西安电子科技大学人工智能学院

专业基础实践 课程实践报告

实践课题名称 基于局部路径指标的

复杂网络链路预测

成 绩

班级 2020031 姓名 牛志康 学号 20009201191

同作者 无

实践日期 2022 年 4 月- 6 月

指导教师评语：
指导教师： 年 月 日

实践报告内容基本要求及参考格式

一、 实践题目功能描述；

复杂网络上的链路预测是指通过已有的网络信息预测网络中缺失的连边或者未来可能出现的连边，其中一种基本算法的思想是两个节点之间相连的路径越多，则它们之间存在连边的可能性就越大，其中长度越短的路径对连边的贡献度越大。为了在算法的精度和计算复杂性之间进行折衷，人们提出了局部路径指标，它利用两个节点之间的二阶和三阶路径的数目预测两个节点之间存在连边的可能性,这就是所谓的局部路径指标（Local Path, LP)。

学生需要实现以下几方面的内容：

1. 网络连接矩阵的读入
2. 编程实现的 LP 指标的计算
3. 利用 LP 指标预测网络中新的连边
4. 设计界面并显示预测网络中新的连边
5. 提交演示系统，完成实践报告

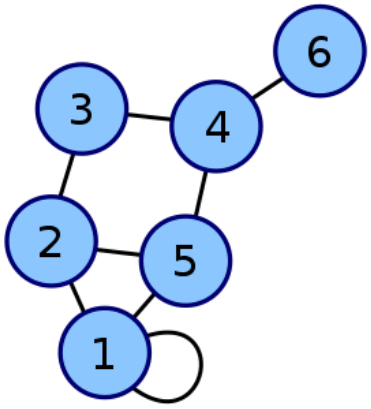
二、 问题分析及算法或方案选择、理论知识基础简要介绍；

2.1 邻接矩阵求路径

在开始之前，我们需要进行图的读入。图是一种非线性数据结构，由节点（顶点）vertex 和边 edge 组成，每条边连接一对顶点。根据边的方向有无，图可分为有向图和无向图，根据是否含有权重还可以分为有权图和无权图。在本文中，我们仅限于探讨基于无向无权图的局部路径复杂网络的链路预测。定义 G（V，E）为该无向无权网络，其中 V 为节点集合，E 为边集合。网络总的节点数为 N，边数为 M。此网络共有 N(N-1)/2 个节点对，即全集 U。给定一种链路预测的方法，对每对没有连边的节点对 x，y 赋予一个分数值 S。然后将所有未连接的节点对按照该分数值从大到小排序，排在最前面的节点对出现连边的概率最大。

常见的图的表示方法包括邻接矩阵和邻接表表示法。我们将图按照邻接表的方式进行读取时可以用邻接矩阵对图进行表示。邻接矩阵（英语：adjacency matrix）常代表每个元素代表各点之间是否有边相连，以无向图为例，无向图的

邻接矩阵计算方法是每条边为对应的单元加上 1，而每个自环加上 2。具体如下图所示，其右边就表示了左边图的邻接矩阵。左图代表的是图的表示，右图是左图无向图的邻接矩阵，其中行列代表 1~6，以第一个元素为例，(1，1）处代表的

无向图	邻接矩阵
	$\begin{pmatrix} 2 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$ 从左到右、从上到下分别代表1-6节点。

是（1，1）之间的连线关系，因为 1 为自环，所以连接关系记为 2，(1，2）处代表的是（1，2）节点的连接关系，因为 1，2 结点之间存在连接关系，因此在邻接矩阵中记为 1，又因为左图无向图，所以邻接矩阵呈对称关系，(2，1）处同样也为 1。

邻接矩阵不仅能很好的表示图的性质，还可以通过求取邻接矩阵的平方和立方项来解决二阶和三阶路径的计算。因为基于局部路径指标的复杂网络链路预测牵扯到结点间二阶路径和三阶路径的求解，因此我们使用邻接矩阵来表示图。

2.2 局部路径指标计算

局部路径指标是在共同邻居指标的基础上考虑了三阶邻居的贡献，其计算公式为：

$$S = A^2 + \alpha A^3$$

其中 α 为可调节参数，代表三阶邻居的贡献参数。根据上述的邻接矩阵的性质，我们可以对邻接矩阵求取平方和立方项以实现得分矩阵的计算。在实现时，我们采用了 Numpy 提供的 np.dot()操作以实现矩阵的运算。由于输入的邻接矩阵 A 是对称矩阵，在计算分数时，我们仅使用上三角矩阵或者下三角矩阵进行计

