

**计算机专业基础强化荣誉课作业**

**基于“混合型数据的混合因果结构学习算法”的代码复现**

**（A Hybrid Causal Structure Learning Algorithm for Mixed-type Data）**

组 别： **05**

论文标号： **14**

论文选题：**混合型数据的混合因果结构学习算法**

课 号： **10077601**

二〇二二年五月

**小组成员**

|  |  |
| --- | --- |
| **2053182** | **王润霖** |
| **1951458** | **周宏韬** |
| **1953246** | **方若愚** |
| **2052610** | **郭 茜** |
| **2052911** | **顾启文** |

基于“混合型数据的混合因果结构学习算法”的代码复现

摘要

本次作业主要解决选定论文的代码复现任务，并对我们的实现方法、心得体会等进行了介绍。

发现多个变量之间潜在的因果关系在许多应用中都是有益的，进行因果关系识别实验通常成本高昂、耗时。因此，需要从纯粹的观测数据中发现因果关系，这通常称为因果发现或因果结构学习。本组选择的论文，提出了观测混合型数据中学习因果结构问题的一种算法，该算法在发现混合型数据的因果关系中，具有明显的优势。

本组（第05小组）首先深入分析了选定论文（选定标号14）的框架结构，并对文章的内容和设计的算法进行了仔细阅读，梳理了选定论文的行文结构并绘制了文章的框架图。然后，我们将课程任务划分为“混合结构方程模型与算法的可行性证明”、“混合型数据的混合因果结构学习算法的完整代码复现”、“文中算法模型与其他方法优势分析”三个主要子任务；除此之外，我们对文中算法的实际应用、课程作业中的心得体会进行了阐述。

对于任务一，本组在理解文章内容的基础上，结合参考文献，对混合SEM可识别性的形式证明、CVMIC分数的局部一致性、混合型数据的随机条件独立性检验进行了阐释和证明。

对于任务二，本组在阅读大量文献资料和代码资料的基础上，成功地完整复现了文中提出的算法，并对算法的代码实现方法进行了详细介绍，获得了代码的正确输出结果。

对于任务三，本组了解并复现了文中作为对照的一些其他的算法，并对选定论文算法在从混合型变量的纯粹的观测数据中发现因果关系的优势进行了讨论，分析了选定论文的先进性。

最后，本组讨论了选定论文中的算法在实际生产生活中发挥的重要作用，并讨论了我们在课程作业中的心得体会、收获和不足。

总体而言，我们圆满地完成了课程任务，并在完成作业的过程中，有了很多新的收获和启发；这次作业经历对于我们今后的学习与科研生活，是一次启发式的帮助！

Code Reproduction of Selected Paper

Abstract

This assignment mainly solves the code reproduction task of the selected paper, and introduces our implementation methods and experience.

Finding the potential causality between multiple variables is beneficial in many applications. The causality identification experiment is usually expensive and time-consuming. Therefore, it is necessary to find causality from pure observation data, which is usually called causal discovery or causal structure learning. In the selected papers of this group, an algorithm for learning causal structure in observation mixed data is proposed. This algorithm has obvious advantages in discovering the causal relationship of mixed data.

This group (Group 05) first deeply analyzed the frame structure of the selected paper (Selected Label 14), carefully read the content of the article and the designed algorithm, combed the text structure of the selected paper and drew the frame diagram of the article. Then, we divide the course tasks into three main sub tasks: "feasibility demonstration of mixed structural equation model and algorithm", "complete code reproduction of mixed causal structure learning algorithm of mixed data" and "advantage analysis of algorithm model and other methods in this paper"; In addition, we describe the practical application of the algorithm and the experience in the course assignment.

For task 1, on the basis of understanding the content of the article and combining with references, this group explained and proved the formal proof of identifiability of mixed SEM, the local consistency of cvmic score and the random conditional independence test of mixed data.

For task 2, on the basis of reading a large number of literature and code materials, this group successfully reproduced the algorithm proposed in this paper, introduced the code implementation method of the algorithm in detail, and obtained the correct output result of the code.

For task 3, the group understands and replicates some other algorithms in the paper as a comparison, discusses the advantages of the selected algorithm in finding causality from the pure observation data of mixed variables, and analyzes the progressiveness of the selected paper.

Finally, this group discussed the important role of the algorithm in the selected paper in the actual production and life, and discussed our experience, harvest and deficiency in the course homework.

On the whole, we have successfully completed the course tasks and gained a lot of new gains and inspiration in the process of completing our homework; This homework experience is a heuristic help for our future study and scientific research life!

目 录

[第一部分 选定论文结构分析 1](#_Toc46432383)

[1.1 选定论文的结构概述](#_Toc46432385) 1

[1.2 介绍](#_Toc46432384) 1

[1.3 混合结构方程模型](#_Toc46432385) 2

[1.4 混合结构学习算法 3](#_Toc46432385)

[1.5 实验结果 3](#_Toc46432385)

[1.6 结论 4](#_Toc46432385)

[第二部分 论文算法的可行性证明](#_Toc46432388) 5

[2.1 两个分类变量的可识别性](#_Toc46432389) 5

[2.2 一个分类变量和一个连续变量的可识别性](#_Toc46432393) 6

[2.3 CVMIC分数的局部一致性](#_Toc46432396) 6

[2.4 MRCIT进行随机条件独立性检验](#_Toc46432399) 7

[第三部分 论文算法的代码复现](#_Toc46432405) 9

[3.1 论文算法的实现步骤概述](#_Toc46432406) 9

[3.1.1 论文算法概述](#_Toc46432407) 9

[3.1.2 框架学习](#_Toc46432407) 9

[3.1.3 因果DAG结构学习](#_Toc46432407) 9

[3.1.4 剪枝 1](#_Toc46432407)0

[3.2 代码复现总体介绍 1](#_Toc46432406)0

[3.3 代码复现分模块介绍 1](#_Toc46432409)1

[3.3.1 run.py代码块 1](#_Toc46432407)1

[3.3.1.1 代码作用 1](#_Toc46432407)1

[3.3.1.2 源代码 1](#_Toc46432407)2

[3.3.1.3 算法相关概念介绍 1](#_Toc46432407)4

[3.3.2 const.py代码块 1](#_Toc46432407)5

[3.3.2.1 代码作用 1](#_Toc46432407)5

[3.3.2.2 源代码 1](#_Toc46432407)5

[3.3.2.3 算法相关概念介绍 1](#_Toc46432407)6

[3.3.3 utils.py代码块 1](#_Toc46432407)7

[3.3.3.1 代码作用 1](#_Toc46432407)7

[3.3.3.2 源代码 1](#_Toc46432407)7

[3.3.3.3 算法相关概念介绍](#_Toc46432407) 22

[3.3.4 camsearch\_prior.py代码块](#_Toc46432407) 22

[3.3.4.1 代码作用](#_Toc46432407) 22

[3.3.4.2 源代码](#_Toc46432407) 22

[3.3.4.3 算法相关概念介绍](#_Toc46432407) 29

[3.3.5 RCIT.py代码块](#_Toc46432407) 30

[3.3.5.1 代码作用](#_Toc46432407) 30

[3.3.5.2 源代码](#_Toc46432407) 30

[3.3.5.3 算法相关概念介绍](#_Toc46432407) 36

[3.3.6 gam\_cvmodel.py代码块](#_Toc46432407) 36

[3.3.6.1 代码作用](#_Toc46432407) 36

[3.3.6.2 源代码](#_Toc46432407) 37

[3.3.6.3 算法相关概念介绍](#_Toc46432407) 41

[3.3.7 lgbm\_cvmodel.py代码块](#_Toc46432407) 42

[3.3.7.1 代码作用](#_Toc46432407) 42

[3.3.7.2 源代码](#_Toc46432407) 42

[3.3.7.3 算法相关概念介绍](#_Toc46432407) 47

[3.3.8 mixed\_causal.py代码块](#_Toc46432407) 48

[3.3.8.1 代码作用](#_Toc46432407) 48

[3.3.8.2 源代码](#_Toc46432407) 48

[3.3.8.3 算法相关概念介绍](#_Toc46432407) 54

[3.3.9 skelprune.py代码块](#_Toc46432407) 54

[3.3.9.1 代码作用](#_Toc46432407) 54

[3.3.9.2 源代码](#_Toc46432407) 54

[3.3.9.3 算法相关概念介绍](#_Toc46432407) 57

[3.4 代码复现环境配置和必要的包](#_Toc46432409) 57

[3.5 代码复现结果展示](#_Toc46432409) 58

[3.5.1 原始输入数据](#_Toc46432407) 61

[3.5.2 算法第一阶段（框架学习）输出数据](#_Toc46432407) 63

[3.5.3 算法第二阶段（因果DAG结构学习）输出数据](#_Toc46432407) 64

[3.5.4 算法第三阶段（剪枝）输出数据](#_Toc46432407) 65

[3.5.5 运行时间与关键参数](#_Toc46432407) 66

[3.6 团队代码与文档仓库](#_Toc46432409) 66

[第四部分 其他相关算法的分析与探讨](#_Toc46432405) 67

[第五部分 论文算法的实际应用与意义](#_Toc46432405) 69

[第六部分 作业总结与心得体会](#_Toc46432405) 70

[致谢](#_Toc46432405) 71

[参考文献](#_Toc46432405) 71

[附录](#_Toc46432405) 72

第一部分 选定论文结构分析

**1.1 选定论文的结构概述**

通过对论文的仔细阅读和分析，我们梳理的文章的行文思路和结构；结合文章的标号，将结构划分为“介绍”、“混合方程结构模型”、“混合结构学习算法”、“实验结果”、“结论”五个主要部分；在理解文章内容的基础之上，依照文章结构，绘制了论文框架结构图。

**1.2 介绍**

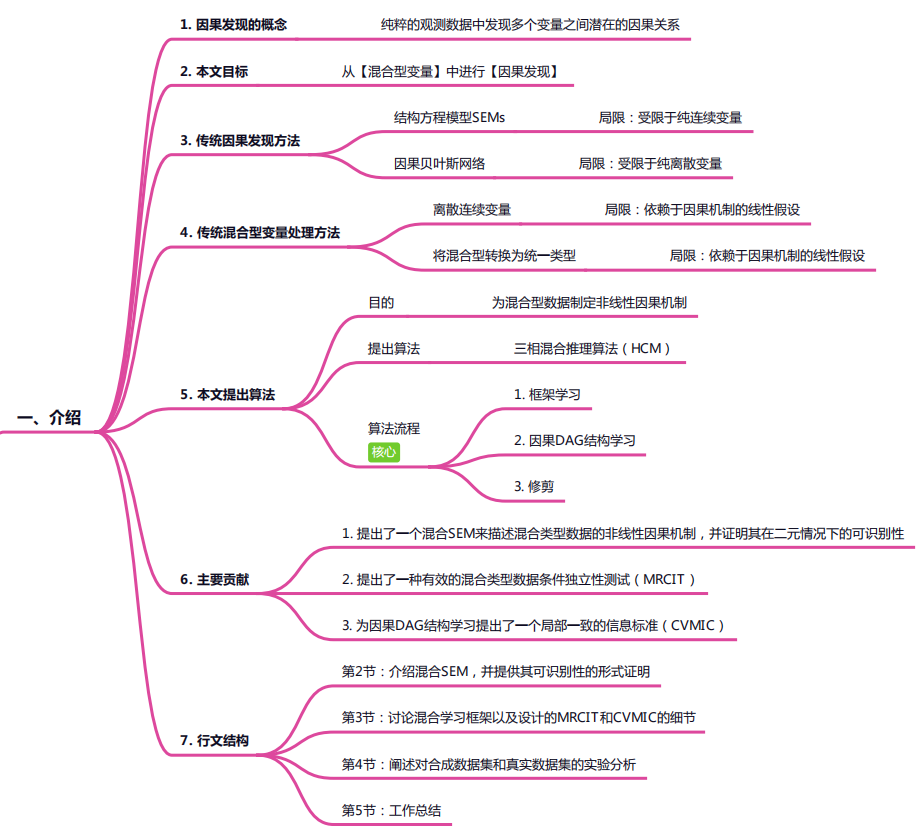
在第一部分“介绍”中，作者首先为我们介绍了因果发现的概念，它是指从纯粹的观测数据中发现因果关系，这在生产生活的许多方面都是有价值的。然后，作者介绍了研究目标，即从混合型变量中进行因果发现；其中的混合型变量，就是指离散与连续两种不同类型的变量。

接着，作者先后介绍了传统的因果发现方法和传统的混合型变量处理方法。对于传统因果发现方法，主要有因果结构方程模型（Structural Equation Models）、因果贝叶斯网络（Causal Bayesian Network）两大类。然而，结构方程模型受限于纯连续变量的因果发现，因果贝叶斯网络受限于纯离散变量的因果发现，也就是说，这些传统的方法，都是为纯离散或连续变量设计的。对于传统的混合型变量处理方法，作者介绍了离散连续变量（Discretize Continuous Variables）、将混合型变量的条件分布转换为统一类型（Convert the Conditional Distributions of Mixed-type Variables into the Same Type），然而，这些方法都依赖于因果机制的线性假设，使得他们同样具有局限性。

于是，作者介绍了他们所基础的算法：三相混合推理算法（HCM），并介绍了其目的，是为混合型数据制定非线性的因果机制。正是这样的优势，使得他们的算法突破了传统算法的局限性，具有很好的研究价值。作者简单介绍了算法的实现流程，分为框架学习、因果DAG结构学习、剪枝三个环节，并简单阐述了是三个环节的各自的功能和实现方法。

本部分的最后，作者介绍了他们工作的主要贡献，即：为混合型数据制定了非线性因果机制的描述方法，证明了其在二元情况下的可识别性；提出了MRCIT独立性测试；提出了CVMIC作为局部一致的信息标准。除此之外，作者介绍了文章其余部分的行文结构，依次为介绍混合SEM及其可识别性的证明、讨论MRCIT和CVMIC的细节、对合成数据集和真实数据集实验分析的阐述、以及作者的工作总结。

