南　　开　　大　　学

机器学习课程设计报告

基于半监督学习的交互式相似细胞预测

付忠缘　2213015

罗瑞　　2210529

刘存　　2213050

2025 年 1 月 19 日

实验任务概述

- **基础要求：**阅读并复现 PairPot 论文中基于随机游走的 Lasso-View 的方法，预测并可视化展示用户感兴趣的细胞群；使用论文中的方法对结果进行评价，屏蔽 90% 正常标签并模拟 10% 错误标签（错误率范围 0-95%）。
- **进阶要求：**使用其他算法实现，如决策树、支持向量机和神经网络等；也可以对论文中的标签传播算法进行优化。

目录

一、 实验背景

(一) 实验任务：单细胞测序数据分析

单细胞测序数据分析是在单个细胞水平上，对基因组、转录组等进行高通量测序分析，旨在精细解释一个样本内细胞间的异质性。简单来说，单细胞测序数据就是一个行是细胞，列是基因的高维稀疏矩阵。针对单细胞测序数据的工作流具体包括：质量控制、数据标准化、数据降维、细胞聚类、细胞类型注释和可视化。

然而，最关键的细胞聚类步骤由于分辨率的不同导致聚类结果无法反应真实的细胞类型，需要手动交互式的修正。而手动圈选必然会导致相似细胞遗漏或误选错误细胞，因此需要一种交互式的相似细胞预测方法弥补这一缺陷。

(二) 实验数据集介绍

单细胞数据集 dataset.h5ad 中存储了图中的所有细胞节点，每个细胞节点的维度数据主要是该细胞的注释（存储在 annotation 中）和它的基因表达情况（存储在稀疏矩阵中）。我们使用 h5py 方法进行 h5ad 格式的数据文件读取。我们还可以引入标准分析流来帮助数据初始化工作，其主要用于处理和预处理单细胞测序数据，通过一系列步骤来规范化和降维数据，为后续分析做准备。

本实验中共有 10 个测试数据。每个 txt 中的第一组数据是用户选定的细胞的索引，即输入；第二组数据是经过 Lasso-View 预测出的用户感兴趣的细胞集合的索引，即输出。

二、 模型算法简介与评价方法介绍

(一) 标签传播算法：LPA

Label Propagation Algorithm，也称作标签传播算法（LPA），是一种基于图的半监督学习的算法，常用于节点分类和图数据的聚类分析等。该算法主要通过在各个节点之间传播标签并逐步确定，最终将图中的节点分为若干个簇或类别。本次作业中，我们在 label_propagation.py 种实现 LPA 标签传播算法。

(二) 基于 LPA 算法的启发式细胞注释方法；Lasso-View

Lasso-View 是 PairPot 数据库中贡献的一种基于 LPA 算法来猜测用户感兴趣节点群的方法。在 PairPot 对单细胞数据的分析中，每一个细胞作为图中的一个节点，当用户手动圈选一部分细胞后，Lasso-View 能在毫秒级响应时间内识别该用户感兴趣的细胞亚群。

具体而言，在数据整合阶段，Pairpot 为每个数据集生成概率转移矩阵。在线分析阶段，用户圈选细胞并生成新的细胞类型 U。运用标签传播算法迭代生成所有细胞属于细胞类型 U 的概率。该过程采用 C++ 优化，可实现毫秒级响应。最后运用 K 近邻调整生成结果。

同样，这部分算法将在基础要求实现部分展示。

(三) 模型评价方法

Lasso-View 采用的评价方式为：遮蔽掉 90% 的初始注释，在 10% 剩余的标签中，添加错误标签（0—95%）；推断出所有细胞类型注释后，计算与初始注释的 ARI 值。ARI（Adjusted Rand Index，调整兰德指数）是一种用于衡量聚类结果与真实分类之间的相似度的评价方法。它

通过比较聚类结果与真实分类之间的成对样本相似性来计算得分，匹配范围 [-1,1]。sklearn 中的 adjustedRandScore 函数可以帮助计算。本次实验我们需要计算两种 ARI:

1. 原始 ARI_o（直接使用 LPA 的预测标签与真实标签的相似度）
2. 修正 ARI_r（修正后的标签传播结果与真实标签的相似度）

三、 基础要求实现与评价结果展示

(一) 复现 PairPot 论文中基于随机游走的 Lasso-View 的方法, 预测并可视化展示用户感兴趣的细胞群

1. 数据集预处理（其中使用标准工作流增强模型预测性能）

使用标准分析流进行数据集读取与分析

```python
# 读取 .h5ad 数据
adata = sc.read_h5ad('./data/dataset1.h5ad')

# 查看数据的基本信息
print(adata)  # 行是细胞，列是基因

# 标准分析流
# 1. 数据标准化
sc.pp.normalize_total(adata, inplace=True)  # 每个细胞的总表达量标准化为1
sc.pp.log1p(adata)  # 对基因表达值进行对数转化

# 2. 筛选高变基因
sc.pp.highly_variable_genes(adata, flavor='seurat', inplace=True)  # 筛选高变基因

# 3. PCA降维
sc.pp.pca(adata, n_comps=50, use_highly_variable=True, svd_solver='arpack')  # 使用高变基因进行PCA降维

# 4. 计算邻近关系
sc.pp.neighbors(adata)  # 基于PCA结果计算细胞之间的邻近关系

# 5. UMAP降维
sc.tl.umap(adata)  # 使用UMAP降维进行可视化

# 6. 可视化结果
sc.pl.umap(adata, color=['highly_variable'])  # 根据需要替换 'highly_variable' 为感兴趣的特征
```