算即可，不仅可以大大减少计算量，同时也可以有效避免两节点间的重复连接。

除此之外，为了让所得分数都在 0~1 之间，并且所有分数之和为 1，我采取了 softmax 函数对整体分数进行了变化，softmax 的函数定义如下：

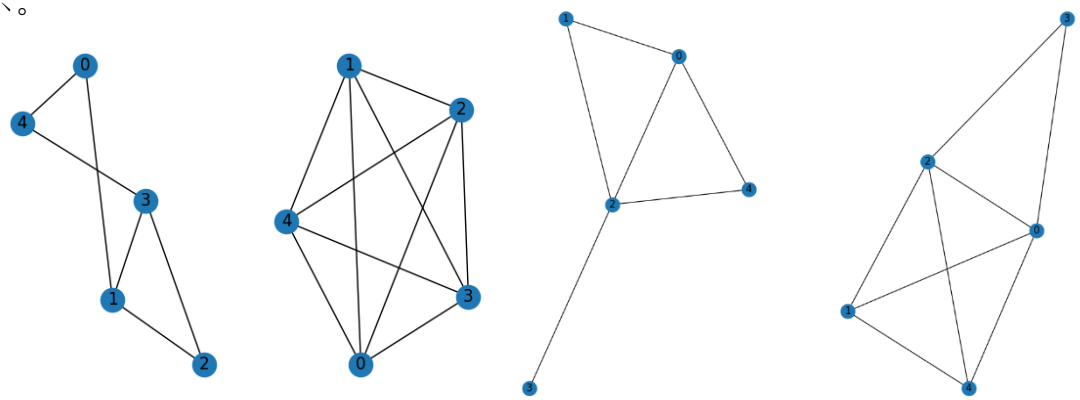
$$P(y=j)=\frac{e^{x^T W_j}}{\sum_{k=1}^{K} e^{x^T W_k}}$$

在算法实现时，我也基于 Python 中的 Numpy 实现了 softmax 函数的具体表达式，具体实现形式如下：

```
1    def softmax(x):
2        exp_x = np.exp(x)
3        softmax_x = exp_x/np.sum(exp_x)
4        return softmax_x
```

2.3 可视化路径连接展示

我们的可视化设计采用了 python 中的 networkx 和 matplotlib 库,networkx 是一个基于 Python 语言开发的图论与复杂网络建模工具,内置了非常多的常用的图与复杂网络分析算法,可以方便的帮助我们进行复杂网络数据分析、仿真建模等工作。Matplotlib 同样也是一个基于 Python 实现的画图工具，基于上述两种工具，我实现了将原始图和添加边后的图放在一起进行对比的效果，如下图所示。



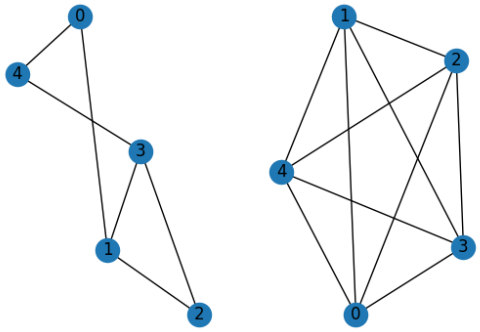
三、 仿真实践结果与分析；

在这部分，我将展示两组实验结果以表现出当前算法的效果，其中第一组输入的对应的邻接矩阵为下表，其所示图与添加边的结果如下图所示，其中左边为

原图，右边为新加边后的图。

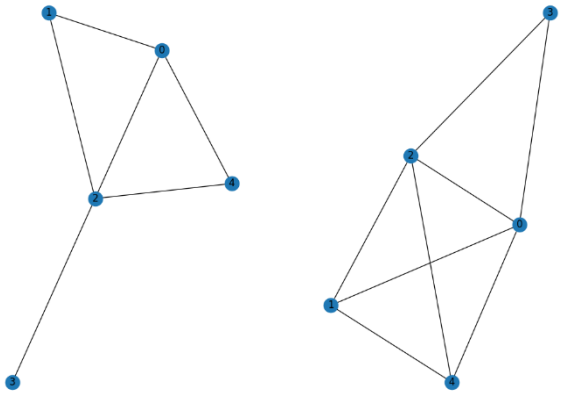
我们的程序首先会计算邻接矩阵A的平方项A^2及其立方项A^3，并基于局部路径指标计算公式$S = A^2 + \alpha A^3$计算得分矩阵，并将得到的得分矩阵通过 softmax 函数使得所有值的和为 1，在计算得到结果后，我们将连接的阈值设为 0.002 时，可以得到原始图添加了（0，2）（0，3）（1，4）（2，4）四条边，其中添加的边的个数随着阈值的变化进行变化。在实现代码时，我也通过整个矩阵的平均值作为阈值，但是发现邻接矩阵的分布并不均匀，平均值做阈值的效果较差。