对于这一部分，我们绘制了行文框架结构图，如下图所示：



**1.3 混合结构方程模型**

在第二部分“混合结构方程模型”中，作者首先为我们介绍了混合SEM，即编码混合类型数据的因果机制模型，对于因果模型，基本要求是可识别性。

因此，作者又提供了混合SEM可识别性的形式证明。首先介绍了研究的重点是在双变量情况下的可识别性。对于双变量（二元混合结构方程），存在三种不同情形：

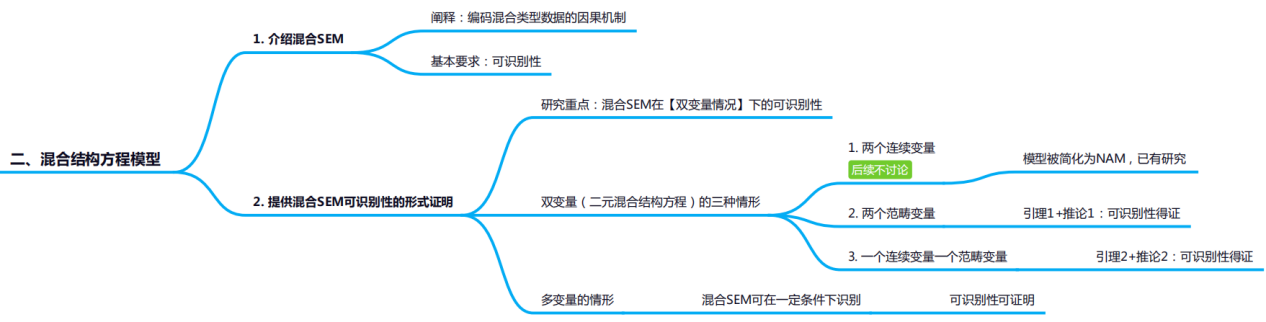
情形1. 两个连续变量。此时，模型被简化为NAM，已经有了研究，因此后续不再讨论；

情形2. 两个分类变量（范畴变量）。作者通过引理1+推论1，证明了其可识别性；

情形3. 一个连续变量和一个分类变量。作者通过引理2+推论2，证明了其可识别性。

最后，作者又介绍了他们提出的混合SEM，在多变量的情况下仍然可以在一定条件下识别，并说明了其可识别性是可被证明的。

对于这一部分，我们绘制了行文框架结构图，如下图所示：



**1.4 混合结构学习算法**

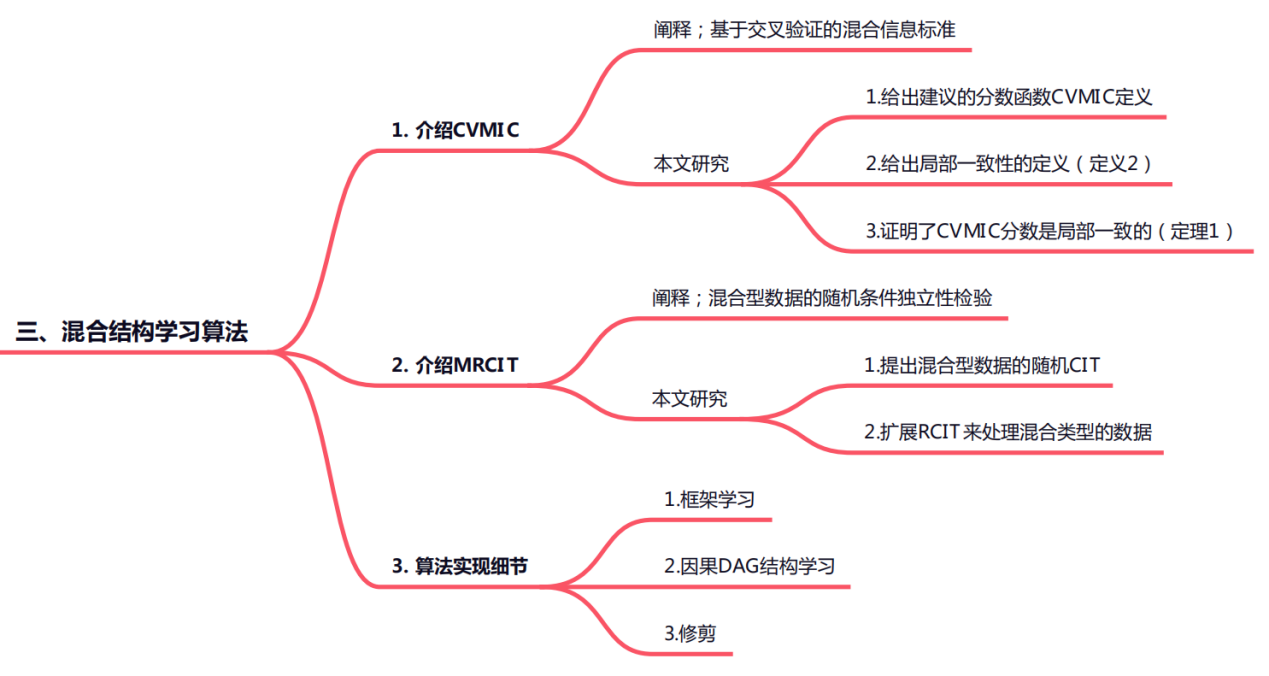
在第三部分“混合结构学习算法”中，作者首先介绍了CVMIC，即基于交叉验证的混合信息标准。对于n个随机混合型变量，给出了建议的分数函数CVMIC定义；然后，通过定义2，给出了局部一致性的定义；通过定理1，证明了CVMIC是局部一致的。

定义2和定理1的结合，对作者提出的CVMIC分数的合理性进行了理论层面的评估。

然后，作者介绍了MRCIT，它是混合型数据的随机条件独立性检验。作者在已有研究的基础上，通过扩展RCIT来处理混合类型的数据。并介绍了MRCIT的详细处理数据方法。

最后，作者介绍了文章算法的实现细节。这是文章的关键所在，也是小组作业中代码复现的核心内容。该算法通过CVMIC、MRCIT等方法的运用，先后进行了框架学习、因果DAG结构学习、剪枝三个环节，并介绍了这三个环节的实现方法。

对于这一部分，我们绘制了行文框架结构图，如下图所示：



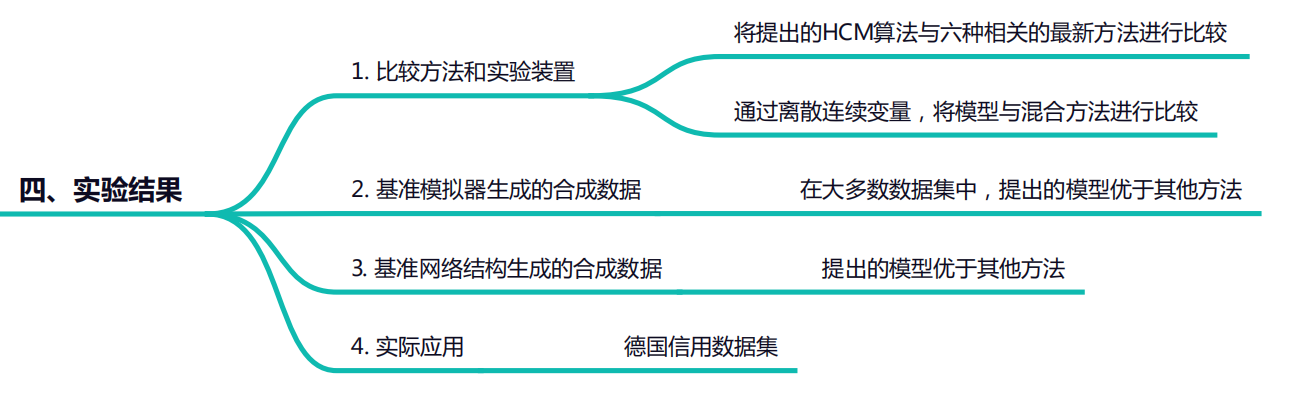
**1.5 实验结果**

在第四部分“实验结果”中，作者将我们提出的HCM算法与六种相关的最新方法进行了比较，这些算法都能够对混合类型数据进行因果发现；作者还通过离散连续变量的方法，将模型与混合方法进行了比较。

由于混合型变量的真实数据集很难找到，作者通过基准模拟器生成了混合型合成数据，在大多数合成数据集中，发现了作者提出的模型优于其他方法（F1分数越高，SHD越低，表现越好）。作者还通过基准网络结构生成了混合型合成数据，为每个网络生成12个混合类型数据集，并给出了平均F1分数和标准化结构汉明距离（N-SHD），以评估每种因果发现方法的性能（F1分数越高，N-SHD越低，表现越好），发现了作者提出的模型优于其他方法。

除此之外，作者还介绍了他们提出的模型在生产生活中的实际应用，在德国信用数据集上的应用，发挥了很好的实际价值。

对于这一部分，我们绘制了行文框架结构图，如下图所示：



**1.6 结论**

在第五部分“结论”中，作者对文章进行了简要的总结：

·他们提出了一种混合算法，称为HCM，用于混合类型数据的因果结构学习，即具有连续变量和范畴变量的数据集；

·提出了一个新的分数CVMIC用于精确因果DAG学习，以及一个新的条件独立性测试MRCIT用于混合型数据；

·还从理论上分析了该模型的可辨识性和局部一致性。

最后，作者介绍了未来的研究工作，即提高算法计算效率、算法对于潜在因果图中未观察到的混杂因素情况的拓展等。

对于这一部分，我们绘制了行文框架结构图，如下图所示：



第三部分 论文算法的代码复现

**3.1 论文算法的实现步骤概述**

**3.1.1 论文算法概述**

在我们选定的论文中，作者提出了一种名为混合SEM的算法。这种算法，混合型数据制定了非线性因果机制，其中产生离散变量的因果机制被建模为分类，而产生连续变量的因果机制被制定为非线性加性噪声模型（NAMs）。与此同时，作者提出了一个广义评分函数，即基于交叉验证的混合信息标准（CVMIC），用于评估候选混合SEM的质量。为了有效地从组合DAG空间中搜索最佳混合SEM，作者开发了一种三相混合推理算法。

概括来说，作者提出的算法（SEM）主要包括三个关键步骤：

第一步：框架学习。即没有方向的边，以减少搜索空间，通过将PC稳定算法与作者新提出的处理混合类型数据的CIT方法（称为混合类型随机化CIT，MRCIT）相结合，学习骨架结构。

第二步：因果DAG结构学习。在骨架结构的约束下，对DAG结构学习应用贪婪搜索过程，该过程从一个空DAG开始，贪婪地添加与CVMIC得分中最大增益对应的边。

第三步：剪枝。MRCIT再次用于使用相对较大的条件集来修剪学习到的因果结构，以减少误报。该算法被命名为“HCM”，缩写为“混合类型数据上的混合因果发现”。

这三步在作者论文的流程图示中有所展示。

其中，在混合方法的框架中，对于第一阶段，A~是框架学习阶段的输出，即表示无向图邻接矩阵的对称矩阵；对于第二阶段，Aˆ是该阶段的输出，即DAG的邻接矩阵。

**3.1.2 框架学习**

在因果马尔可夫假设和因果忠实性假设下，作者采用PCstable，它迭代地扩大条件集的大小，如果节点之间的对应变量通过MRCIT（条件）独立，则丢弃节点之间的边。为了加速算法，在每次迭代中并行测试MRCIT，并且在实现中将条件集的最大大小设置为1。作者不建议使用更大的条件集，因为文中的算法希望在丢弃可能导致误判的边缘之前确定因果关系，如果条件集中也考虑了同子项。

**3.1.3 因果DAG结构学习**

由于离散变量的值空间有限，分类模型的估计对数似然通常大于回归模型的估计对数似然。因此，对于标准搜索算法，如爬山，添加得分最高的边可能会导致系统性选择偏差，即选择指向离散变量的边的偏好。为了缓解这种情况，作者引入了一个贪婪过程，该过程从一个空DAG开始，并在每次迭代中添加对应于得分最大增益的边。离散变量的经验概率通过频率计算，而连续变量的经验概率通过核密度估计（KDE）估计。为了保证无环，有向路径矩阵将相应地更新。只有当第j个节点的父节点集更新且因此有效时，才会更新第j个节点的得分和添加潜在边的增益。对于连续变量，CVMIC评分中的可能性也通过KDE进行估计，以避免假设噪声分布。证明了在似然计算中使用KDE的合理性。在这项工作中，作者使用LightGBM对因果机制进行非参数估计。此外，框架学习后搜索空间减小，该算法适用于多达数百个节点。

**3.1.4 剪枝**

在前一阶段，只要不打破DAG约束，就会贪婪地添加边。此设置帮助我们避免选择难以确定的阈值来确定添加边缘的最低分数增益。然而，此设置倾向于添加更多“多余”边，需要进行修剪以提高推理精度。剪枝是通过MRCITs进行的，以测试每个父子对条件作用对所有其他直接原因的独立性，如果相应的父子是条件独立的，则删除“多余”边。

**3.2 代码复现总体介绍**

我们的代码复现，共有16个关键文件；其中13个为原始文件，3个为代码的运行结果输出文件。

包括：

1. 参数文件：requirements.txt

对代码需要导入的包的版本参数进行了说明；

2. 数据文件：alarm.csv

含有表示实际图（true graph）的数据；

3. 数据文件：alram\_simulate.csv

含有需要读取的仿真数据；

4. 程序文件：run.py

主程序，读取文件、设置模型参数、调用各相关功能代码段、输出结果；

5. 程序文件：const.py

设置模型的默认参数、设置并检查Lightgbm、Pygam模型的默认参数；

6. 程序文件：utils.py

包含通用的数据处理与评估函数；

7. 程序文件：camsearch\_prior.py

贪心地添加边、计算每个变量的对数似然、对ScoreMat进行计算；

8. 程序文件：RCIT.py

实现骨架学习的混合类型随机化CIT，包括随机傅立叶特征（RFF）混合、分类变量的内核岭回归、连续变量的内核岭回归、混合类型数据独立性条件测试；

9. 程序文件：gam\_cvmodel.py

建立逻辑回归、线性回归模型，并实现模型封装；

10. 程序文件：lgbm\_cvmodel.py

实现lgb模型参数配置、模型训练并得出预测结果、实现Lightgbm基于决策树算法的梯度提升算法的模型封装；

11. 程序文件：mixed\_causal.py

进行先验知识编码、数据处理，通过调用代码段，先后完成了文中算法的所有三个关键环节，分别评估三个结果的召回率、精确率等数据，并返回评估结果；

12. 程序文件：skelprune.py

具体实现框架学习PC-stable算法、剪枝算法；

13. 模拟程序文件：ConditionalGaussianSimulation.java

进行高斯条件数据模拟；

14. 输出结果数据文件：alram\_simulate\_skl.csv

框架学习阶段（第一环节）的输出，表示无向图邻接矩阵的对称矩阵；

15. 输出结果数据文件：alram\_simulate\_dag2.csv

因果DAG结构学习（第二环节）的输出，表示DAG的邻接矩阵；

16. 输出结果数据文件：alram\_simulate\_dag.csv

剪枝（第三环节）后的结果。

代码复现的输出，除了上述三个文件外，还包含控制台的输出，计算了算法的运行时间、以及TP、TN、FP、FN、SHD等关键参数。

**3.3 代码复现分模块介绍**

**3.3.1 run.py代码块**

**3.3.1.1 代码作用**

本模块为代码的主程序。

主要包括：

(1)参数代码段：args = easydict.EasyDict；

(2)函数代码段：def check\_model\_para()；

(3)主函数代码段：main。

在(1)中，设置了数据文件的位置和名字、实际图（true graph）文件的位置和名字的读取参数；并可以用于测试步骤2在不同设置下的性能；即：如果已经有了骨架文件"alaram\_simulate\_skl.csv"，可以实现避免步骤1的再次运行。