现在我们获得了对数据 adata 的一系列重要信息:

adata.obs: 细胞的注释信息，也就是细胞的所属类型标签（名字）。

adata.X: 一个稀疏矩阵，记录着细胞之间的关联权重，表示一个图，细胞与细胞之间有边，边的值是细胞之间的关系权重。

adata.X_pca: 原始的细胞特征矩阵，在本实验中是 3000 行的矩阵，每一行代表一个细胞，有若干特征列，表示细胞若干基因的表达情况。

adata.X_umap: 进行主成分分析降维后的细胞特征矩阵，每个细胞有两个维度，依据这个矩阵信息可以画出根据此主成分分析降维方法后的细胞分类情况。

我们展示数据预处理后的细胞特征降维聚类图 Umap：

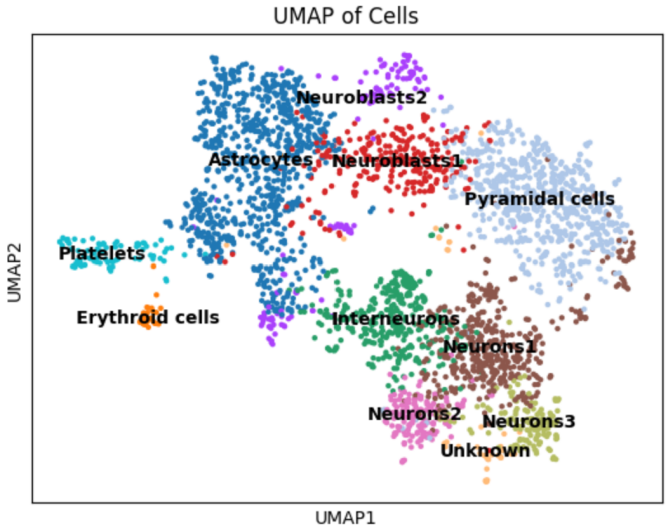

图 1: 细胞特征聚类图 Umap

我们还进行 t-SNE 数据集可视化，线性降维 t-SNE 通过计算数据点之间的相似度，能够尽量保持高维空间中相似的数据点在低维空间中的相对位置。

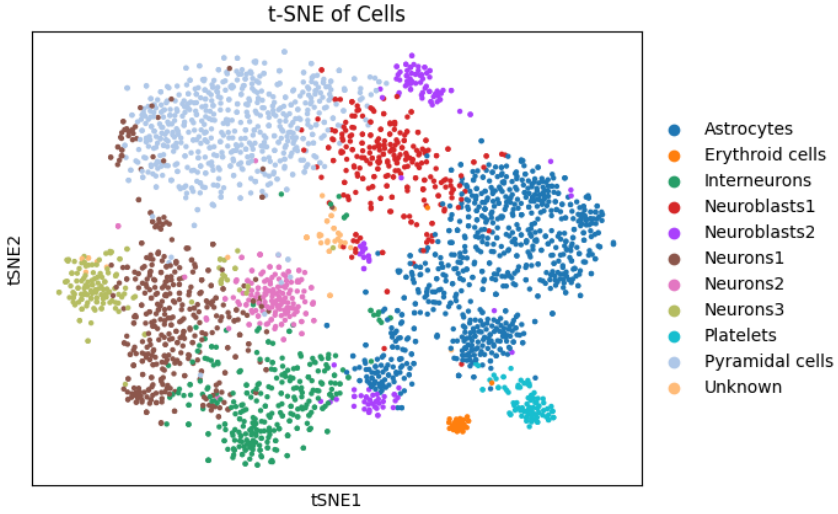

图 2: 细胞特征 t-SNE 图

当然，我们初始化后的数据集还可以进行细胞基因（原始特征）状况的可视化，比如展示一些高变基因，这些高变基因的特征信息将会在降维处理中更多地被保留，是影响我们进行细胞预测的关键性数据。

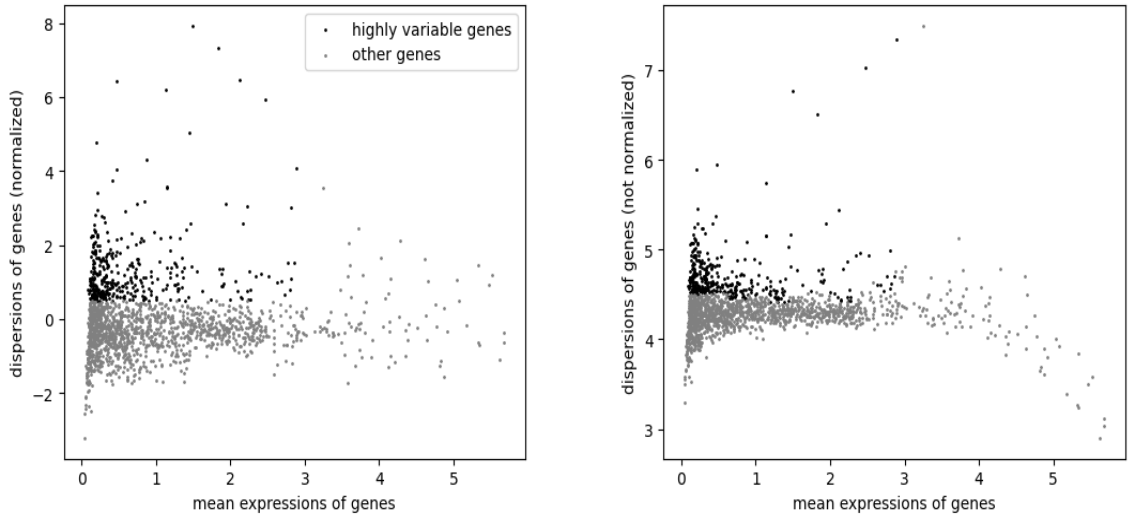

图 3: 细胞高变基因可视化

要完成预测用户感兴趣的细胞的任务，我们依次读取 txt 文件中用户挑选的细胞索引系列，对每个序列都调用 lassoview 方法进行预测。

读取 txt 文件中用户感兴趣的细胞并使用 lassoview 方法进行预测

```
with open(f'./data/test/{i}.txt', 'r') as f:
    # 读取第一行并解析为列表
    first_line = f.readline().strip()  # 去除首尾空格或换行符
    lassoed_index = eval(first_line)  # 使用 eval 将字符串转换为列表

refined_index = pt.lassoView(lassoed_index, adata, do_correct=True, function="anndata")
print(refined_index)
```

2. 基于 LPA 标签传播算法的 Lasso-view 预测方法实现

Lasso-view 方法实现的具体函数位于 lassoview.py 文件中，其中输入数据是单细胞 RNA-seq 数据的邻接矩阵，通常以稀疏矩阵的形式存储，表示细胞间的相似性。我们的目标是通过半监督学习对部分细胞进行标签传播，推断未标注细胞的类别，并可根据需要进行结果修正。

lassoview 实现

```
#将标签映射为数值型字典（val），确保标签类别可以被传递给标签传播算法。
val = {}
for i in np.unique(mat):
    val[i] = len(val)
val[len(val)] = len(val)

#构建标签传播矩阵
X = LPA.matCoo(mat.shape[0], mat.shape[0])
for i in range(len(data)):
    X.append(rows[i], cols[i], data[i])
#构建一个 matCoo 对象，表示稀疏邻接矩阵，用于后续的标签传播。
```

```
#rows, cols, 和 data 分别表示稀疏矩阵的行、列和对应的权值。

#初始化标签
random_list = random.sample(range(mat.shape[0]), int(mat.shape[0] * 0.1))
select_list = np.zeros(mat.shape[0])
select_list[random_list] = 1
select_list[selected] = 1
```