	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	1	0



与第一组方法相同，第二组对应的邻接矩阵为下表，在计算完后，我们可以发现新添加了（0，3）（1，4）（2，3），其所示图与添加边的结果如下图所示，其中左边为原图，右边为新加边后的图：

	0	1	2	3	4
0	0	1	1	0	1
1	1	0	1	0	0
2	1	1	0	1	1
3	0	0	1	0	0
4	1	0	1	0	0



整体两个示例取得了较良好的结果，但是在阈值超参数的选择上仍然存在不足，阈值超参数的选择影响着新的边的连接。当阈值较高，会使得没有新的边连接，新图与原图相同；但是当阈值较低时，会出现较多的新边。因此阈值的选择也是

四、　程序源代码（添加注释）；

```
1    """
2        a  b  c  d  e
```

```
    a   0  1  0  0  1
    b   1  0  1  1  0
    c   0  1  0  1  0
    d   0  1  1  0  1
    e   1  0  0  1  0

    用邻接矩阵读取

    邻接矩阵的平方来进行计算

    传入参数两个点(以元组形式传入)
        计算二长路，并得出数量，邻接矩阵平方项相对应的数字
        计算三长路，并得出数量，邻接矩阵立方项对应的数字

    计算公式给予二阶一个参数，三阶一个参数，返回一个结果（二分类）
    """

    import numpy as np
    import networkx as nx
    import matplotlib.pyplot as plt

    def LocalPathMatrix(AdjacenceyMatrix):
        # 需要判断矩阵是否为空或者全 0
        if AdjacenceyMatrix.sum() == 0:
            scoreMatrix = np.zeros(AdjacenceyMatrix.shape)
            return scoreMatrix
        else:
            # 计算邻接矩阵的平方和立方项
            L2Matrix = np.dot(AdjacenceyMatrix,AdjacenceyMatrix
    )
            L3Matrix = np.dot(L2Matrix,AdjacenceyMatrix)
            # 三阶路径的参数
            # Alpha 的参数不应该是一个定值，应该根据二阶和三阶的数目
     来确定
            L2alpha = L2Matrix.sum()
            L3alpha = L3Matrix.sum()
            if (L2alpha+L3alpha) != 0:
                Alpha = L3alpha/(L2alpha+L3alpha)
            else:
                Alpha = 0.5
            # 计算得分矩阵
            scoreMatrix = L2Matrix + Alpha * L3Matrix
            # 返回结果矩阵
            return scoreMatrix
```

```
45
46  def softmax(x):
47      exp_x = np.exp(x)
48      softmax_x = exp_x/np.sum(exp_x)
49      return softmax_x
50
51  def clshead(scoreMatrix,checkpoint=None):
52      # 矩阵归一化
53      NormalizedMatrix = softmax(scoreMatrix)
54      if checkpoint is not None:
55          # 获取两点的坐标
56          i = checkpoint[0]
57          j = checkpoint[1]
58          score = NormalizedMatrix[i][j]
59          return NormalizedMatrix,score
60      else:
61          return NormalizedMatrix
62
63
64  if __name__ == '__main__':
65      AdjacenceyMatrix = np.array(eval(input("please input Gr
    aph AdjacenceyMatrix: ")))
66      input_graph = nx.from_numpy_array(AdjacenceyMatrix)
67      print(input_graph)
68
69      scoreMatrix = LocalPathMatrix(AdjacenceyMatrix)
70      flag = eval(input("if you need check points prob, pleas
    e input 1: "))
71      if flag == 1:
72          checkpoint = tuple(eval(input("please input the poi
    nt that you want to check: ")))
73          NormalizedMatrix,prob = clshead(scoreMatrix,checkpo
    int=checkpoint)
74      else:
75          NormalizedMatrix = clshead(scoreMatrix)
76      print('-'*6 + 'scoreMatrix' + '-'*6)
77      print("ScoreMatrix = \n{}".format(scoreMatrix))
78      print('-'*6 + 'Normalized scoreMatrix' + '-'*6)
79      print("After Normalized ScoreMatrix = \n{}".format(Norm
    alizedMatrix))
80      print('-'*6 + 'prob of point' + '-'*6)
81
82      # 计算平均值和矩阵的大小
83      mean = 1/AdjacenceyMatrix.size
```