在(2)中，对模型的参数及默认参数进行了设置，并且实现了对lgbm\_cvmodel、gam\_cvmodel模型的参数导入，以及在两个模型均不符合时抛出异常。

在(3)中，作为主函数，实现了对表格文件（数据）的读取、对骨架文件的读取，导入了模型及其参数，调用了相关功能代码段，依次进行了数据的预处理、模型间特征关系的处理、根据文中算法得到关系的有向图、计算运行时间，并保存计算结果矩阵，最终将结果与真实值进行评估，返回损失率、精确率、TP、TN、FP、FN等参数；这些参数在后面的模块中进行了详细介绍。

**3.3.1.2 源代码**

|  |
| --- |
| # coding: utf-8  import pandas as pd  import numpy as np  from mixed\_causal import mixed\_causal, \      prior\_knowledge\_encode, data\_processing, evaluate  import easydict  from const import DEFAULT\_MODEL\_PARA, DEFAULT\_LGBM\_PARA, DEFAULT\_GAM\_PARA  args = easydict.EasyDict({      'data\_file': 'alram\_simulate.csv',  # 数据文件的位置和名字      'cat\_index': ['2','3','4','7','8','10','11','13','16','17','18','19','21','22','27','33','36','37'],      'true\_G':'alarm.csv',  # 实际图（true graph）文件的位置和名字      'model\_para': {'step1\_maxr': 1, 'step3\_maxr': 3, 'num\_f': 100,                    'num\_f2': 20, 'indep\_pvalue': 0.05, 'downsampling': False,                    'cv\_split': 5, 'll\_type': 'local', 'alpha': 0.0,                    'maxNumParents': 10, 'score\_type': 'bic'                    },      # 可用于测试步骤 2 在不同设置下的性能；即:      # 如果已经有了骨架文件"alaram\_simulate\_skl.csv"      # 然后可以使用它来避免再次运行步骤 1      'skl\_file': "",      'base\_model':'lgbm',  # 'lgbm' or 'gam'      'base\_model\_para': {},      'source\_nodes': [],      'direct\_edges': {},      'not\_direct\_edges': {},      'happen\_before': {},  })  def check\_model\_para(model\_para, base\_model, base\_model\_para):      model\_para\_out = {}      for para in DEFAULT\_MODEL\_PARA:  # 自己设置的参数          if para in model\_para:              model\_para\_out[para] = model\_para[para]          else:                        # 使用默认参数              model\_para\_out[para] = DEFAULT\_MODEL\_PARA[para]      base\_model\_para\_out = {}      base\_model\_para\_out['base\_model'] = base\_model      if base\_model=='lgbm':   # 如果用模型 lgbm\_cvmodel 的话导入该模型参数，可以自己设置，可以默认          for para in DEFAULT\_LGBM\_PARA:              if para in base\_model\_para:                  base\_model\_para\_out[para] = base\_model\_para[para]              else:                  base\_model\_para\_out[para] = DEFAULT\_LGBM\_PARA[para]      elif base\_model=='gam':  # 如果用模型 gam\_cvmodel 的话导入该模型参数，可以自己设置，可以默认          for para in DEFAULT\_GAM\_PARA:              if para in base\_model\_para:                  base\_model\_para\_out[para] = base\_model\_para[para]              else:                  base\_model\_para\_out[para] = DEFAULT\_GAM\_PARA[para]      else:  # 两个都不是抛出异常          raise NotImplementedError(              f"currently we only support 'lgbm' and 'gam'.")      return model\_para\_out, base\_model\_para\_out  if \_\_name\_\_ == '\_\_main\_\_':      # df = pd.read\_csv("C:\\Users\\lenovo\\Desktop\\AAAI2022-HCM-main\\alram\_simulate.csv") # 读取表格文件      df = pd.read\_csv(args.data\_file) # 读取表格文件      if args.skl\_file == "":          # 如果没有骨架文件，就不用          selMat = None      else:                            # 如果有骨架文件，读入          selMat = pd.read\_csv(args.skl\_file, header=None).values > 0      # csv(comma-separated values),hearder=None 取消表头      # 读取 csv 文件      print(df.columns)      print(df.columns.values)      # 导入模型及其参数      model\_para\_out, base\_model\_para\_out = check\_model\_para(          args.model\_para, args.base\_model, args.base\_model\_para)      # 数据预处理，也就是编码，根据需要可对数据进行双权'biweight'和标准'standard'编码      df, X\_encode = data\_processing(df, args.cat\_index, normalize='biweight')      # 根据自己填入的参数告诉模型哪些特征是有关系的，可以是单向的，也可以是双向的，返回邻接矩阵      prior\_adj, prior\_anc = prior\_knowledge\_encode(          feature\_names=df.columns, source\_nodes=args.source\_nodes,          direct\_edges=args.direct\_edges, not\_direct\_edges=args.not\_direct\_edges)      # 运用文中算法得到关系的有向图和运行时间      selMat, dag2, dag, step1\_time, step2\_time, step3\_time = mixed\_causal(          df, X\_encode, model\_para= model\_para\_out,          prior\_adj=prior\_adj, prior\_anc=prior\_anc,          base\_model\_para=base\_model\_para\_out, selMat=selMat)      print(step1\_time, step2\_time, step3\_time)      # 将获得的结果存起来，参数分别是文件名字，要保持的矩阵，分隔符      np.savetxt(args.data\_file[:-4]+'\_skl.csv', selMat, delimiter=",")      np.savetxt(args.data\_file[:-4]+'\_dag2.csv', dag2, delimiter=",")      np.savetxt(args.data\_file[:-4]+'\_dag.csv', dag, delimiter=",")      if args.true\_G != '':          trueG = pd.read\_csv(args.true\_G).values  # 获得表格的所有值          # 将结果与真实值进行评估，返回损失率，精确率，TP  TN  FP  FN 等          skl\_result, dag2\_result, dag\_result = evaluate(trueG, selMat, dag2,dag)          print(skl\_result)          print(dag2\_result)          print(dag\_result)          print(trueG)          print(selMat)          print(dag2)          print(dag) |

**3.3.1.3 算法相关概念介绍**

下面，列举了上述算法中关键参数指标的含义：

trueG 是真实值，estG是估计值

TP,TN,FP,FN中，T,F代表true,false；P,N代表postive,negative；

TP：被模型预测为正类的正样本（真阳）真实和预测都为 1（两个方向）；

TN：被模型预测为负类的负样本（真阴）真实和预测都为 0（两个方向）；

FP：被模型预测为正类的负样本（假阳）真实 0 预测 1（一个方向，另外一个方向TN）；

FN：被模型预测为负类的正样本（假阴）真实 1 预测 0（一个方向，另外一个方向TN）；

FD：预测错误的相反正样本，即 01 预测为 10，10 预测为 01（两个方向一个FP，另外一个FN）；

MD：01，10 都预测为 11（一个FP 一个TP）；

FPMD：00 预测为 11（两个FP）；

TP 与 TN 是预测正确的；

Precise：准确率 TP/(TP + FP + FD)；

Recall：召回率，TP/所有边  Recall 召回率，(TP + FN + FD)；

SHD：结构汉明距离，所有边预测和真实不同的个数(不论哪一个方向)/2。

**3.3.2 const.py代码块**

**3.3.2.1 代码作用**

在本模块中，主要包括：

(4)参数代码段：DEFAULT\_MODEL\_PARA；

(5)参数代码段：DEFAULT\_LGBM\_PARA；

(6)参数代码段：DEFAULT\_GAM\_PARA。

在(4)中，对模型的默认参数进行了设置，包括第一步、第三步的最大环境设置、条件集的随机fft特征数、测试变量的随机fft特征数、独立测试的阀值、交叉验证次数等；设置了默认的分数类型、是否需要在MRCIT中采样等参数。

在(5)中，对Lightgbm中默认参数进行了设置和检查，包括：提升类型、每个基学习器的最大叶子结点、特征分数/子特征处理列采样、梯度下降的步长与迭代次数。

在(6)中，对Pygam中默认参数进行了设置和检查，包括：样条函数、分层、个数以及使用edof标志的状态。

**3.3.2.2 源代码**

|  |
| --- |
| DEFAULT\_MODEL\_PARA = {      'step1\_maxr': 1,  # step 1 的最大环境设置      'step3\_maxr': 3,  # step 3 的最大环境设置      'num\_f': 100,     # 条件集的随机 fft 特征数      'num\_f2': 10,     # 测试变量的随机 fft 特征数      'indep\_pvalue': 0.05,  # 独立测试的阀值      'alpha': 0.0,      'll\_type': 'local',    # koe估算kde: 'local' 或者 'global'      'cv\_split': 5,         # 交叉验证次数（可以为 0 次）      'downsampling': False, # 是否需要在 MRCIT 中进行下采样      'maxNumParents': 10,   # NumParents 最大值      'score\_type': 'bic'    # 分数类型: 'bic'、'll'、'aic'；LH: 对数似然  }  # 检查 Lightgbm 中的参数，Lightgbm 基于决策树算法的梯度提升算法(lgb)  DEFAULT\_LGBM\_PARA = {      'boosting': 'gbdt',        # 提升类型：传统的梯度提升决策树——决策树类型      'num\_leaves': 31,          # 每个基学习器的最大叶子节点      'feature\_fraction': 0.5,   # 特征分数/子特征处理列采样      'learning\_rate': 0.05,     # 梯度下降的步长。常用 0.1, 0.001, 0.003      'num\_boost\_round': 200,    # 迭代次数  }  # 检查 Pygam 中的参数  DEFAULT\_GAM\_PARA = {      "spline\_order": 10,  # 样条函数      "lam": 0.6,          # 分层      "n\_jobs": 1,         # 个数      "use\_edof": True,    # 使用edof标志  } |

**3.3.2.3 算法相关概念介绍**

GBDT (Gradient Boosting Decision Tree) ：机器学习中⼀个长盛不衰的模型，其主要思想是利用弱分类器（决策树）迭代训练以得到最优模型，该模型具有训练效果好、不易过拟合等优点。GBDT 在⼯业界应用⼴泛，通常被⽤于点击率预测，搜索排序等任务。

LightGBM：⼀个实现 GBDT 算法的框架，是快速的、分布式的、高性能的基于决策树算法的梯度提升框架。可用于排序，分类，回归以及很多其他的机器学习任务中。它具有以下优点：

更快的训练速度、更低的内存消耗、更好的准确率、分布式支持，可以快速处理海量数据。

**3.3.3 utils.py代码块**

**3.3.3.1 代码作用**

本模块均为通用的数据处理与评估函数。

在本模块中，主要包括：

(7)函数代码段：def normalize\_biweight；

(8)函数代码段：def normalize；

(9)函数代码段：def data\_preprocess；

(10)函数代码段：def evaluate\_binary；

(11)函数代码段：def skeleton\_metrics；

(12)函数代码段：def check\_connect\_skel；

(13)函数代码段：def reachable。

在(7)中，进行了非线性转化，返回x的归一化双权值。采用的方法是使用双权标度，在方差较大的情况下合理估计标准化；如果分布的标准差太小，则用标准化的标准差代替。

在(8)中，实现了将x中的数据标准差标准化。先标准化标准差，再用lambda表达式代替函数，将x中的数据标准差标准化。

在(9)中，实现了数据处理，将df中的数据进行处理，离散值转换成独热码，连续值不变。本函数中的若干参数各自的含义为：dataframe 是带行列名的二维矩阵；iloc可以通过位置索引访问df元素；loc可以通过列名，行名访问df元素；category是一种类型。

在(10)中，对图G中的预测结果进行了评估，边的两个方向一起被计算，trueG中某一条边不会出现两个方向都有。作用是计算准确率、召回率等，与下面的函数不同的是，这应是一种二分评估。

在(11)中，对图G中的预测结果进行评估，计算被预测正确或错误的正负样本。跟上一个函数很相似，但是它不区分方向，相当于把G看成无向图；即：这里只要[i,j]或[j,i]满足一个就可以。