我们对细胞的稀疏矩阵进行处理并进行数据的初始化与读取细胞的标签信息，这里的关键是要进行标签映射并构建标签传播矩阵，即一个 matCoo 对象，我们随机选择部分细胞（10%）作为初始标签。将用户选取细胞也标记为已知类别，在本实验中就是被标注为第 12 类已知类别，我们将从数据集中随机选取的细胞类别与用户挑选的类别一起作为细胞标签类别来执行标签传播算法（LPA）。

标签传播算法 LPA 的核心做法就是，先构建相似性矩阵并初始化标签矩阵，再通过迭代过程传播已知标签信息，更新邻近样本的标签分布，最后在收敛后得以输出预测结果。

1. 初始化

LPA 初始化

```
Y = deepcopy(y_label)
W = matCoo(n_samples, n_samples)
for elem in X.elem:
    ...
    similarity = math.exp(-alpha * dist)
    W.append(row, col, similarity)
```

标签初始化：Y 初始化为 y_label，存储每个样本对不同类别的置信度。

相似度矩阵 W：基于邻接矩阵 X，通过高斯核函数计算相似度权重，使用高斯核函数的一个优点是可以对相似样本赋予更高的权重，从而在我们的预测过程中能够强调细胞之间的相似性。

2. 标签传播的迭代过程

LPA 迭代进行标签传播

```
for iter_num in range(max_iter):
    Y_old.assign(Y_new)
    matMultiply_x1_x2_res(W, Y, Y_new)  # 标签传播
    ...
    diff = Y_old.findDiff(Y_new) / n_samples
    if diff < 1e-5:
        break
```

标签传播：通过矩阵乘法 W × Y，将已知标签的信息传播到邻近的细胞。

先验更新：如果某细胞有先验标签，则强制其标签保持不变。

收敛判断：通过计算两次迭代间的差异 diff，当差异小于阈值 1e-5 时停止迭代。

3. 标签修正

LPA 算法的标签修正

```
for i in range(n_samples):
    max_index = np.argmax(Y.v[i, :])
    y_pred.v[i, 0] = max_index
```

预测结果：每个细胞选择置信度最高的类别作为预测标签。
修正过程：调用 rectify 函数进一步修正标签分配。

LPA 算法的标签修正 rectify 函数

```
def rectify(x: matCoo, y_label: mat, y_ori: mat, y_new: mat):
    # 初始化修正矩阵为原始预测矩阵
    y_new.assign(y_ori)

    # 对邻接矩阵排序，方便查找邻居
    x.sortElems()

    # 遍历每个样本进行修正
    for i in range(y_label.n):
        # 如果当前样本有先验标签，则需要修正
        if y_label.v[i, 0] != -1:
            # 找到当前样本的邻居范围
            p1 = bisect.bisect_left([elem.row for elem in x.elem], i)
            p2 = bisect.bisect_left([elem.row for elem in x.elem], i + 1)

            # 统计邻居的标签分布
            p = {}
            p[y_ori.v[i, 0]] = 1  # 初始化当前样本标签计数
            maxx = 1
            maxxj = y_ori.v[i, 0]

            # 遍历邻居
            for j in range(p1, p2):
                val = y_ori.v[x.elem[j].col][0]  # 获取邻居的标签
                if val in p:
                    p[val] += 1
                else:
                    p[val] = 1
                # 更新出现次数最多的标签
                if p[val] > maxx:
                    maxx = p[val]
                    maxxj = val

            # 将标签最多的类别赋值为修正后的标签
            y_new.v[i, 0] = maxxj
```

修正的方法是先检查邻居的标签分布，每个样本根据其邻居的标签分布进行修正，保证相似样本的标签趋于一致，再优先保留先验标签，如果当前样本有先验标签（y_label.v[i, 0] 不为 -1），则邻居中的标签分布会以此为基础进行统计。最后选择标签最多的类别，修正过程的核心是“多数表决”：一个样本的标签由其邻居中标签数量最多的类别决定。

3. 细胞群预测可视化

下面展示根据用户挑选的细胞种类来进行细胞群预测的结果 UMAP 图（由于篇幅原因，这里只展示第一个 txt 文件对应的预测可视化，完整的可视化结果在作业提交文件部分 1 中）：

