 15, 72, 59, 98]”
巴拉巴拉,购物服务;专卖店;儿童用品店,”[116.9184409683297, 117.0487200623952, 117.2265239832789, 116.9699318493467, 116.9162878357841,
        116.9685862509238, 117.1232751782797, 116.9765365659153]”,”[36.7166918557168, 36.71575676708599, 36.69076631970337, 36.66033462447786,
        36.71584047134091, 36.66072115060988, 36.68534053512016, 36.63285917669518]”,”[86, 93, 67, 36, 86, 36, 64, 10]”
绿能电动车,购物服务;专卖店;专营店,”[116.9553366474382, 117.0542860517775, 117.1375817813 54, 117.1150193252241, 117.0674438409209,
        117.1809399707232, 116.9914004966939, 117.1079942801645, 117.0116639683536, 117.0997243007297]”,”[36.67300333474505, 36.71129970238661,
        36.73335742719549, 36.68928933212071, 36.71061673340619, 36.72203683502129, 36.66977323153279, 36.6977984063399, 36.62508696696515,
        36.69316031721775]”,”[35, 93, 105, 63, 79, 99, 37, 81, 12, 62]”
杰克琼斯,购物服务;服装鞋帽皮具店;品牌服装店,”[117.0939859999953, 117.1030660151984, 117.1919618382704, 117.122413351018, 117.0581444822647,
        117.2263799631202, 117.127057273006, 116.9530586022749, 117.0165067544813, 117.0168819348173, 116.9161997937927]”,”[36.65882156309365,
        36.66781148160168, 36.6712140470736, 36.68558436801666, 36.68288265791443, 36.69050839174971, 36.717877414642, 36.619690823 07267,
        36.66120438214303, 36.663682796 61755, 36.65290008657293]”,”[43, 44, 49, 64, 60, 67, 97, 1, 39, 39, 16]”
立聪堂助听器,医疗保健服务;医药保健销售店;医疗保健用品,”[117.0743981567748, 116.9504006346039, 117.0300145426286, 117.0138111133497,
        116.9950914008877, 117.0004169856987]”,”[36.68034533367015, 36.64870287347598, 36.6543501093 5824, 36.65936424237968, 36.62487595645675,
        36.64456707602362]”,”[61, 18, 23, 39, 11, 21]”
```

Listing 9: 测试数据样本 (test_data.csv)

A.4 完整源代码文件结构

项目包含以下主要文件：

- `src/model.py` - 神经网络模型定义
- `src/main.py` - 主程序入口
- `src/train.py` - 训练逻辑实现
- `src/evaluate.py` - 模型评估
- `src/data_preprocessing.py` - 数据预处理

- `data/train_data.csv` - 训练数据
- `data/test_data.csv` - 测试数据
- `data/grid_coordinates-2.csv` - 网格坐标映射

详细的代码实现请参考 GitHub 仓库：`https://github.com/szw0407/DL-project-2025`