在(12)中，在i的范围内检查了j是否已经存在，skeleton是一个邻接矩阵；即：检查ij两个结点在skeleton图上是否是联通的（不考虑边的方向）。

在(13)中，判断并返回是否存在一个路径从fr到to（考虑边的方向）。

**3.3.3.2 源代码**

|  |
| --- |
| ### 本文件：通用的数据处理与评估函数  import numpy as np  from sklearn.preprocessing import OneHotEncoder  from astropy import stats  # 非线性转化，返回 x 的归一化双权值（转换器用的函数）  # 采用的方法是使用双权标度，在方差较大的情况下合理估计标准化；如果分布的标准差太小，则用标准化的标准差代替  def normalize\_biweight(x, eps=1e-10):      median = np.median(x)                      # 计算分布的中位数      scale = stats.biweight.biweight\_scale(x)   # 返回分布的双权重比例。双权标度是确定分布标准差的稳健统计      if np.std(x) < 1e+2 or np.isnan(scale) or scale < 1e-4:  # np.std() 是得到 x 标准差, np.isnan() 判断是否非数字          norm =  (x-np.mean(x))/np.std(x)       # 标准化（太小或者非数字时就用个体标准差/样本标准差代替双权值）      else:          norm = (x - median) / (scale + eps)    # 否则，返回（x-中位数）/（双权标准差+eps）      return norm  # 将 x 中的数据标准差标准化  def normalize(x):      norm = lambda x: (x-np.mean(x))/np.std(x)  # 标准化标准差      return np.apply\_along\_axis(norm, 0, x)     # 用 lambda 表达式代替函数，将 x 中的数据标准差标准化  # 数据处理，将 df 中的数据进行处理，离散值转换成独热码，连续值不变  # 参数:  #   dataframe 是带行列名的二维矩阵，iloc 可以通过 0，1，2 的索引访问 df 元素  #   iloc 可以通过位置索引访问 df 元素  #   loc 可以通过列名，行名访问 df 元素  #   'category'是一种类型，'O'是 o 不是 0，表示 object  def data\_preprocess(df):      def \_encoding(i):          if df.iloc[:,i].dtype == 'O' or df.iloc[:, i].dtype.name == 'category':  # 每行第 i 个输入数据是分类有关的              tempX = df.iloc[:, i].values.reshape(-1, 1)   # 把数据变成只有 1 列的，行自动计算（reshape 中的 -1 表示该维度的大小自动计算）              enc = OneHotEncoder(handle\_unknown='ignore')  # 创建一个对象，把每一行数据编成独热码，参数表示遇到没标识过的时不报错              enc.fit(tempX)                                # 加入训练数据（也就是告诉这个对象有哪些类别）              out = enc.transform(tempX).toarray()          # 得到 tempX 的独热码          else:              out = df.iloc[:, i].values.reshape(-1, 1)     # 不是则直接变成 1 列，不编码          return out      p = df.shape[1]                                       # 得到 df 的列数 p      X\_encode = [\_encoding(i) for i in np.arange(p)]       # 每一列编码一次      return X\_encode                                       # 返回编码  # trueG 是真实值，estG是估计值  # 对图 G 中的预测结果进行评估，边的两个方向一起被计算，trueG 中某一条边不会出现两个方向都有  # 作用是计算准确率、召回率等，与下面的函数不同的是，这应是一种二分评估  def evaluate\_binary(trueG, estG):      # TP,TN,FP,FN 中 T,F 代表 true, false; P,N 代表 postive, negative      # TP：被模型预测为正类的正样本（真阳）真实和预测都为 1（两个方向）      # TN：被模型预测为负类的负样本（真阴）真实和预测都为 0（两个方向）      # FP：被模型预测为正类的负样本（假阳）真实 0 预测 1（一个方向，另外一个方向TN）      # FN：被模型预测为负类的正样本（假阴）真实 1 预测 0（一个方向，另外一个方向TN）      # FD：预测错误的相反正样本，即 01 预测为 10，10 预测为 01（两个方向一个FP，另外一个FN）      # MD：01，10 都预测为 11（一个FP 一个TP）      # FPMD：00 预测为 11（两个FP）      # TP 与 TN 是预测正确的      # precise 准确率 TP/(TP + FP + FD)      # recall 召回率，TP/所有边  recall1 召回率，(TP + FN + FD)      # SHD 结构汉明距离，所有边预测和真实不同的个数(不论哪一个方向)/2      TP, TN, FP, FN, FD, MD, FPMD = 0, 0, 0, 0, 0, 0, 0      n\_node = trueG.shape[0]    # 行数      for i in range(1, n\_node):          for j in range(i):              # trueG中 10 01 代表正样本，00 代表负样本              if trueG[i, j] == 1 and trueG[j, i] == 0 and estG[i, j] == 1 and \                      estG[j, i] == 0:                  TP += 1              if trueG[i, j] == 0 and trueG[j, i] == 1 and estG[i, j] == 0 and \                      estG[j, i] == 1:                  TP += 1              if trueG[i, j] == 0 and trueG[j, i] == 0 and estG[i, j] == 0 and \                      estG[j, i] == 0:                  TN += 1              if trueG[i, j] == 0 and trueG[j, i] == 0 and estG[i, j] == 1 and \                      estG[j, i] == 0:                  FP += 1              if trueG[i, j] == 0 and trueG[j, i] == 0 and estG[i, j] == 0 and \                      estG[j, i] == 1:                  FP += 1              if trueG[i, j] == 1 and trueG[j, i] == 0 and estG[i, j] == 0 and \                      estG[j, i] == 0:                  FN += 1              if trueG[i, j] == 0 and trueG[j, i] == 1 and estG[i, j] == 0 and \                      estG[j, i] == 0:                  FN += 1              if trueG[i, j] == 1 and trueG[j, i] == 0 and estG[i, j] == 0 and \                      estG[j, i] == 1:                  FD += 1              if trueG[i, j] == 0 and trueG[j, i] == 1 and estG[i, j] == 1 and \                      estG[j, i] == 0:                  FD += 1              if trueG[i, j] == 0 and trueG[j, i] == 1 and estG[i, j] == 1 and \                      estG[j, i] == 1:                  MD += 1              if trueG[i, j] == 1 and trueG[j, i] == 0 and estG[i, j] == 1 and \                      estG[j, i] == 1:                  MD += 1              if trueG[i, j] == 0 and trueG[j, i] == 0 and estG[i, j] == 1 and \                      estG[j, i] == 1:                  FPMD += 1      if (TP + FP + FD)>0:          Precision = TP / (TP + FP + FD)  # 计算精确率      else:          Precision = 0.0      Recall = TP / sum(sum(trueG))        # 计算总召回率      if (TP + FN + FD) > 0:          Recall1 = TP / (TP + FN + FD)      else:          Recall1 = 0.0      SHD = sum(sum((trueG != estG) | np.transpose((trueG != estG)))) / 2 # transpose()是转置，也就是计算所有 trueG[i,j][j,i]!=estG[i,j][j,i] 的和      return {'TP': TP, 'TN': TN, 'FP': FP, 'FN': FN, 'FD': FD, 'MD': MD,              'FPMD': FPMD,'Precision': Precision, 'Recall': Recall,              'Recall\_NOMD': Recall1, 'SHD': SHD}  # 图 G 中的预测结果进行评估，计算被预测正确或错误的正负样本  # 跟上一个函数很相似，但是它不区分方向，相当于把 G 看成无向图；即：这里只要 [i,j] 或 [j,i] 满足一个就可以  def skeleton\_metrics(trueG, estG):      # TP,TN,FP,FN 中 T,F 代表 true, false; P,N 代表 postive, negative      #TP：被模型预测为正类的正样本      #TN：被模型预测为负类的负样本      #FP：被模型预测为正类的负样本      #FN：被模型预测为负类的正样本      TP,TN,FP,FN = 0,0,0,0      n = trueG.shape[0]      for i in range(n):          for j in range(i):              # 按照定义算              if trueG[i, j] == 1 or trueG[j, i] == 1:                  if estG[i, j] != 0 or estG[j, i] != 0:                      TP += 1                  else:                      FN += 1              else:                  if estG[i, j] == 0 and estG[j, i] == 0:                      TN += 1                  else:                      FP += 1      return {'TP': TP, 'TN': TN, 'FP': FP, 'FN': FN}  # 在 i 的范围内检查 j 是否已经存在，skeleton 是一个邻接矩阵  # 即：检查 ij 两个结点在 skeleton 图上是否是联通的（不考虑边的方向）  def check\_connect\_skel(skeleton, i, j):      depth = set.union(set(np.where(skeleton[:,i]==1)[0]),    # 将第 i 行或第 i 列中为 1 的行索引索引加入集合                        set(np.where(skeleton[i,:]==1)[0]))      checked = depth      while depth:          if j in depth:              return True          next = {}          for k in depth:              next = set.union(next, set.union(set(np.where(skeleton[:,k]==1)[0]),                                              set(np.where(skeleton[k,:]==1)[0])))          depth = set.difference(next, checked)    # 获得有差异的集合，即还没被检查过的          checked = set.union(checked, depth)      # 更新已经检查过的      return False  # 返回是否存在一个路径从 fr 到 to (考虑边的方向)  def reachable(dag, fr, to):      depth = set(np.where(dag[fr,:]==1)[0])      checked = depth      while depth:          if to in depth:              return True          next = set()          for k in depth:              next = set.union(next, set(np.where(dag[k,:]==1)[0]))          depth = set.difference(next, checked)          checked = set.union(checked, depth)      return False |

**3.3.3.3 算法相关概念介绍**

离散值转换成独热码（在数据处理函数中）：独热编码即One-Hot编码，又称一位有效编码，其方法是使用N位状态寄存器来对N个状态进行编码，每一个状态都由他独立的寄存器位，而且在任意时候，其中只有一位有效。

例如：

天然状态码为：000,001,010,011,100,101

独热编码为：000001,000010,000100,001000,010000,100000

能够这样理解，对于每个特征，若是它有m个可能值，那么通过独热编码后，就变成了m个二元特征。而且，这些特征互斥，每次只有一个激活。所以，数据会变成稀疏的。

这样作的好处主要有：解决了分类器很差处理属性数据的问题，在必定程度上也起到了扩充特征的作用。

**3.3.4 camsearch\_prior.py代码块**

**3.3.4.1 代码作用**

在本模块中，主要包括：

(14)函数代码段：def greedy\_edgeadding；

(15)函数代码段：def compute\_init\_ll；

(16)类代码段：class ScoreMatCompute。

在(14)中，主要实现了贪心地添加边的功能。首先，在selMat中对先验知识进行编码，去除无向边；然后，填充对角线、计算分数，进行scoreMat初始化，并开始贪心地添加边。

在(15)中，实现了计算没有模型的每个变量的对数似然，它的返回值是每个变量的对数似然(log likelihood)。在考虑初始得分的基础上，离散的通过经验概率，连续的通过核密度估计。