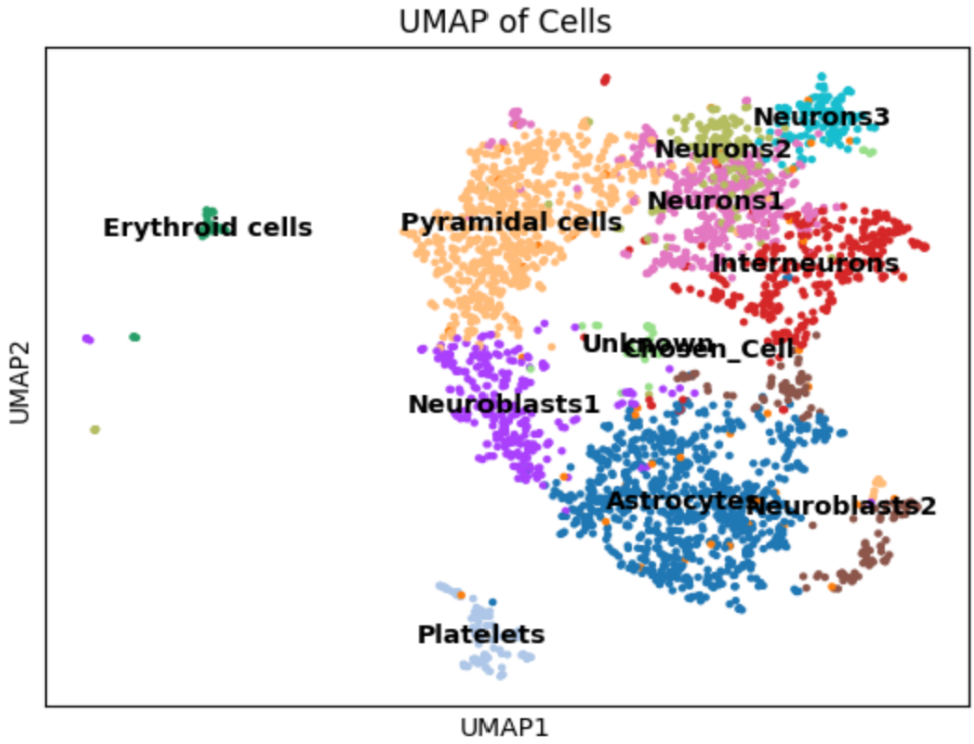

图 4: 细胞群预测结果可视化

根据上图可以看到，我们的算法已经根据用户一开始挑选的一部分感兴趣的细胞对所有同类型的细胞进行了预测，并显示在图中（标签：“chosen_cell”）

(二) 使用论文中的方法对结果进行评价，屏蔽 90% 的正常标签并模拟 10% 错误标签（错误率范围 0-95%）

我们来评估 LPA（Label Propagation Algorithm）的鲁棒性，主要是通过调整标签传播的参数 k1（给出的 10%的细胞的标签中的正确率）来观察算法在不同参数下的性能变化。下面对关键步骤进行简要解释：

1. 标签准备

标签映射

```
val = {}
for i in mat:
    if i not in val:
        val[i] = len(val)
```

标签映射：将原始标签（如 annotation）映射到一个整数字典，方便处理，这跟上一部分所述 lasso-view 算法复现的方法是一致的，包括下面的初始化传播矩阵等步骤。

2. 通过逐步调整参数 k_1 来实现计算不同 10%标签错误率下的细胞群预测性能评估

调整给予标签部分的错错误率

```
while True:
    for _ in range(100):  # 循环100次进行评估
        ...
```

```
        random_list = random.sample(range(mat.shape[0]), int(mat.shape[0] *
            0.1))
        select_list = np.zeros(mat.shape[0])  # 选择的节点标记
```

随机标签选择：随机选择 10% 的样本作为初始有标签样本，模拟部分标注的情景。

初始化循环：对标签传播的参数 k_1 （10% 标签中的正确率吧）从 1.0 开始逐步减少到 0.05（对应要求中的 0 95% 错误率），来评估不同错误率对算法性能的影响，从而分析模型的鲁棒性。

3. 执行标签传播

构建稀疏邻接矩阵 X

```
X = LPA.matCoo(mat.shape[0], mat.shape[0])
for i in range(len(data)):
    X.append(rows[i], cols[i], data[i])
```

这里构建出稀疏邻接矩阵 X，从而用于标签传播的相似性计算。我们后面才呢使用标签传播算法 LPA。

对标签正确率进行设置并执行标签传播算法

```
LPA.dataProcess(y_label, y_new, k1, (1 - k1), 0)
LPA.labelPropagation(X, y_new, y_pred, y_res, 0.5, 1000)
```

我们在 dataprocess 函数中来设置一定错误率（0 95%）的 10% 标签和已屏蔽的 90% 的标签。这里 y_label 就是初始化的真实的标签。y_new 就是 10% 的含有一定错误率的标签，y_pred 就是通过 LPA 标签传播算法得到的全图预测结果，并且通过这个结果进一步对输入的 10% 的含有一定错误的标签进行修正，y_res 就是修正后的全图预测结果。

4. 原始 ari 性能评估 (ARI_o 计算) 和修正后的 ARI 性能评估 (ARI_r 计算)

ARI 是用来评估聚类效果的指标，衡量预测标签与真实标签的一致性。ari_o 是原始传播结果的评估。我们还基于修正后的传播标签 y_res 计算 ARI_r，观察修正对结果的改进效果。

ARI_o 计算

```
res_arr = np.zeros(mat.shape[0])
for i in range(mat.shape[0]):
    res_arr[i] = y_pred.getval(i, 0)
ari_o = adjusted_rand_score(out_arr, res_arr)
```

ARI_r 计算

```
for i in range(mat.shape[0]):
    res_arr[i] = y_res.getval(i, 0)
ari_r = adjusted_rand_score(out_arr, res_arr)
```

5. 实验结果记录

打印模型性能

```
item = [round((1 - k1), 2), ari_o, ari_r, round(execution_time, 5)]
df.loc[len(df)] = item
```

我们衡量的 LPA 算法性能如下表所示：

已知标签错误率	未修正 ARI	已修正 ARI
0.0	0.8323058285502125	0.8243622972012948
0.05	0.8253419306944084	0.8257975885947261
0.1	0.8107381843967011	0.8173063593403628
0.15	0.8098550753324645	0.8218867421988373
0.2	0.7936431454339102	0.815512872115508
0.25	0.7858226567737391	0.8196046306461262
0.3	0.7771139500591867	0.8083307416270427
0.35	0.762246736540323	0.8041274524048376
0.4	0.7607617099652959	0.8154106883756561
0.45	0.742862750485476	0.7932149200886788
0.5	0.6953815955777545	0.7551458768983242
0.55	0.6739936782393342	0.7466284588178486
0.6	0.6343176990565895	0.690278150306603
0.65	0.5534145617253509	0.625918116719425
0.7	0.49917769466248	0.5640594915857289
0.75	0.42442015857739335	0.4893755049068251
0.8	0.3292479071039574	0.3813118885400974
0.85	0.2019784648737845	0.23388470969058758
0.9	0.12054273832074222	0.1459316014106804
0.95	0.11068551899167184	0.12875393331598772

表 1: LPA 算法性能评价结果

结果如图所示:

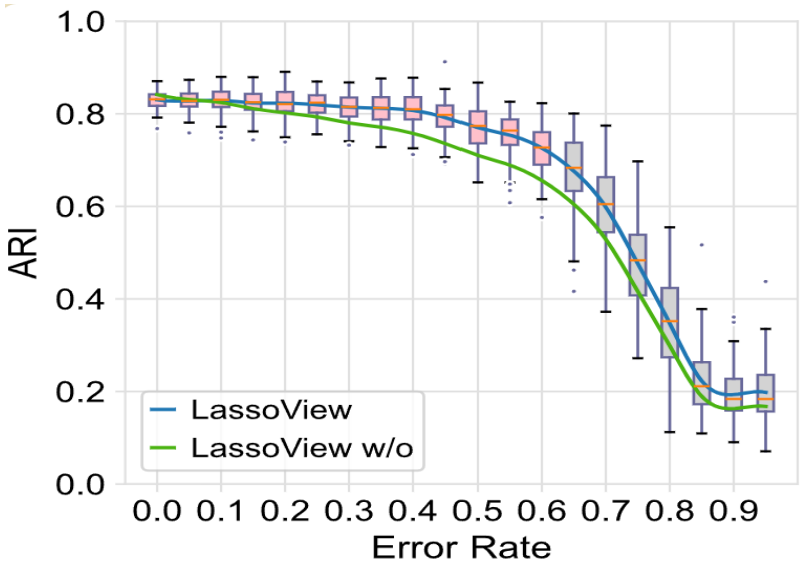

图 5: LPA 算法的 ARI 指数图

四、 进阶要求实现与评价结果展示

(一) LPA 标签传播模型优化

在基本要求实现的基础上，我们还对 LPA 标签传播算法进行改进（Opt-LPA）以及使用其他种类的机器学习算法进行标签传播模型的优化尝试，以下几种方法的评价方式和数据生成方式均与基于要求部分保持一致，下面我们将重点展示基于新算法的标签传播模型的实现过程以及对应的算法性能。

在进阶任务中，我们对优化 LPA 算法进行了初步尝试，我们注意到 sc.pp.neighbors(adata) 默认会在图中构建每个节点的 15 个元素，但是在实现过程中我们发现，若标签错误率较低，那么过多的邻居将干扰标签传播的准确度，此时我们可以仅使用 8 个邻居节点的信息进行标签传播算法；若标签错误率较高，那么更多邻居节点将更有效的反映整个图的信息，因此适当使用更多的邻居将增强 LPA 算法的鲁棒性。因此我们将首先试计算一次 ARI，然后按照 ARI 的取值依照区间 [8,12,15,20] 依次选择对应的邻居节点数量。

此外，原先邻接图的构建方法使用欧氏距离，这或许并不能完全解释细胞之间线性相关性。我们分析细胞邻接图依据基因表达情况进行构建的，因此可以采用皮尔逊相关系数表达细胞之间的关联程度。其公式为：

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

其中，x_i 和 y_i 分别是样本 x 和样本 y 中的第 i 个数据点，$\bar{x}$ 和 $\bar{y}$ 是样本 x 和样本 y 的均值，n 是样本的数量。

1. 改进的 LPA 标签传播模型实现

改进的 LPA 标签传播模型

```
def labelPropagation(X: matCoo, y_label: mat, y_pred: mat, y_res: mat, alpha: float = 0.5, max_iter: int = 1000):
    ......省略初始化部分代码......
    # 计算相似度矩阵 W
    W = matCoo(n_samples, n_samples)  # 创建稀疏矩阵 W, 存储样本之间的相似度
    for elem in X.elem:
        row = elem.row
        col = elem.col
        val = elem.v
        # 计算两个样本之间的皮尔逊相关系数
        similarity = corr(row, col)  # 使用皮尔逊相关系数计算相似度
        W.append(row, col, similarity)  # 将相似度添加到矩阵 W 中
    W.totalElements = len(W.elem)  # 记录 W 中的元素数量
    Y_old = mat()  # 存储上一次迭代的标签矩阵
    Y_new = mat()  # 存储当前迭代的标签矩阵
    # 迭代标签传播过程
    for iter_num in range(max_iter):
        Y_old.assign(Y_new)  # 将当前 Y_new 的值赋给 Y_old，保存上一次的标签
        matMultiply_x1_x2_res(W, Y, Y_new)  # 使用相似度矩阵 W 和标签矩阵 Y 进行矩阵乘法，更新 Y_new
        # 遍历所有样本，更新标签
```

```
        for i in range(n_samples):
            # 计算当前样本的标签值总和，进行归一化
            row_sum = np.sum(Y_new.v[i, :])
            if row_sum != 0:
                Y_new.v[i, :] /= row_sum  # 归一化标签概率分布
            # 检查该样本是否有先验标签
            has_prior = False
            for j in range(n_classes):
                if y_label.v[i, j] != -1:  # 如果标签不为 -1，说明该样本有先验标签
                    Y_new.v[i, j] = y_label.v[i, j]  # 将先验标签赋值给 Y_new
                    has_prior = True
                    break  # 找到先验值后退出内层循环
            # 如果没有先验标签，则更新 Y
            if not has_prior:
                Y.v[i, :] = Y_new.v[i, :]  # 用 Y_new 更新 Y 中的标签值
    ..... 省略预测及修正部分代码 .....
```

2. 改进的 LPA 标签传播模型的性能评估与分析

已知标签错误率	未修正 ARI	已修正 ARI
0.0	0.8607025080438879	0.8569499361361422
0.05	0.8352319723149705	0.8339867935987297
0.1	0.8498593984023911	0.8515746974046982
0.15	0.8054609646516404	0.8214541175456828
0.2	0.8029434433176845	0.8260760419060241
0.25	0.7555481804573041	0.7902792877578423
0.3	0.7240547840422906	0.7576506465360572
0.35	0.7406051321431951	0.7810386650830382
0.4	0.7249919552331184	0.7765268079324245
0.45	0.6879520274940104	0.7517852142947051
0.5	0.6470896058856911	0.6991668728329887
0.55	0.6100842770983099	0.6641834883782889
0.6	0.6226858633478577	0.6902882869185195
0.65	0.42400450524217975	0.4791056917855146
0.7	0.4313334073310493	0.48641411386504163
0.75	0.2822593658630945	0.32124752966851455
0.8	0.254761634446954	0.29632095184863516
0.85	0.07312854066565244	0.07913907427294575
0.9	0.11778189347435712	0.13326372074861767
0.95	0.08297050820894285	0.09316906305339627

表 2: 改进的 LPA 标签传播模型性能评价结果

结果如图所示:

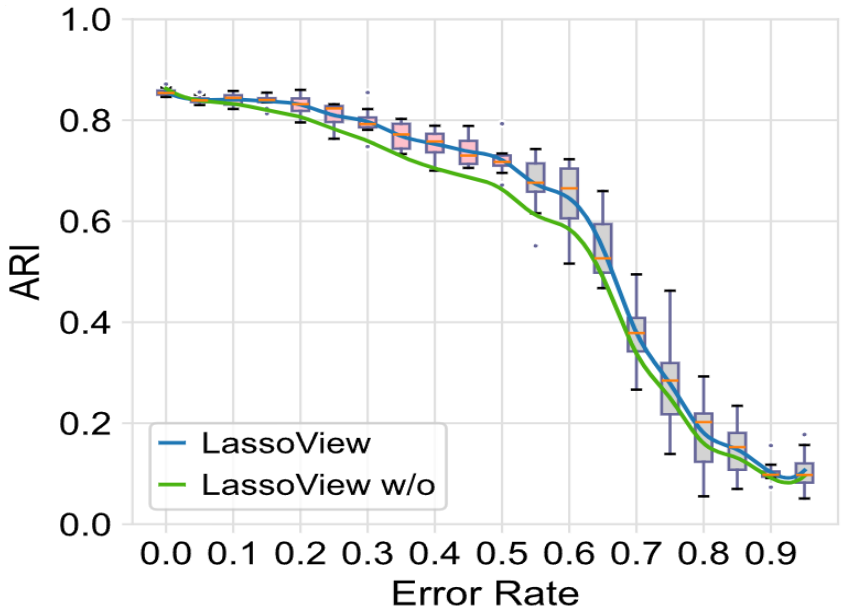

图 6: 改进的 LPA 标签传播模型的 ARI 指数图

(二) 基于生成对抗网络 GAN 的标签传播模型优化

我们结合生成对抗网络 (GAN) 和标签传播 (Label Propagation) 的模型，用于处理半监督学习问题，特别是本实验中的图数据上的标签传播任务。

1. 基于 GAN 的标签传播模型实现过程

基于 GAN 的标签传播型

```python
# 定义 GAN 标签传播模型，结合生成对抗网络（GAN）与标签传播
class GANLabelPropagation(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(GANLabelPropagation, self).__init__()
        # 初始化模型结构
        # 定义两层图卷积神经网络
        self.gcn1 = GCNLayer(input_dim, hidden_dim)
        self.gcn2 = GCNLayer(hidden_dim, hidden_dim)
        self.gcn2 = GCNLayer(hidden_dim, output_dim)
        self.criterion = nn.CrossEntropyLoss()  # 使用交叉熵损失函数
        # 定义 GAN 中的判别器部分
        self.discriminator = nn.Sequential(
            nn.Linear(output_dim, 64),  # 第一层全连接
            nn.ReLU(),
            nn.Linear(64, 128),  # 第二层全连接
            nn.ReLU(),
            nn.Linear(128, 1),  # 第三层全连接
            nn.Sigmoid()  # Sigmoid 激活函数
        )
    def forward(self, adj_matrix, features):
        # 前向传播: 首先通过图卷积层进行标签传播
```

```
22          x = torch.relu(self.gcn1(adj_matrix, features)) # 经过第一层 GCN 后进行 ReLU 激活
23          x = self.gcn2(adj_matrix, x) # 经过第二层 GCN
24          return x # 返回最终的输出（预测的标签分布）
25      def discriminator_loss(self, y_real, y_fake):
26          # 判别器的损失函数，计算真实样本与伪造样本的二元交叉熵损失
27          real_loss = torch.mean(torch.log(y_real)) # 真实样本的损失
28          fake_loss = torch.mean(torch.log(1 - y_fake)) # 伪造样本的损失
29          return -(real_loss + fake_loss) # 返回总的判别器损失
30      def generator_loss(self, y_fake):
31          # 生成器的损失函数，计算生成器生成伪造样本的损失
32          return -torch.mean(torch.log(y_fake)) # 生成器的损失
33      def label_propagation(self, adj_matrix, y_label, epochs=10, lr=0.01, device='cuda'):
34          # 标签传播过程：通过优化模型使得标签得以传播
35          optimizer = optim.Adam(self.parameters(), lr=lr)
36          mask = (y_label != -1).astype(np.float32) # 创建掩码，标记已知标签的位置
37          y_label_tensor = torch.tensor(y_label, dtype=torch.float32).to(device)
38          y_label_tensor = y_label_tensor * torch.tensor(mask, dtype=torch.float32).to(device) # 根据掩码处理标签
39          # 初始化特征矩阵，这里使用单位矩阵作为初始特征
40          features = torch.eye(adj_matrix.shape[0]).to(device)
41          # 训练过程
42          for epoch in range(epochs):
43              self.train()
44              optimizer.zero_grad()
45              # 前向传播
46              outputs = self(adj_matrix.to(device), features)
47              # 计算已知标签的损失
48              labeled_loss = self.criterion(outputs[mask == 1], y_label_tensor[mask == 1])
49              # 更新 GAN 判别器
50              y_real = self.discriminator(outputs) # 判别器判断真实输出
51              y_fake = self.discriminator(outputs.detach())# 判别器判断伪造输出
52              d_loss = self.discriminator_loss(y_real, y_fake) # 判别器损失
53              g_loss = self.generator_loss(y_fake) # 生成器损失
54              # 总损失：标签传播损失 + GAN 损失
55              total_loss = labeled_loss + d_loss + g_loss
56              # 反向传播
57              total_loss.backward()
58              optimizer.step() # 更新模型参数
59              # 每10轮输出一次损失
60              if epoch % 10 == 0:
61                  print(f'Epoch {epoch}/{epochs}, Loss: {total_loss.item()}')
62          # 返回预测结果和精炼后的标签
63          with torch.no_grad():
```

```
64            self.eval()  # 设置模型为评估模式
65            y_pred = torch.argmax(outputs, dim=1).cpu().numpy()  # 获取预测的
                  标签
66            y_res = torch.argmax(outputs, dim=1).cpu().numpy()  # 获取精炼后
                  的标签
67        return y_pred, y_res  # 返回预测结果和精炼后的标签
```