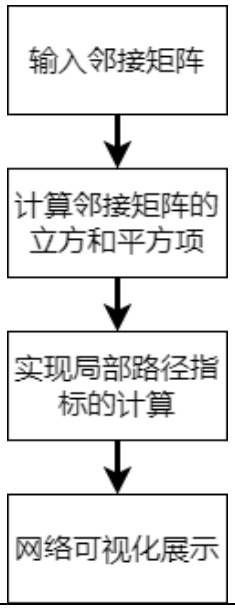
```
        row,col = AdjacenceyMatrix.shape
        mean = 0.002
        print("threshold = {}".format(mean))
        # 对矩阵进行copy，并对符合条件的置1，不满足置0
        addEdgeMetrix = NormalizedMatrix.copy()
        addEdgeMetrix[addEdgeMetrix>mean] = 1
        addEdgeMetrix[addEdgeMetrix<=mean] = 0

        # 记录下需要加点的曲线
        new_graph = input_graph.copy()
        for j in range(col):
            for i in range(row):
                if addEdgeMetrix[i][j] == 1 and i!=j:
                    new_graph.add_edge(i,j,color="red")
                else:
                    continue

        subax1 = plt.subplot(121)
        nx.draw(input_graph,with_labels = True)   # default spring_layout
        subax2 = plt.subplot(122)
        nx.draw(new_graph,with_labels=True)
        plt.show()
```

五、 实践过程或算法总结；

实践过程如下图流程图所示：



六、 心得体会；（如解决了什么问题，还存在什么问题，哪些方面得到了提高或受到了启发；在设计该算法或解决方案时，遇到的最大困难是什么？是如何解决的？）

网络中的链路预测是指如何通过已知的网络节点以及网络结构等信息预测网络中尚未产生连边的两个节点之间产生连接的可能性。这种预测既包含了对未知链接的预测，也包含了对未来链接的预测。链路预测问题对实际生产生活有着重要的意义，本次专业实践基础的课题是基于局部路径指标的复杂网络链路预测，完成了代码的实现和可视化的展示。当前整个实践代码虽然已经完成，但是尚存在几个不足：

1. 局部路径的链路连接指标不清晰，超参数的设置不合理，比如在阈值的选择上按照平均制的指标不够清晰准确，因为结点的连接分数不是呈正态分布，可能出现过大值和过小值影响整体的阈值的选择，从而影响到整体预测的新边上。因此在阈值的选择上，应该忽略最小值和最大值等异常值的影响，以免影响到整体的预测，可以使用除去一个最大值一个最小值求平均的方式进行，尽可能忽略异常值的影响。
2. 链路预测的结果没有经过常见的链路预测数据集的测试，基于局部路径指标的链路预测，由于仅仅依赖二阶和三阶路径，在大规模数据集上表现欠佳。

在本次学习的过程中我也了解到了链路预测的传统的方法例如基于相似性的链路预测，基于最大似然的链路预测，基于概率模型的链路预测和一些基于深度学习图神经网络 GNN 的方法，对本专业有了一个更加清晰的认识。除此以外，本次实验均是我一个人完成，也在这个过程中提高了自己的代码能力和思考问题、解决问题的能力。

在设计核心算法时，我遇到的最大问题是二阶和三阶路径的求法不能与图的表示相结合起来，在最初的思考，采用了遍历结点的方法求取路径，这种方法虽然可以表示路径，但由于使用了大量的循环判断语句，使得在时间复杂度上较复杂，影响了计算效率。为了解决这个问题，我在查阅大量图论相关材料后，发现以邻接矩阵表示图，可以通过求邻接矩阵的平方项和立方项轻松的求解二阶

和三阶路径，从大量的循环判断运算转换为矩阵运算后，大大减少了计算量，提高了计算速度。

最后，关于本次专业实践基础我也十分感谢刘波老师对我的指导与帮助。

七、 小组每个人的详细分工及自我评价（说明每人承担的具体任务和完成情况）

姓名	学号	小组分工与贡献	自我评价（优秀、良好，中等、及格、不及格）
牛志康	20009201191	负责查阅资料，代码实现，可视化实现	优秀