在(16)中，对ScoreMat进行了一系列计算。计算步骤已经在代码段注释中说明。

**3.3.4.2 源代码**

|  |
| --- |
| import itertools  import numpy as np  import pandas as pd  from sklearn.neighbors import KernelDensity  from scipy.stats import gaussian\_kde  import warnings  warnings.filterwarnings("ignore")  def greedy\_edgeadding(df, X\_encode, selMat, maxNumParents,base\_model\_para,                        alpha=0.0,                        cv\_split=5, ll\_type='local',                        prior\_adj=None, prior\_anc=None,                        score\_type='bic', debug=False):      # 首先在 selMat 中对先验知识进行编码，去除无向边      if prior\_adj is not None:          if prior\_adj.shape != selMat.shape:              raise ValueError("the shape of prior\_adj is not same as selMat")          selMat = selMat & (prior\_adj >= 0)          must\_have = np.argwhere(prior\_adj > 0)    # 返回 prior\_adj 中满足条件的位置索引      else: # 这是默认情况          must\_have = []      if prior\_anc is not None:          if prior\_anc.shape != selMat.shape:              raise ValueError("the shape of prior\_anc is not same as selMat")          selMat = selMat & (prior\_anc >= 0)          not\_prior\_anc = prior\_anc < 0          np.fill\_diagonal(not\_prior\_anc, False)    # 填充not\_prior\_anc 对角线      else:          not\_prior\_anc = np.zeros(selMat.shape, dtype=bool)      path = np.zeros(selMat.shape)      np.fill\_diagonal(path, 1)                     # 填充对角线      Adj = np.zeros(selMat.shape)      # 计算分数      ScoreMatComputer = ScoreMatCompute(          df, X\_encode, selMat,          maxNumParents=maxNumParents,          cv\_split=cv\_split,          ll\_type=ll\_type,          score\_type=score\_type,          debug=debug,          base\_model\_para=base\_model\_para)      # scoreMat 初始化      scoreMat, scoreNodes = ScoreMatComputer.initialScoreMat()      # 贪心地添加边      while np.max(scoreMat) > -float('inf'):          diff = scoreMat-np.transpose(scoreMat)          # 两个"-inf"的不同之处在于 set to -inf          diff[np.isnan(diff)] = -float('inf')          # "non -inf"和"-inf"的不同之处在于 set to 0 来避免错误          # 在上一步中没有对称纯化（none symmetric puring）时赋值          diff[np.isinf(diff)] = 0.0          weighted\_gain\_diff = (1 - alpha) \* scoreMat + alpha \* diff          # 如果有一些边必须加上          if len(must\_have) > 0:    # 如果有必须加的，就 add              mind = np.argmax([weighted\_gain\_diff[tuple(i)] for i in must\_have])   # argmax 找某个维度的最大值，每次加最大的              row\_index, col\_index = must\_have[mind][0], must\_have[mind][1]              must\_have = np.delete(must\_have, mind, axis=0)          else:              # 找到最好的边              # np.unravel\_index 返回的是拉成一维之后的索引              # 这里找的是最大值在原来中的索引              row\_index, col\_index = np.unravel\_index(                  weighted\_gain\_diff.argmax(), weighted\_gain\_diff.shape)          # 需要考虑是否添加 (row\_index, col\_index)          # 将避免原因顺序 (cause order)          # 更新有向无环路径          t\_path = path.copy()          t\_path[row\_index, col\_index] = 1          DescOfNewChild = np.append(np.where(t\_path[col\_index,:]==1), col\_index)   # 父节点；添加边          AncOfNewParent = np.append(np.where(t\_path[:,row\_index]==1), row\_index)   # np.where 返回的是满足条件的坐标          # itertools.product 用来求笛卡尔积，这里用来组合出所有的元素位置          # 找到所有已经够存在路径的          for element in list(itertools.product(AncOfNewParent, DescOfNewChild)):              t\_path[element] = 1          # 如果有一些避免，那么改变不包括边缘和为"-inf"设置分数          if np.any(not\_prior\_anc & (t\_path == 1)):              # not\_prior\_anc 的情况下，如果已经存在路径了，删掉他              scoreMat[row\_index, col\_index] = -float('inf')              continue          else:              if debug:                  print(f"before add the edge ({row\_index, col\_index}), the "                        f"score of {col\_index} is {scoreNodes[col\_index]}")              scoreNodes[col\_index] = (scoreNodes[col\_index] +                                      scoreMat[row\_index, col\_index])              if debug:                  print(f"after the score is {scoreNodes[col\_index]}")                  print(scoreNodes)              ScoreMatComputer.set\_scoreNodes(scoreNodes)              scoreMat[row\_index, col\_index] = -float('inf')              scoreMat[col\_index, row\_index] = -float('inf')              Adj[row\_index, col\_index] = 1              path = t\_path.copy()              scoreMat[np.transpose(path) == 1] = -float('inf')              # 更新 scoreMat；算分              ScoreMatComputer.set\_scoreMat(scoreMat)              scoreMat, scoreNodes = ScoreMatComputer.scoreupdate(                  Adj=Adj, j=col\_index)      return Adj  # 计算没有模型的每个变量的对数似然  # param x\_col:pandas 的一个数据系列  # 返回：每个变量的对数似然(log likelihood)  def compute\_init\_ll(x\_col, bandwidth=1.0, kernel='gaussian'):      # 就是文献中初始得分的定义      if x\_col.dtypes == 'O' or x\_col.dtypes == 'bool':          # 离散的通过经验概率          prob\_dic = x\_col.value\_counts(normalize=True).to\_dict()   # value\_counts 统计出现频率          prob\_list = x\_col.replace(prob\_dic)          return np.mean(np.log(prob\_list))      else:          # 连续的通过核密度估计          data\_x = x\_col.values          kde = gaussian\_kde(data\_x)          logprob = np.log(kde.evaluate(data\_x))          """          kde = KernelDensity(bandwidth=bandwidth, kernel=kernel)          kde.fit(data\_x[:, None])          # score\_samples 返回的是 log of the probability density          logprob = kde.score\_samples(data\_x[:, None])          """          return np.mean(logprob)  class ScoreMatCompute(object):      def \_\_init\_\_(self, X, X\_encode, selMat, maxNumParents, base\_model\_para,                  cv\_split=5,                  ll\_type='local', score\_type='bic', debug=False):          # 初始的设置，复制          self.X = X          self.X\_encode = X\_encode          self.selMat = selMat          self.p = selMat.shape[0]          self.maxNumParents = maxNumParents          self.valid\_pair = np.argwhere(selMat)          self.scoreMat = np.ones(selMat.shape) \* (-float('inf'))          # scoreNodes 是每个变量的对数 log(p(x))          self.scoreNodes = (self.X).apply(compute\_init\_ll, axis=0).values          self.score\_type = score\_type          self.debug = debug          self.Dn = X.shape[0]          self.pn = X.shape[1]          self.bicterm = np.log(self.Dn) / self.Dn / 2          self.base\_model\_para = base\_model\_para          base\_model = base\_model\_para['base\_model']          if self.debug:              print("score of each variable without add any edge")              print(self.scoreNodes)          # 分类选取交叉检验          if base\_model == 'lgbm':              from lgbm\_cvmodel import ModelWrapper          elif base\_model == 'gam':              from gam\_cvmodel import ModelWrapper          else:              raise NotImplementedError(                  f"currently we only support 'lgbm' and 'gam'.")          self.ModelWrapper = ModelWrapper          self.cv\_split = cv\_split          self.ll\_type = ll\_type      def set\_scoreMat(self, scoreMat):          self.scoreMat = scoreMat      def set\_scoreNodes(self, scoreNodes):          self.scoreNodes = scoreNodes      def \_compute\_ll(self, x):          Y = self.X.iloc[:, x[1]]          # isinstance 判断是否是已知类型          if isinstance(self.X\_encode, list):              model = self.ModelWrapper(X=self.X\_encode[x[0]], Y=Y,                                        cv\_split=self.cv\_split,                                        ll\_type=self.ll\_type,                                        para=self.base\_model\_para                                        )          elif isinstance(self.X\_encode, pd.DataFrame):              X\_input = self.X\_encode.iloc[:, [x[0]]]              model = self.ModelWrapper(X=X\_input, Y=Y, cv\_split=self.cv\_split,                  ll\_type=self.ll\_type,para=self.base\_model\_para)          else:              raise ValueError("The type of X\_encode must be list of numpy "                              "array or pandas DataFrame")          ll, edof = model.fit()          if edof == 0:              edof = 1          #print(ll, edof, self.bicterm, x[0], x[1])          if self.score\_type == 'll':              self.scoreMat[x[0], x[1]] = ll          elif self.score\_type == 'bic':              self.scoreMat[x[0], x[1]] = ll - edof\*self.bicterm          elif self.score\_type == 'aic':              self.scoreMat[x[0], x[1]] = ll - edof/self.Dn      def initialScoreMat(self):          np.apply\_along\_axis(self.\_compute\_ll, axis=1, arr=self.valid\_pair)  # apply\_along\_axie 将某个函数沿坐标轴作用与 arr          # 当前，self.scoreMat 是每个模型的分数          if self.debug:              print("score of each variable when adding the first edge")              print(self.scoreMat)          self.scoreMat = self.scoreMat - self.scoreNodes          # 当前，self.scoreMat是添加边与不添加边的分数差异(adding the edge and not add the edge)          if self.debug:              print("score improve of each variable when adding the first edge")              print(self.scoreMat)          return self.scoreMat, self.scoreNodes      def \_update\_ll(self, x):          Y = self.X.iloc[:, x[-1]]          if isinstance(self.X\_encode, list):              X\_input = np.concatenate([self.X\_encode[i] for i in x[:-1]],                                      axis=1)          # np.concatenate 用于合并数组          elif isinstance(self.X\_encode, pd.DataFrame):              X\_input = self.X\_encode.iloc[:, x[:-1]]          else:              raise ValueError("The type of X\_encode must be list of numpy "                              "array or pandas DataFrame")          model2 = self.ModelWrapper(X=X\_input, Y=Y, cv\_split=self.cv\_split,                                    ll\_type=self.ll\_type,                                    para=self.base\_model\_para)          ll, edof = model2.fit()          if edof==0:              edof = len(x)-2          if self.score\_type == 'll':              self.scoreMat[x[-2], x[-1]] = ll          elif self.score\_type == 'bic':              self.scoreMat[x[-2], x[-1]] = ll - edof\*self.bicterm          elif self.score\_type == 'aic':              self.scoreMat[x[-2], x[-1]] = ll - edof/self.Dn      def \_fillninf(self, x):          self.scoreMat[x[-2], x[-1]] = -float('inf')      def scoreupdate(self, Adj, j):          existingParOfJ = np.where(Adj[:, j] == 1)[0]          # 找有连接的          notAllowedParOfJ = np.setdiff1d(              np.where(self.scoreMat[:, j] == -float('inf'))[0],              np.append(existingParOfJ, [j]))          if len(existingParOfJ) + len(notAllowedParOfJ) < self.p:              # 获取 undecided candidate 的索引              toUpdate = np.setdiff1d(np.arange(self.p), np.concatenate(                  (existingParOfJ, notAllowedParOfJ, [j])))              # np.setdiff1d 作差集              update\_need = np.concatenate(                  (                  np.tile(existingParOfJ, (len(toUpdate), 1)),  # existingParOfJ                  toUpdate.reshape(-1, 1),        # 添加的 candidate                  np.tile(j, (len(toUpdate), 1))  # 目标                  )                  , axis=1)              # 这里，reshape 弄成 1 列              # 最后一个 np.tile()，这里是 j 沿 y 轴复制 len，x 轴不变              if update\_need.shape[0] > 0:        # 需要更新的情况                  if len(existingParOfJ) < self.maxNumParents:                      np.apply\_along\_axis(self.\_update\_ll, axis=1,                                          arr=update\_need)                  else:                      np.apply\_along\_axis(self.\_fillninf, axis=1,                                          arr=update\_need)                  if self.debug:                      print("the score matrix after adding an edge")                      print(self.scoreMat)                  self.scoreMat[:, j] = self.scoreMat[:, j] - self.scoreNodes[j]                  if self.debug:                      print(                          "score improve of each variable when adding an edge")                      print(self.scoreMat)          return self.scoreMat, self.scoreNodes |

**3.3.4.3 算法相关概念介绍**

该模块实现的算法，即作者在文中提到的“引入了一个贪婪过程”。

作者提到的贪婪过程如下（第二环节，因果DAG结构学习）：

To alleviate this, we introduce a greedy procedure (summarized as Algorithm 1 in Appendix A) that starts with an empty DAG and adds the edge corresponding to the largest gain in score during each iteration.

作者引入的这个贪婪过程（在原文附录a中总结为算法1），从一个空DAG开始，并在每次迭代中添加对应于得分最大增益的边。对第j变量的初始得分进行了定义。离散变量的经验概率通过频率计算，而连续变量的经验概率通过核密度估计（KDE）估计。在每次迭代中，添加一条潜在边的增益，增益最大的边将被加到DAG中。为了保证无环，有向路径矩阵（O）将相应地更新，如果存在从节点j到节点l的路径，则一个边l→j将不被视为有效候选。对于连续变量，CVMIC评分中的可能性也通过KDE进行估计，以避免假设噪声分布。

**3.3.5 RCIT.py代码块**

**3.3.5.1 代码作用**

本模块实现的是，第一步骨架学习的混合类型随机化 CIT ，对应原论文3.2部分。

在本模块中，主要包括：

(17)函数代码段：def rff\_mixed；

(18)函数代码段：def RIT\_core；

(19)函数代码段：def RCIT\_core；

(20)类代码段：class RCITIndepTest。

在(17)中，实现了随机傅立叶特征（RFF）混合。其中，x是某一个或者或一组特征的所有样本的值，离散变量以独热码的形式传递进来；函数通过RFF方法计算出RFF的具体值，送到core函数进行核函数计算；输出是一个矩阵、大小、样本数\*numf。随机傅立叶特征混合的详细实现方法，我们在注释中说明。

在(18)中，实现了分类变量的内核岭回归。参数的含义分别是：four\_x：特征x的RFF大小；four\_y：特征y的 RFF；r：样本数；num\_f：RFF中包含的特征数。分类变量的内核岭回归的详细实现方法，我们在注释中说明。

在(19)中，实现了连续变量的内核岭回归。与分类变量不同的是：需要根据给定的数据和权重估计协方差矩阵，计算矩阵的 Cholesky 分解，解决ax=b方程中的x。其余的步骤，与分类变量的内核岭回归基本一致。

在(20)中，实现了混合类型数据独立性条件测试；功能是做独立性检验，返回一个p值。如果p值大于alpha，则接受独立假设，说明输入的数据是独立的则说明两个数据没有边。参数的含义分别是：X\_encode是经过离散值独热码处理的df, 它是一个列表，有df的列数个矩阵，每个矩阵对应一个特征；down：为是否下采样；num\_f，num\_f2为模型固有参数。