1. 模型定义

图卷积层：使用两层图卷积网络对节点特征进行提取。输入邻接矩阵（表示图的结构）和节点特征矩阵，输出每个节点的标签分布。

判别器：判别器通过三层全连接网络，判断标签传播生成的标签是否可信。输出值通过 Sigmoid 激活，表示判别为“真实标签”的置信度。

2. 损失函数

标签传播损失：对已标注样本，使用交叉熵损失计算预测标签与真实标签之间的误差。

判别器损失：通过二元交叉熵损失，区分“真实样本”（有标注）与“伪造样本”（预测标签）。

生成器损失：鼓励生成的标签让判别器更倾向于判断为真实样本，提高预测标签的可信度。

总损失 = 标签传播损失 + 判别器损失 + 生成器损失。

3. 训练流程

初始化：使用 Adam 优化器，设置学习率等超参数。将邻接矩阵、特征矩阵和标签数据加载到模型。

前向传播：GCN 标签传播：通过图卷积网络传播标签，输出预测的标签分布。

判别器预测：对 GCN 输出的标签进行判别，区分“真实”与“预测”。

反向传播与参数更新：计算总损失，并通过反向传播更新 GCN 和判别器的参数。

2. 基于 GAN 的标签传播模型的性能评估与分析

我们衡量的 GAN 算法性能如下表所示：

已知标签错误率	未修正 ARI	已修正 ARI
0.0	0.8871128469194094	0.8871128469194094
0.05	0.8757303537919072	0.8757303537919072
0.1	0.3429397394517232	0.3429397394517232
0.15	0.21020742171134582	0.21020742171134582
0.2	0.14948825795580606	0.14948825795580606
0.25	0.08441895511573567	0.08441895511573567
0.3	0.0448645404700105	0.0448645404700105
0.35	0.041966790627480176	0.041966790627480176
0.4	0.01578282520276142	0.01578282520276142
0.45	0.015648716126162785	0.015648716126162785
0.5	0.014351668564801974	0.014351668564801974
0.55	0.008009248038443351	0.008009248038443351
0.6	0.007185117344663178	0.007185117344663178
0.65	0.002737982388260387	0.002737982388260387
0.7	0.0033542389674259053	0.0033542389674259053
0.75	0.002256426060194775	0.002256426060194775
0.8	0.00024821854599292256	0.00024821854599292256
0.85	0.0032104612586080863	0.0032104612586080863
0.9	0.0007772949118791389	0.0007772949118791389
0.95	0.0014512134718702168	0.0014512134718702168

表 3: 基于 GAN 的标签传播算法性能评价结果

结果如图所示:

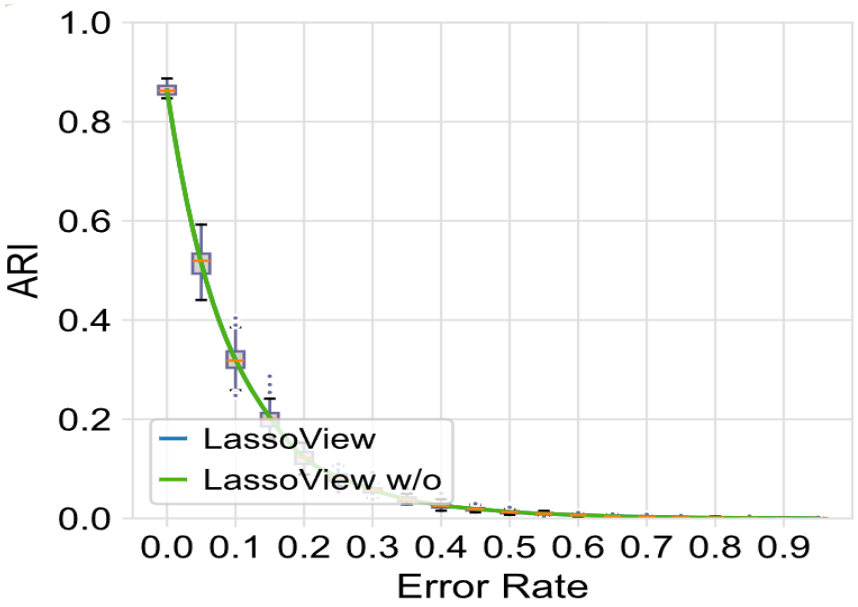

图 7: 基于 GAN 的标签传播算法的 ARI 指数图

(三) 基于图神经网络 GNN 的标签传播模型优化

我们将标签传播算法结合多层 GNN 与标签传播，尝试提高预测准确性和鲁棒性。

1. 基于 GNN 的标签传播模型实现过程

ARI_o 计算

```
def labelPropagation_GNN(X: matCoo, y_label: mat, y_pred: mat, y_res: mat,
                         hidden_dim: int = 32, num_layers: int = 4, max_iter
                             : int = 200, lr: float = 0.005, weight_decay:
                             float = 5e-4):
    # X: 稀疏矩阵，包含样本的相似性数据
    # y_label: 包含已标记样本的标签，未标记样本的标签为 -1
    # y_pred: 预测结果，算法最终输出
    # y_res: 修正后的预测结果
    # hidden_dim: GNN 隐藏层的维度
    # num_layers: GNN 的层数
    # max_iter: 训练的 epoch 数

    n_samples = X.n
    n_classes = y_label.m

    # 构建 PyG 数据对象
    edge_index = torch.tensor([[elem.row, elem.col] for elem in X.elem],
        dtype=torch.long).t()
    edge_weight = torch.tensor([elem.v for elem in X.elem], dtype=torch.float
        )

    # 初始化节点特征和标签
    x = torch.eye(n_samples, dtype=torch.float)
    y = torch.tensor(y_label.v, dtype=torch.float)

    # 为有标签的样本构造 mask
    mask = (y_label.v != -1).any(axis=1)
    train_mask = torch.tensor(mask, dtype=torch.bool)
    train_labels = torch.argmax(y[train_mask], dim=1)

    # 创建 PyG 数据
    data = Data(x=x, edge_index=edge_index, edge_attr=edge_weight)
    # 初始化 GNN 模型
    model = OptimizedGNN(input_dim=n_samples, hidden_dim=hidden_dim,
        output_dim=n_classes, num_layers=num_layers, dropout=0.5)
    optimizer = torch.optim.Adam(model.parameters(), lr=lr, weight_decay=
        weight_decay)

    # 训练模型
    model.train()
    for epoch in range(max_iter):
```

```
        optimizer.zero_grad()
        out = model(data.x, data.edge_index)
        loss = F.nll_loss(out[train_mask], train_labels)  # 使用负对数似然损失函数
        loss.backward()
        optimizer.step()

        if epoch % 10 == 0 or epoch == max_iter - 1:
            print(f"Epoch {epoch}/{max_iter}, Loss: {loss.item():.4f}")

    # 推断未标记样本的标签
    model.eval()
    with torch.no_grad():
        out = model(data.x, data.edge_index)
        y_pred_np = out.argmax(dim=1).numpy()

    # 赋值预测结果
    y_pred.createmat(n_samples, 1)  # 创建一个新的矩阵来存储预测结果
    for i in range(n_samples):
        y_pred.v[i, 0] = y_pred_np[i]

    print("Optimized multi-layer GNN-based label propagation complete.")

    # 修正预测结果
    rectify(X, y_label, y_pred, y_res)
```

1. 输入与初始化

输入数据:

X: 稀疏相似性矩阵，包含节点间的边信息和权重。

y_{label}: 标签矩阵，标记样本的标签为真实值，未标记样本为 -1。

y_{pred} 和 y_{res}: 分别存储预测标签和修正后的标签。

PyG 数据构建: 转换 X 为边索引 (edge_index) 和边权重 (edge_weight)，以 PyG 的 Data 对象形式表示图结构数据。节点特征初始化为单位矩阵（每个节点的特征独立)。

训练样本掩码: 根据 y_{label}，创建 train_mask 标记已标注节点，并提取对应的真实标签 train_labels。

2. GNN 模型构建

OptimizedGNN 是一个多层 GNN 模型，输入节点特征与边信息，通过隐层嵌入学习输出每个节点的标签分布。使用 Adam 优化器, 学习率为 $lr = 0.005$, 并设置权重衰减 ($weight\_decay = 5 \times 10^{-4}$)，以防止过拟合。使用负对数似然损失 (NLL Loss)，仅针对有标注的样本计算损失。

3. 模型训练

通过 $max\_iter = 200$ 的最大迭代次数进行训练，主要流程如下:

前向传播: GNN 模型计算每个节点的嵌入表示，输出为节点的标签分布。

损失计算: 针对标注样本，通过 NLL Loss 计算模型预测与真实标签间的误差。

梯度更新: 通过反向传播优化 GNN 模型的参数。

进度输出: 每 10 轮或最后一轮输出损失值，便于观察训练效果。

4. 推断与结果分配

未标记样本预测：模型训练完成后，利用 GNN 模型对所有节点进行推断，输出每个节点的标签分布，并选择概率最高的标签作为预测结果。

结果存储：将预测标签存入 y_{pred}，为后续修正准备。

5. 修正预测结果

调用 rectify 函数，根据节点的邻居标签分布对预测结果进行修正。检查邻居标签的多数分布。优先保留已标注样本的标签，修正未标注样本的预测结果。修正后的标签存储在 y_{res}。

2. 基于 GNN 的标签传播模型的性能评估与分析

已知标签错误率	未修正 ARI	已修正 ARI
0.0	0.8972631795044452	0.8904973091130401
0.05	0.79177541546505	0.7946289628013696
0.1	0.7281464599718355	0.7420987168676892
0.15	0.6088726124437196	0.6395460398771106
0.2	0.495136642498576	0.5248130970152545
0.25	0.39083291564868355	0.4319010122746426
0.3	0.14737872469966917	0.17210124709631017
0.35	0.1522626942847077	0.14743189719955968
0.4	0.20149586786785856	0.2284921824541999
0.45	0.23363958804067908	0.26377252494796266
0.5	0.22365895551881587	0.28063630501497777
0.55	0.07866149419367222	0.09831964937639771
0.6	0.045111365404812385	0.06101300563480302
0.65	0.049250267688081224	0.05903378073172371
0.7	0.061770652240861354	0.05046529879773083
0.75	0.0055218784551513125	0.01725637124854113
0.8	0.0031853453333321874	0.0036801759855119138
0.85	0.0008934340538120864	0.007137394160025594
0.9	0.001337361566875778	0.013792856892130716
0.95	0.0004516276616394498	0.0021083777744847613

表 4: 基于 GAN 的标签传播算法性能评价结果

结果如图所示:

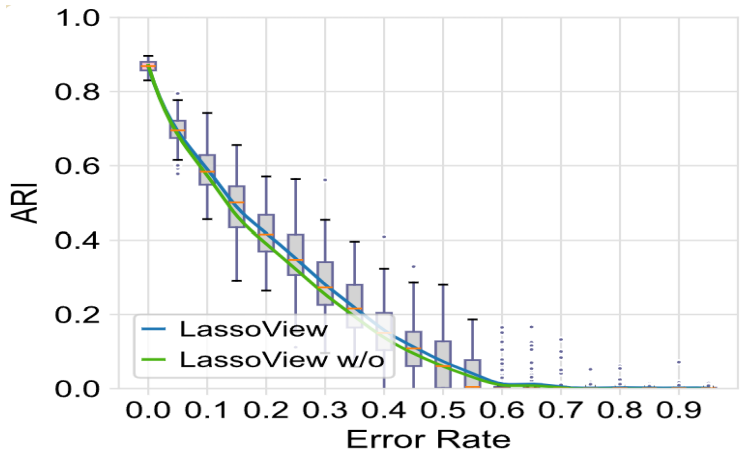

图 8: 基于 GAN 的标签传播算法的 ARI 指数图

五、 大作业总结

(一) 完成任务概述

我们在本次大作业中完成了基于机器学习和标签传播的半监督学习任务，进行了在单细胞 RNA-seq 数据和图结构数据上的分类与预测。我们使用了普通 LPA 标签传播算法来实现细胞群预测。此外，我们还结合了改进 LPA 算法、图神经网络 GNN 和生成对抗网络 GAN 分别对 LPA 模型进行了优化，虽然后两者优化后错误率为 0 时的 ARI 值相较于基准方法有较大提升，但模型鲁棒性有所降低，抗干扰能力弱。

我们对优化后的模型的鲁棒性降低进行了分析。神经网络的模型复杂度增大会增加过拟合风险，因为复杂模型更容易拟合训练数据中的噪声或过于依赖特定数据分布，从而降低泛化能力。这就是为什么在错误率上升的时候，我们使用神经网络的模型 ARI 指标下降得更快的原因；另外，标签传播具有长距离依赖性，需要依赖图中节点的多跳邻居关系，但远距离传播容易受低质量邻居影响，而 GNN 通过堆叠多层图卷积捕捉长距离依赖关系，每层传播都会导致误差累积，进而影响最终的预测结果。这是我们需要反思的地方。

(二) 作业亮点之处：优化模型性能的做法

1. 引入标准分析流来进行数据集的初始化工作

在单细胞 RNA-seq 数据分析中，数据的高维度、稀疏性和噪声特性需要一套标准化的预处理方法，以确保后续分析的准确性和有效性。引入标准分析流可以有效规范数据处理流程，为后续模型训练提供高质量的输入。

2. 使用多种机器学习方法来对细胞群预测进行优化

我们通过多种机器学习方法（改进 LPA、GNN、GAN 等），在某些方面显著优化和扩充了细胞群预测任务的能力，但也引入了一些模型鲁棒性方面的问题。最终，我们希望通过不同方法的尝试能够更好地应对单细胞 RNA-seq 数据和图结构数据的多样性和复杂性，为半监督学习任务提供更强大的支持。