**3.3.5.2 源代码**

|  |
| --- |
| ### 本文件：第一步骨架学习的混合类型随机化 CIT ，对应论文 3.2 部分  import numpy as np  import math  import random  from utils import normalize  from scipy.spatial.distance import pdist  from scipy.linalg import cholesky, solve\_triangular  from rpy2.robjects.numpy2ri import numpy2rpy  import rpy2.robjects.packages as rpackages  momentchi2 = rpackages.importr('momentchi2')  # 第一步骨架学习的混合类型随机化 CIT ，对应论文 3.2 部分  # Random Fourier Features：随机傅里叶特征（RFF）  # 随机傅立叶特征混合  # x是某一个或者或一组特征的所有样本的值，离散变量以独热码的形式传递进来  # 函数通过 RFF 方法计算出 RFF 的具体值，送到 core 函数进行核函数计算  # 输出是一个矩阵 大小 样本数\*numf  def rff\_mixed(x, num\_f=10):      # 使一维向量 reshape 成二维矩阵      if len(x.shape) == 1:          x = x.reshape(-1,1)      cat\_idx = []    # 离散变量 discrete      for i in range(x.shape[1]):          if len(np.unique(x[:,i])) == 2:    # 如果这一列只有两个不同的值（独热吗），cat\_idx 增加列索引              cat\_idx.append(i)      # 提取出来所有的离散变量      x\_disc = x[:, cat\_idx]      # x\_cont 是 xdisc 的补集，即所有的连续变量      x\_cont = x[:, np.setdiff1d(range(x.shape[1]), cat\_idx)]      # 离散变量的个数，连续变量的个数      n\_disc, n\_cont = np.sum(x\_disc[0, :]), x\_cont.shape[1]      # 按照比例分配特征数      num\_f\_cont = int(n\_cont/(n\_disc+n\_cont)\*num\_f)      num\_f\_disc = num\_f - num\_f\_cont      if x\_cont.shape[1] != 0:          r, c = x\_cont.shape   # 样本数和特征数          r1 = min(r, 500)          # pdist的功能 计算矩阵 X 样本之间 (m\*n) 的欧氏距离 (2-norm) ，返回值为 Y (m\*m) 为压缩距离元组或矩阵。          # 也就是计算前 r1 个样本的欧式距离中位数          sigma = np.median(pdist(x\_cont[:r1, :], "euclidean")) # 沿指定轴计算中间值          if sigma == 0 or np.isnan(sigma):              sigma = 1          # w，b 是满足一定分布的随机值，w 的形状是连续 rff 特征数量\*连续变量的个数          # 从正态（高斯）分布中随机抽取样本          w = (1 / sigma) \* np.random.normal(0, 1, size=num\_f\_cont \* c).reshape(num\_f\_cont, c)          # 用 b 作为元素构造矩阵，每一列是相同的连续 rff 特征数量个值，一共有样本数个，为了方便计算          # 从均匀分布中抽取样本          b = 2 \* math.pi \* np.random.uniform(0, 1, size=num\_f\_cont)          b = np.tile(b.reshape(-1, 1), (1, r))          # RFF算法的z(x)计算结果          cont\_feat = np.sqrt(2) \* np.cos(np.dot(w, x\_cont.T) + b).T      # 与上面的基本一样区别在于 sigma 的计算      if x\_disc.shape[1] != 0:          r, c = x\_disc.shape          r1 = min(r, 500)          sigma = 1   # 区别在于 sigma 的计算          # 从正态（高斯）分布中随机抽取样本          w = (1 / sigma) \* np.random.normal(0, 1, size=num\_f\_disc \* c).reshape(num\_f\_disc, c)          # 从均匀分布中抽取样本          b = 2 \* math.pi \* np.random.uniform(0, 1, size=num\_f\_disc)          b = np.tile(b.reshape(-1, 1), (1, r))          disc\_feat = np.sqrt(2) \* np.cos(np.dot(w, x\_disc.T) + b).T      # 将两个特征进行连接      if x\_cont.shape[1]!=0 and x\_disc.shape[1] != 0:          return np.concatenate([cont\_feat,disc\_feat],axis=1)      elif x\_cont.shape[1] != 0:          return cont\_feat      else:          return disc\_feat  # 分类变量的内核岭回归 (RFF)  # rit 核函数，没有 z 这个附加条件  # 参数：  #   four\_x：特征 x 的 RFF 大小 样本数\*numf  #   four\_y：特征 y 的 RFF  #   r：是样本数  #   num\_f：RFF中包含的特征数  def RIT\_core(four\_x, four\_y, r, num\_f):      # cov 会把一行当成一个特征，一列当成一个样本，所以需要进行转置      # cov(a,b) 会把 ab 的所有特征都两两求协方差，所以大小是 2numf x 2numf      # 计算出特征之间的协方差，Cxy 的大小为 num\_f x num\_f      Cxy = np.cov(four\_x.T,four\_y.T)[:-num\_f, -num\_f:]      # 标准差      Sta = r\*np.sum(Cxy\*Cxy)      # 每一个样本都减去特征的均值      res\_x = four\_x - np.tile(np.mean(four\_x,0),(r,1))      res\_y = four\_y - np.tile(np.mean(four\_y,0),(r,1))      # 计算样本的协方差      # d 是所有的二元对，x 的某一个特征，y 的某一个特征      d = np.array([(x, y) for x in range(four\_x.shape[1]) for y in range(four\_y.shape[1])])      res = res\_x[:,d[:,1]]\*res\_y[:,d[:,0]]      Cov = 1/r \* np.dot(res.T,res)      eig\_d = np.linalg.eig(Cov)          # 返回特征值和特征向量      eig\_d = eig\_d[0][eig\_d[0].imag==0]  # 筛选出是正实数的特征值      eig\_d = eig\_d[eig\_d.real > 0]      eig\_d = eig\_d.real      try:          p = 1- momentchi2.lpb4(numpy2rpy(eig\_d), Sta.item())[0]      except:          p = 1 - momentchi2.hbe(numpy2rpy(eig\_d), Sta.item())[0]      p = max(p,np.exp(-40))      return p  # 连续变量的内核岭回归 (RFF)  def RCIT\_core(four\_x, four\_y, four\_z, r, num\_f, num\_f2):      # 根据给定的数据和权重估计协方差矩阵      Cxy = np.cov(four\_x.T,four\_y.T)[:-num\_f2, -num\_f2:]      Cxz = np.cov(four\_x.T,four\_z.T)[:-num\_f, -num\_f:]      Czy = np.cov(four\_z.T,four\_y.T)[:-num\_f2, -num\_f2:]      Czz = np.cov(four\_z.T)      # 计算矩阵的 Cholesky 分解      Lzz = cholesky(Czz + np.eye(num\_f)\*(1e-10), lower=True)      # 解决 ax=b 方程中的 x，(假定 a 是一个上/下三角矩阵)      A = solve\_triangular(Lzz, Cxz.T, lower=True)      e\_x\_z = np.dot(four\_z, solve\_triangular(Lzz.T, A, lower=False))      A = solve\_triangular(Lzz, Czy, lower=True)      B = solve\_triangular(Lzz.T, A, lower=False)      e\_y\_z = np.dot(four\_z, B)      res\_x = four\_x - e\_x\_z      res\_y = four\_y - e\_y\_z      Cxy\_z = Cxy - np.dot(Cxz, B)      Sta = r\*np.sum(Cxy\_z\*Cxy\_z)      # 其余的步骤与原先一致      d = np.array([(x, y) for x in range(four\_x.shape[1]) for y in range(four\_y.shape[1])])      res = res\_x[:,d[:,1]]\*res\_y[:,d[:,0]]      Cov = 1/r \* np.dot(res.T,res)      eig\_d = np.linalg.eig(Cov) # 计算正方形阵列的特征值和右特征向量      eig\_d = eig\_d[0][eig\_d[0].imag==0]      eig\_d = eig\_d[eig\_d.real > 0]      eig\_d = eig\_d.real      try:          p = 1- momentchi2.lpb4(numpy2rpy(eig\_d), Sta.item())[0]      except:          p = 1 - momentchi2.hbe(numpy2rpy(eig\_d), Sta.item())[0]      p = max(p,np.exp(-40))      return p  # 混合类型数据独立性条件测试；功能是做独立性检验，返回一个 p 值，  # 如果 p 值大于 alpha，则接受独立假设，说明输入的数据是独立的则说明两个数据没有边  # 参数:  #   X\_encode, 经过离散值独热码处理的 df, 是一个列表，有 df 的列数个矩阵，每个矩阵对应一个特征  #   down (是否下采样)  #   num\_f,num\_f2 模型固有参数  class RCITIndepTest(object):      def \_\_init\_\_(self, suffStat,  down=False, num\_f=100, num\_f2=10):          # n 是样本数，c 是特征数，r 是处理后的最终样本数          self.n, self.c = suffStat[0].shape[0], len(suffStat)          # 进行下采样处理          if not down:              self.suffStat = suffStat              self.r = self.n          else:              # idx 是保留的样本索引              # 也就是说，会控制样本数不超过特征数（独热码算一个特征）的100倍              self.idx = random.sample(range(self.n), self.c \* 100) \                  if self.c \* 100 < self.n  else np.array(range(self.n))              self.suffStat = []              for i in range(len(suffStat)):                  self.suffStat.append(suffStat[i][self.idx])              self.r = len(self.idx)          # 保持傅里叶变换          self.fft\_feature\_f2 = {}  # 将计算好的 RFF 存在里面，避免重复计算          self.fft\_feature\_f = {}          self.num\_f = num\_f          self.num\_f2 = num\_f2      # KRR 核岭回归，返回估计的 p 值      # xy 是需要检验的边，z 是 x 为起点的所有不是 x--y 的边的索引，具体形式见 skelprune.py，kwarg 是空的      # 以下的判断 if 语句的作用都是判别某一组特征的 RFF 是否已经被计算过      # 测试独立性是否成立      def fit(self, x, y, z=None, \*\*kwargs):          if z is None or len(z) == 0:  # 分类变量              if x not in self.fft\_feature\_f2:                  self.fft\_feature\_f2[x] = normalize(                      rff\_mixed(self.suffStat[x], num\_f=self.num\_f2))              if y not in self.fft\_feature\_f2:                  self.fft\_feature\_f2[y] = normalize(                      rff\_mixed(self.suffStat[y], num\_f=self.num\_f2))              return RIT\_core(four\_x=self.fft\_feature\_f2[x],                              four\_y=self.fft\_feature\_f2[y],                              r=self.r, num\_f=self.num\_f2)          else:  # 连续变量              # print(x, y, z)              if x not in self.fft\_feature\_f2:                  self.fft\_feature\_f2[x] = normalize(                      rff\_mixed(self.suffStat[x], num\_f=self.num\_f2))              y = frozenset([y] + z)  # 不可修改的集合              if y not in self.fft\_feature\_f2:                  # 数组拼接函数，把 y 里的所有特征的具体值全部拿进来                  suffStaty = np.concatenate([self.suffStat[i]                                              for i in y], axis=1)                  self.fft\_feature\_f2[y] = normalize(                      rff\_mixed(suffStaty, num\_f=self.num\_f2))              setz = frozenset(z)              if setz not in self.fft\_feature\_f:                  suffStatz = np.concatenate([self.suffStat[i]                                              for i in setz], axis=1)                  self.fft\_feature\_f[setz] = normalize(                      rff\_mixed(suffStatz, num\_f=self.num\_f))              return RCIT\_core(four\_x=self.fft\_feature\_f2[x],                              four\_y=self.fft\_feature\_f2[y],                              four\_z=self.fft\_feature\_f[setz],                              r=self.r, num\_f=self.num\_f, num\_f2=self.num\_f2) |

**3.3.5.3 算法相关概念介绍**

本模块实现的是论文当中的第一环节，骨架学习的混合类型随机化CIT。

在原文中的具体表述为：

MRCIT: A Randomized Conditional Independence Test for Mixed-type Data.

As mentioned in Introduction section and Figure 1, in our hybrid algorithm, CITs are used in both causal skeleton learning and final pruning to reduce the search space of causal DAG learning and reduce false positive, respectively. Therefore, we propose a randomized CIT for Mixed-type Data. To test whether X1⊥X2|X3, where X3 can be multivariate, a nonparametric methods called KCIT is introduced by Zhang. Strobl et al. further propose RCIT utilizing random Fourier features (RFFs) to get a faster CIT without sacrificing accuracy. However, cases when variables {X1, X2, X3} involving both continuous and categorical variables are not considered. Here we extend RCIT to handle mixed-type data.

也就是说，在作者的混合算法中，CIT分别用于因果骨架学习和最终剪枝，以减少因果DAG学习的搜索空间和减少误报。因此，作者提出了混合型数据的随机CIT。有其他作者介绍了一种称为KCIT的非参数方法。Strobl等人进一步提出，RCIT利用随机傅立叶特征（RFF）获得更快的CIT，而不牺牲准确性。然而，这种算法当变量同时涉及连续变量和分类变量的情况没有被考虑。在这里，作者使用了扩展RCIT算法，来处理混合类型的数据。

除此之外，对于随机特征(Random Feature)方法，即利用一种“以具有转移不变性(Transition Invariant)的核的取值”作为不变量的变换(称为随机特征变换, Random Feature Map)，施加于输入的随机变量，得到“特征”（经过适当的常数正则化，成为一个随机变量）。将核的取值，转化为特征的数学期望，以这种方法来取代Gram矩阵，近似未知的核的取值。

而文中提及的随机傅里叶特征方法，是一种近似核函数的方法，旨在找到一个低阶的映射函数z(x)，将d维原始数据映射到D维随机特征空间中。

**3.3.6 gam\_cvmodel.py代码块**

**3.3.6.1 代码作用**

在本模块中，主要包括：

(21)函数代码段：def GamClsModel；

(22)函数代码段：def GamRegModel；

(23)类代码段：class ModelWrapper。

在(21)中，建立了逻辑回归模型，将多分类任务转为二分类任务，并将输入的train放入模型；返回的是test的对数损失值和模型有效自由度的均值。

在(22)中，建立了线性回归模型，利用输入的train得到testX对数预测值，与testY相减得到差；利用高斯核进行核密度估计，最后返回差值，核密度估计的和，模型的有效自由度。

在(23)中，实现了模型的封装。该模型先后进行了初始化操作、分情况使用逻辑/线性回归，（如果数据类型是离散变量用逻辑回归，否则用线性回归）、标签标准化、交叉验证、计算模型分数平均值、返回利用线性回归得到的训练值与测试值只差的高斯核密度分布之和、高斯核密度估计等功能。

**3.3.6.2 源代码**

|  |
| --- |
| from pygam import LogisticGAM, LinearGAM, s, f  from sklearn.multiclass import OneVsRestClassifier  import pandas as pd  import numpy as np  from sklearn.neighbors import KernelDensity  from sklearn.metrics import log\_loss  from sklearn import preprocessing  from sklearn.model\_selection import train\_test\_split  from scipy.stats import gaussian\_kde  from sklearn.model\_selection import StratifiedKFold, KFold  # 建立逻辑回归模型将多分类任务转为二分类任务，并将输入的 train 放入模型  # 返回 test 的对数损失值和模型有效自由度的均值  def GamClsModel(sfun=None,                n\_jobs=1,         # 并行作业数，使用的是默认值 1                X\_train=None,                y\_train=None,                X\_test=None,                y\_test=None                ):      base\_model = LogisticGAM(sfun) # 建立逻辑模型      ovr\_classifier = OneVsRestClassifier(base\_model, n\_jobs=n\_jobs) # 转为二分类任务      ovr\_classifier.fit(X\_train, y\_train) # 输入参数      prob\_pred = ovr\_classifier.predict\_proba(X\_test) # 预测      # 计算模型的对数似然（log likelihood）      edofs = [est.statistics\_['edof'] for est in ovr\_classifier.estimators\_]      ll = -log\_loss(y\_true=y\_test, y\_pred=prob\_pred, normalize=False) # 得到损失率      return ll, np.mean(edofs) # mean 为去平均值  # 建立线性回归模型，利用输入的 train 得到 testX 对数预测值，与 testY 相减得到差，  # 利用高斯核进行核密度估计，最后返回差值，核密度估计的和，模型的有效自由度  def GamRegModel(sfun=None,                  X\_train=None,                  y\_train=None,                  X\_test=None,                  y\_test=None                  ):      gam = LinearGAM(sfun).fit(X\_train, y\_train)      y\_pred = gam.predict(X\_test)      residule = y\_test - y\_pred      kde = gaussian\_kde(residule)               # 进行高斯核密度分布估计      logprob = np.log(kde.evaluate(residule))   # 类似于取值并取对数      return residule, np.sum(logprob), gam.statistics\_['edof']  class ModelWrapper(object):      def \_\_init\_\_(self,                  X,                  Y,                             # X,Y 皆为测试集                  para=None,                     # 参数类型                  train\_test\_split\_ratio=0.0,                  cv\_split=5,                    # 交叉验证的次数                  ll\_type='local'                # koe 估算 kde: 'local' 或者 'global'                  ):          """          :param X: X 是一个 pandas 数据框架（data frame）          :param Y: Y 是一个 pandas 数据序列（data series）          DEFAULT\_GAM\_PARA = {              "spline\_order": 10,              "lam": 0.6,              "n\_jobs": 1,              "use\_edof": True,          }          """          spline\_order = para['spline\_order']          lam = para['lam']          n\_jobs = para['n\_jobs']          use\_edof = para['use\_edof']  # 是否需要计算自由度          self.X = X          self.train\_test\_split\_ratio = train\_test\_split\_ratio          p = X.shape[1]          cols = list(X.columns)          if (X[cols[0]].dtypes == 'O' or X[cols[0]].dtypes == 'bool'                  or X[cols[0]].dtype.name == 'category'):  # 判断是否是离散特征              sfun = f(0, lam=lam)  # 每一个特征都对应一个 item，sfun 是所有 item 的集合          else:              sfun = s(0, spline\_order=spline\_order)          for i in range(1, p):              if (X[cols[i]].dtypes == 'O' or X[cols[i]].dtypes == 'bool'                      or X[cols[i]].dtype.name == 'category'):                  sfun = sfun + f(i, lam=lam)              else:                  sfun = sfun + s(i, spline\_order=spline\_order)          self.Y = Y          self.sfun = sfun          self.n\_jobs = n\_jobs          self.use\_edof = use\_edof          self.cv\_split = cv\_split          self.ll\_type = ll\_type      def fit(self):          n\_split = self.cv\_split          ll\_type = self.ll\_type          total\_ll = 0          total\_num = 0          total\_edof = 0          # 分情况使用逻辑/线性回归。如果数据类型是离散变量用逻辑回归，否则用线性回归。          if (self.Y.dtypes == 'O' or self.Y.dtypes == 'bool'                  or self.Y.dtype.name == 'category'):              # 下面，将标签标准化；例：1, 2, 6, 7 ---> 0, 1, 2, 3              le = preprocessing.LabelEncoder()              le.fit(self.Y)              self.Y = le.transform(self.Y)   # 将 self.Y 标签化为 0 ~ len(self.Y)-1 之间的数字              if n\_split == 0:  # 划分次数为 0 不需要划分                  sumll, edof = GamClsModel(sfun=self.sfun, n\_jobs=self.n\_jobs,                                            X\_train=self.X, y\_train=self.Y,                                            X\_test=self.X, y\_test=self.Y)                  total\_ll += sumll           # 损失                  total\_num += len(self.Y)    # 预测的样本数                  total\_edof += edof          # 自由度              else:                  skf = StratifiedKFold(n\_splits=n\_split) # 等比例划分                  skf.get\_n\_splits(self.X, self.Y)                  # 随机将 self.X, self.Y 划分，实现交叉验证                  # 具体就是将将训练集等比例划分为 n\_split 份，然后每次取一份，把这一份当成训练集放到测试集上，                  # 进行 n\_split 次后得到分类率的平均值，作为该模型平均值                  for train\_ind, test\_ind in skf.split(self.X, self.Y):                      X\_train, X\_test = self.X.iloc[train\_ind], self.X.iloc[test\_ind]                      y\_train, y\_test = self.Y[train\_ind], self.Y[test\_ind]                      sumll, edof = GamClsModel(                          sfun=self.sfun, n\_jobs=self.n\_jobs,                          X\_train=X\_train, y\_train=y\_train,                          X\_test=X\_test, y\_test=y\_test)                      total\_ll += sumll                      total\_num += len(y\_test)                      total\_edof += edof              if self.use\_edof:                  return total\_ll/total\_num, total\_edof/n\_split              else:                  return total\_ll/total\_num, 0          else:              residule = np.array([])              if n\_split == 0:  # 划分次数为 0 则不需要划分                  presidule, sumll, edof = GamRegModel(                      sfun=self.sfun, X\_train=self.X,                      y\_train=self.Y, X\_test=self.X, y\_test=self.Y)                  residule = np.append(residule, presidule)                  total\_ll += sumll                  total\_num += len(self.Y)                  total\_edof += edof              else:                  # 具体就是将将训练集比例划分为 n\_split 份，然后每次取一份，把这一份当成训练集放到测试集上，                  # 进行 n\_split 次后得到分数的平均值，作为该模型平均值                  # 注意，与上面不同的是 KFold 不是等比例划分                  kf = KFold(n\_splits=n\_split)  # 随机比例划分                  kf.get\_n\_splits(self.X)                  residule = np.array([])                  for train\_ind, test\_ind in kf.split(self.X):                      X\_train, X\_test = self.X.iloc[train\_ind], self.X.iloc[test\_ind]                      y\_train, y\_test = self.Y[train\_ind], self.Y[test\_ind]                      presidule, sumll, edof = GamRegModel(                          sfun=self.sfun, X\_train=X\_train,                          y\_train=y\_train, X\_test=X\_test, y\_test=y\_test)                      residule = np.append(residule, presidule)                      total\_ll += sumll                      total\_num += len(y\_test)                      total\_edof += edof              # 有 local，则直接返回利用线性回归得到的训练值与测试值只差的高斯核密度分布之和；没有 local，则还需要进行平均值              # self.use\_edof 代表返不返回平均有效自由度              if ll\_type == 'local':                  if self.use\_edof:                      return total\_ll / total\_num, total\_edof/n\_split                  else:                      return  total\_ll / total\_num, 0              else:                  # residule 是所有折的 ytest-ypre 的集合，相当于把函数内的所有偏差综合在一起进行高斯核密度估计                  # 也就是说在 global 模式下，不采用每一折的高斯核密度估计进行取均值，而是重新计算整个高斯核密度估计                  kde = gaussian\_kde(residule)              # 高斯核密度分布                  logprob = np.log(kde.evaluate(residule))  # 取对数                  if self.use\_edof:                      return np.mean(logprob), total\_edof/n\_split                  else:                      return np.mean(logprob), 0 |

**3.3.6.3 算法相关概念介绍**

代码段中利用的高斯核进行核密度估计中，核密度估计（kernel density estimation）是在概率论中用来估计未知的密度函数，属于非参数检验方法之一。

高斯核函数：K(x,xc)=exp[-||x-xc||2/(2\*σ)2]，其中xc为核函数中心，σ为函数的宽度参数。

核密度估计通过核函数（如高斯）将每个数据点的数据+带宽当作核函数的参数，得到N个核函数，再线性叠加就形成了核密度的估计函数，归一化后就是核密度概率密度函数了。

**3.3.7 lgbm\_cvmodel.py代码块**

**3.3.7.1 代码作用**

在本模块中，主要包括：

(24)类代码段：class LgbClsModel；

(25)类代码段：class LgbRegModel；

(26)类代码段：class ModelWrapper。

在(24)、(25)中，实现了lgb模型的参数配置、模型训练并得出预测结果的功能。其中，is\_unbalance用于binary分类，使用列集中的pos/neg分数；boosting='gbdt'为提升类型：传统的梯度提升决策树——决策树类型；num\_leaves为每个基学习器的最大叶子节点；feature\_fraction为特征分数/子特征处理列采样；learning\_rate为梯度下降的步长；num\_boost\_round为迭代次数；num\_class用于multiclass分类；early\_stopping\_round用于如果一次验证数据的一个度量在最近的round中没有提高，使模型停止训练；bagging\_fraction为每次迭代时用的数据比例；bagging\_freq为bagging的次数。

在(26)中，实现了Lightgbm基于决策树算法的梯度提升算法的模型封装。进行了模型参数定义、模型训练预测和交叉验证。

**3.3.7.2 源代码**

|  |
| --- |
| import lightgbm as lgb  import pandas as pd  import numpy as np  from sklearn.neighbors import KernelDensity  from sklearn.metrics import log\_loss  from sklearn import preprocessing  from sklearn.model\_selection import train\_test\_split  from scipy.stats import gaussian\_kde  from sklearn.model\_selection import StratifiedKFold, KFold  class LgbClsModel(object):      def \_\_init\_\_(self,                  is\_unbalance='true',       # 用于 binary 分类，使用列集中的 pos/neg 分数                  boosting='gbdt',           # 提升类型：传统的梯度提升决策树——决策树类型                  num\_leaves=31,             # 每个基学习器的最大叶子节点                  feature\_fraction=0.5,      # 特征分数/子特征处理列采样                  learning\_rate=0.05,        # 梯度下降的步长。常用 0.1, 0.001, 0.003                  num\_boost\_round=20,        # 迭代次数                  num\_class=2,               # 只用于 multiclass 分类                  early\_stopping\_round=3,    # 如果一次验证数据的一个度量在最近的 round 中没有提高，模型将停止训练                  bagging\_fraction=0.5,      # 每次迭代时用的数据比例，建树的样本采样比例                  bagging\_freq=20            # bagging 的次数(与 bagging\_fraction 同时设置)                  ):          # lgb 模型参数配置          self.parameters = {              'objective': 'multiclass',              'metric': 'multi\_logloss',              'is\_unbalance': is\_unbalance,              'boosting': boosting,              'num\_leaves': num\_leaves,              'feature\_fraction': feature\_fraction,              'learning\_rate': learning\_rate,              'num\_class': num\_class,              'bagging\_fraction': bagging\_fraction,              'bagging\_freq': bagging\_freq,              'verbose': -1    # <0 显示致命的, =0 显示错误 (警告), >0 显示信息          }          self.num\_boost\_round = num\_boost\_round          self.early\_stopping\_round = early\_stopping\_round          self.model = None      # lgb 模型训练，得出预测结果      def fit(self, X\_train=None, y\_train=None, X\_test=None, y\_test=None):          lgb\_train = lgb.Dataset(X\_train, y\_train)          test\_data = lgb.Dataset(X\_test, label=y\_test)          # 训练          self.model = lgb.train(              self.parameters,              lgb\_train,              valid\_sets=test\_data,              verbose\_eval=False,              num\_boost\_round=self.num\_boost\_round,              early\_stopping\_rounds=self.early\_stopping\_round          )          # 预测数据集          y\_pred = self.model.predict(X\_test)          # 对数损失是逻辑模型的负对数可能性          return -log\_loss(y\_true=y\_test, y\_pred=y\_pred, normalize=False)  class LgbRegModel(object):      def \_\_init\_\_(self,                  boosting='gbdt',           # 提升类型：传统的梯度提升决策树——决策树类型                  num\_leaves=31,             # 每个基学习器的最大叶子节点                  feature\_fraction=0.5,      # 特征分数/子特征处理列采样                  learning\_rate=0.05,        # 梯度下降的步长                  num\_boost\_round=20,        # 迭代次数                  bandwidth=0.4,                  kernel='gaussian',                  early\_stopping\_round=3,    # 如果一次验证数据的一个度量在最近的 round 中没有提高，模型将停止训练                  bagging\_fraction=0.5,      # 每次迭代时用的数据比例,建树的样本采样比例                  bagging\_freq=20            # bagging 的次数(与 bagging\_fraction 同时设置)                  ):          # lgb 模型参数配置          self.parameters = {              'objective': 'regression',     # 目标函数              'boosting': boosting,              'num\_leaves': num\_leaves,              'feature\_fraction': feature\_fraction,              'learning\_rate': learning\_rate,              'bagging\_fraction': bagging\_fraction,              'bagging\_freq': bagging\_freq,              'verbose': -1                  # <0 显示致命的, =0 显示错误 (警告), >0 显示信息          }          self.early\_stopping\_round = early\_stopping\_round          self.num\_boost\_round = num\_boost\_round          self.bandwidth = bandwidth          self.kernel = kernel          self.model = None      # lgb 模型训练，得出预测结果      def fit(self, X\_train=None, y\_train=None, X\_test=None, y\_test=None):          lgb\_train = lgb.Dataset(X\_train, y\_train)          test\_data = lgb.Dataset(X\_test, label=y\_test)          # 训练          self.model = lgb.train(              self.parameters,              lgb\_train,              valid\_sets=test\_data,              verbose\_eval=False,              num\_boost\_round=self.num\_boost\_round,              early\_stopping\_rounds=self.early\_stopping\_round          )          # 预测数据集          y\_pred = self.model.predict(X\_test)          residule = y\_test - y\_pred          kde = gaussian\_kde(residule)          logprob = np.log(kde.evaluate(residule))          return residule, np.sum(logprob)  # 模型封装: Lightgbm 基于决策树算法的梯度提升算法  class ModelWrapper(object):      def \_\_init\_\_(self,                  X,                  Y,               # pandas 数据序列                  para=None,       # 参数类型                  cv\_split=5,      # 交叉验证次数（可能需要0次）                  ll\_type='local'  # koe 估算 kde: 'local' 或者 'global'                  ):          """          :param X:          :param Y: Y 是一个 pandas 数据序列（data series）          :param is\_unbalance: 是否使用列集中的pos/neg分数          :param boosting: 传统的梯度提升决策树——决策树类型          :param num\_leaves: 每个基学习器的最大叶子节点          :param feature\_fraction: 特征分数/子特征处理列采样          :param learning\_rate: 梯度下降的步长          :param num\_boost\_round: 迭代次数          """          boosting = para['boosting']          num\_leaves = para['num\_leaves']          feature\_fraction = para['feature\_fraction']          learning\_rate = para['learning\_rate']          num\_boost\_round = para['num\_boost\_round']          self.X = X          self.Y = Y          if (Y.dtypes == 'O' or Y.dtypes == 'bool' or                  Y.dtype.name == 'category' or Y.dtypes == 'int'): # 模型参数定义，数据类型为布尔型、int 型、类属              num\_class = len(Y.unique())              self.pred\_model = LgbClsModel(                  boosting=boosting,                  num\_leaves=num\_leaves,                  feature\_fraction=feature\_fraction,                  learning\_rate=learning\_rate,                  num\_boost\_round=num\_boost\_round,                  num\_class=num\_class              )          else: # 模型参数定义：其他数据类型              self.pred\_model = LgbRegModel(                  boosting=boosting,                  num\_leaves=num\_leaves,                  feature\_fraction=feature\_fraction,                  learning\_rate=learning\_rate,                  num\_boost\_round=num\_boost\_round              )          self.fited = False          self.cv\_split = cv\_split          self.ll\_type = ll\_type      def fit(self):          n\_split = self.cv\_split          ll\_type = self.ll\_type          total\_ll = 0          total\_num = 0          if (self.Y.dtypes == 'O' or self.Y.dtypes == 'bool'                  or self.Y.dtype.name == 'category'): # 模型训练预测：数据类型为布尔型、int 型、类属              le = preprocessing.LabelEncoder()              le.fit(self.Y)              self.Y = le.transform(self.Y)              if n\_split == 0: # 交叉验证次数为 0                  sumll = self.pred\_model.fit(X\_train=self.X, y\_train=self.Y,                                              X\_test=self.X, y\_test=self.Y)                  total\_ll += sumll                  total\_num += len(self.Y)              else:                  skf = StratifiedKFold(n\_splits=n\_split)                  skf.get\_n\_splits(self.X, self.Y)                  for train\_ind, test\_ind in skf.split(self.X, self.Y):                      X\_train, X\_test = self.X[train\_ind], self.X[test\_ind]                      y\_train, y\_test = self.Y[train\_ind], self.Y[test\_ind]                      sumll = self.pred\_model.fit(X\_train=X\_train, y\_train=y\_train,                                                  X\_test=X\_test, y\_test=y\_test)                      total\_ll += sumll                      total\_num += len(y\_test)              return total\_ll/total\_num, 0          else: # 模型训练预测：其他数据类型              residule = np.array([])              if n\_split == 0: # 交叉验证次数为 0                  presidule, sumll = self.pred\_model.fit(X\_train=self.X, y\_train=self.Y, X\_test=self.X, y\_test=self.Y)                  residule = np.append(residule, presidule)                  total\_ll += sumll                  total\_num += len(self.Y)              else:                  kf = KFold(n\_splits=n\_split)                  kf.get\_n\_splits(self.X)                  total\_num = 0                  for train\_ind, test\_ind in kf.split(self.X):                      X\_train, X\_test = self.X[train\_ind], self.X[test\_ind]                      y\_train, y\_test = self.Y[train\_ind], self.Y[test\_ind]                      presidule, sumll = self.pred\_model.fit(                          X\_train=X\_train, y\_train=y\_train,                          X\_test=X\_test, y\_test=y\_test)                      residule = np.append(residule, presidule)                      total\_ll += sumll                      total\_num += len(y\_test)              if ll\_type=='local':                  return total\_ll/total\_num, 0              else:                  kde = gaussian\_kde(residule)                  logprob = np.log(kde.evaluate(residule))              return np.mean(logprob), 0 |

**3.3.7.3 算法相关概念介绍**

对于LightGBM，在作者的算法中，被用于建模因果机制，以实现高度的通用性。此外，在LightGBM中，在树构造过程中引入了多重正则化（例如L1和L2范数）和约束（例如树的深度）。

作者原文中的表述为：

We employ LightGBM in this work for non-parametric estimation of the causal mechanism.

作者使用LightGBM对因果机制进行非参数估计。该段代码段即对应原文中第二阶段的后半部分。

**3.3.8 mixed\_causal.py代码块**

**3.3.8.1 代码作用**

在本模块中，主要包括：

(27)函数代码段：def prior\_knowledge\_encode；

(28)函数代码段：def data\_processing；

(29)函数代码段：def mixed\_causal；

(30)函数代码段：def evaluate。

在(27)中，实现了先验知识编码的功能。处理了设置源节点（source\_nodes），源节点没有任何祖先（ancestor）；根据先验知识设置直接边缘（direct\_edges）；根据先验知识设置确定的无直接边缘（not\_direct\_edges）；根据先验知识设置确定的无祖先（no ancestor）四种情形。

在(28)中，实现了数据处理功能。在处理数据时，仍然分为双权和标准化两种。最后，将处理好的数据编码，方式为：编成独热码。

代码段(29)为本算法的核心，通过导入模型参数、并对前面的函数段进行调用，先后完成了文中算法的三个关键环节，即：

Step 1：使用 pc 算法进行框架学习；

Step 2：基于贪心搜索算法，创建dag；

Step 3：通过条件独立性（conditional independence test）测试去除边缘。

按步骤运行了原文中3.3提到的框架学习、因果DAG结构学习、剪枝三个环节。

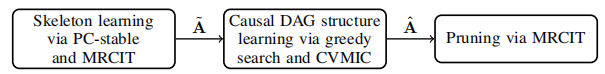
在(30)中，分别评估了三个结果的召回率、精确率等数据，并返回了评估结果。

**3.3.8.2 源代码**

|  |
| --- |
| import pandas as pd  import numpy as np  import time  from utils import data\_preprocess, evaluate\_binary, \      normalize\_biweight, skeleton\_metrics  from camsearch\_prior import greedy\_edgeadding  from sklearn.preprocessing import StandardScaler, FunctionTransformer  from sklearn\_pandas import DataFrameMapper  from skelprune import skeleton, pruning  from RCIT import RCITIndepTest  # 先验知识编码  def prior\_knowledge\_encode(feature\_names,                            source\_nodes=None, direct\_edges=None,                            not\_direct\_edges=None, happen\_before=None):      """      :param feature\_names: list of string, list of feature names      :param source\_nodes: list of string, list of source node      :param direct\_edges: dictionary, {start1:end1, start2:end2}      :param not\_direct\_edges: dictionary, {start1:end1, start2:end2}      :param happen\_before: dictionary,              {node: [ac1, ac2, ac3], node2: [ac1, ac2, ac3]}      :return:      """      p = len(feature\_names)      feature2index = {}      for i, feature in enumerate(feature\_names): # 枚举特征的序号和名字，并储存（建立名字和数字的索引）          feature2index[feature] = i              # feature2index 中存的是随机变量      # 生成 p\*p 初始化为 0 的矩阵      # 类似邻接矩阵的数据结构      prior\_adj = np.zeros((p, p))                # prior\_adj 表示有向图中是否有 start->end 的边      prior\_anc = np.zeros((p, p))      # 设置源节点（source\_nodes），源节点没有任何祖先（ancestor）      if source\_nodes: # 自行添加参数，加上后执行          # 空的就创建，不空就去重          source\_nodes = set(source\_nodes)          for s\_node in source\_nodes:              if s\_node not in feature2index:                  raise ValueError(                      f"the feature: {s\_node} you provide in the source\_nodes "                      f"is not in the column names")              prior\_adj[:, feature2index[s\_node]] = -1  # 该节点（特征名字）的列全部变为 -1              prior\_anc[:, feature2index[s\_node]] = -1              # 无连接      else: # 默认情形          source\_nodes = set()      # 根据先验知识设置直接边缘（direct\_edges）      if direct\_edges: # 自行添加参数，加上后执行          set\_direct\_edges = set()          for start in direct\_edges:              if start not in feature2index:                  raise ValueError(                      f"the feature: {start} you provide in the direct\_edges"                      f" is not in the column names")              for end in direct\_edges[start]:                  if end not in feature2index:                      raise ValueError(                          f"the feature: {end} you provide in the "                          f"direct\_edges is not in the column names")                  if end in source\_nodes:                      raise ValueError(                          f"The prior knowledge you provide is conflict you"                          f" claim the feature {end} is a source node but "                          f"there is an edge to it.")                  ind\_s = feature2index[start]        # 得到该点的序号                  ind\_e = feature2index[end]          # 得到指向的点的序号                  prior\_adj[ind\_s, ind\_e] = 1         # 表明两点单向可达                  set\_direct\_edges.add((start, end))  # 加入到有向图中                  # 建立连接      else:          set\_direct\_edges = set()      # 根据先验知识设置确定的无直接边缘（not\_direct\_edges）      if not\_direct\_edges: # 自行添加参数，加上后执行          for start in not\_direct\_edges:              if start not in feature2index:                  raise ValueError(                      f"the feature: {start} you provide in the "                      f"not\_direct\_edges is not in the column names")              for end in not\_direct\_edges[start]:                  if end not in feature2index:                      raise ValueError(                          f"the feature: {end} you provide in the "                          f"not\_direct\_edges is not in the column names")                  if (start, end) in set\_direct\_edges:                      raise ValueError(                          f"The prior knowledge you provide is conflict please "                          f"check the existence of edge {(start, end)}")                  ind\_s = feature2index[start]        # 得到点的索引                  ind\_e = feature2index[end]          # 得到可到达的点的索引                  prior\_adj[ind\_s, ind\_e] = -1        # 双向可达的状态      # 根据先验知识设置确定的无祖先（no ancestor）      if happen\_before: # 自行添加参数，加上后执行                        # 应是 {feature2index: [feature2index, feature2index, feature2index]} 的形式          for late in happen\_before:              if late not in feature2index:                  raise ValueError(                      f"the feature: {late} you provide in order information "                      f"is not in the column names")              for anc in happen\_before[late]:                  if anc not in feature2index:                      raise ValueError(                          f"the feature: {anc} you provide in order information "                          f"is not in the column names")                  if (late, anc) in set\_direct\_edges:                      raise ValueError(                          f"The prior knowledge you provide is conflict "                          f"please check the existence of edge ({late, anc})")                  ind\_s = feature2index[late]                  ind\_e = feature2index[anc]                  prior\_adj[ind\_s, ind\_e] = -1  # 双向可达状态                  prior\_anc[ind\_s, ind\_e] = -1  # 两点间的关系状态      return prior\_adj, prior\_anc  def data\_processing(df, cat\_index, normalize='biweight'):      columns = df.columns      if normalize == 'biweight':  # 双权          BiweightScaler = FunctionTransformer(normalize\_biweight)   # 创建一个转换器，转换方式为 normalize\_biweight 函数，预期得到一个可以获得归一化的双权值          standardize = [(col, None) if col in cat\_index else        # 遍历每一列，如果此列在测试中(col在cat\_index中)则不做双权标准化(col, None)，否则做                        ([col], BiweightScaler) for col in columns]          x\_mapper = DataFrameMapper(standardize)                    # 变成转换器可处理的格式，DataFrame 是二维数据结构，类似表格          df = x\_mapper.fit\_transform(df).astype('float32')          # 转换          df = pd.DataFrame(df, columns=columns)                     # 转换为具有列标签和行标签的格式，行标签默认为索引，列标签为 columns      elif normalize == 'standard':  # 过程同上，但转化方法不同          standardize = [(col, None) if col in cat\_index else                        ([col], StandardScaler()) for col in columns]          x\_mapper = DataFrameMapper(standardize)          df = x\_mapper.fit\_transform(df).astype('float32')          df = pd.DataFrame(df, columns=columns)      else:          raise NotImplementedError(              f"currently we only support 'biweight' and 'standard'.")      # 编码      df[cat\_index] = df[cat\_index].astype(object)      X\_encode = data\_preprocess(df)  # 将处理好的数据编码，方式为编成独热码      return df, X\_encode  def mixed\_causal(df, X\_encode,model\_para, base\_model\_para,                  prior\_adj=None, prior\_anc=None, selMat=None):      # 导入模型参数      step1\_maxr = model\_para['step1\_maxr']      step3\_maxr = model\_para['step3\_maxr']      maxNumParents = model\_para['maxNumParents']      num\_f = model\_para['num\_f']      num\_f2 = model\_para['num\_f2']      cv\_split = model\_para['cv\_split']      ll\_type = model\_para['ll\_type']      alpha = model\_para['alpha']      downsampling= model\_para['downsampling']      indep\_pvalue = model\_para['indep\_pvalue']      base\_model = base\_model\_para['base\_model']      score\_type = model\_para['score\_type']      p = df.shape[1]  # 返回数据的列数      #######################################################################      # Step 1.使用 pc 算法进行框架学习      indepTest = RCITIndepTest(suffStat=X\_encode, down=downsampling, # 独立性测试 MRCIT                                num\_f=num\_f, num\_f2=num\_f2)      if selMat is None:  # 此处默认没有，如果已经有了这个 skeleton 的话可以跳过，没学习 skeleton 就学习，统计学习时间          t1 = time.time()          skel = skeleton(indepTest, labels=range(p),                          m\_max=step1\_maxr, alpha=indep\_pvalue,                          priorAdj=prior\_adj,                          )          selMat = skel['sk']                     # selMat 为学习到的 Skelton          step1\_train\_time = time.time() - t1     # 得到 step1\_train\_time，框架学习训练时间      else:          step1\_train\_time = 0      #######################################################################      # step 2: 基于贪心搜索算法，创建 dag      # 选择交叉检验的方式      if base\_model == "lgbm":  # lgbm 模型用编码          X = X\_encode      elif base\_model =="gam":  # gam 模型直接用数据          X = df      else:          raise NotImplementedError(              f"currently we only support 'lgbm' and 'gam'.")      # 贪心搜索      t2 = time.time()      dag2 = greedy\_edgeadding(df, X, selMat,                              maxNumParents=maxNumParents,                              alpha=alpha,                              cv\_split=cv\_split,                              ll\_type=ll\_type,                              base\_model\_para=base\_model\_para,                              prior\_adj=prior\_adj,                              prior\_anc=prior\_anc,                              score\_type = score\_type,                              )      step2\_train\_time = time.time() - t2      ######################################################################      # step 3: 通过条件独立性（conditional independence test）测试去除边缘      # 剪枝      t3 = time.time()      dag = pruning(indepTest, dag2, m\_max=step3\_maxr,                    alpha=indep\_pvalue, priorAdj=prior\_adj)      step3\_train\_time = time.time() - t3      return(selMat, dag2, dag, step1\_train\_time,            step2\_train\_time, step3\_train\_time)  def evaluate(trueG, skel\_bool, dag2, dag):      skel = skel\_bool.astype('int')      # skelton 修改类型      if skel.shape != trueG.shape != dag.shape:  # 保证邻接矩阵的尺寸相同          raise ValueError(f"the shape of true adjacency matrix and the "                          f"predicted skeleton and dag is not same!")      # 分别评估 3 种      skl\_result = pd.DataFrame(skeleton\_metrics(trueG, skel), index=[0])      # print(skl\_result)      dag2\_result = pd.DataFrame(evaluate\_binary(trueG, dag2), index=[0]) # 将召回率精确率等数据      dag\_result = pd.DataFrame(evaluate\_binary(trueG, dag), index=[0])      return skl\_result, dag2\_result, dag\_result |

**3.3.8.3 算法相关概念介绍**

本模块运行了对应原文3.3中三个环节的每一个环节。



原文中的流程图介绍了算法的实现流程。该算法中，A~是框架学习阶段的输出，即表示无向图邻接矩阵的对称矩阵。Aˆ是第二阶段的输出，即DAG的邻接矩阵。

除此之外，通过召回率、精确率等数据，本模块还对三个步骤的结果进行了评估。

**3.3.9 skelprune.py代码块**

**3.3.9.1 代码作用**

在本模块中，主要包括：

(31)函数代码段：def skeleton；

(32)函数代码段：def pruning。

在(31)中，具体地实现了框架学习PC-stable算法，在因果马尔可夫假设和因果忠实性假设下学习有向无环图 (DAGs)。该算法中，迭代地扩大条件集的大小，如果节点之间的对应变量通过MRCIT（条件）独立，则丢弃节点之间的边。

在(32)中，具体实现了剪枝算法，利用 MRCITs 测试每个父子对条件作用对所有其他直接原因的独立性，对避免选择难以确定的阈值而新增的"多余"的边进行修剪。

**3.3.9.2 源代码**

|  |
| --- |
| import numpy as np  import itertools  from utils import reachable  # 框架学习：PC-stable 算法 [ 在因果马尔可夫假设和因果忠实性假设下学习有向无环图 (DAGs) ]  # 迭代地扩大条件集的大小，如果节点之间的对应变量通过 MRCIT（条件）独立，则丢弃节点之间的边  # 参数含义:  #   indepTest 条件独立性检验类  #   labels range(p), p是 df 的列数, df 是 alram\_simulate.csv  #   m\_max model\_para['step1\_maxr']=1  #   alpha 显著性标准 0.05 对应 95% 置信水平  def skeleton(indepTest, labels, m\_max, alpha=0.05, priorAdj=None, \*\*kwargs):      # 创建一个 p\*p 大小的矩阵，矩阵内的元素都为 None      sepset = [[None for i in range(len(labels))] for i in range(len(labels))]      # 把有向图 G 的有向边变成无向边，形成完整无向图；如果需要研究边 i--j，则为真      G = [[True for i in range(len(labels))] for i in range(len(labels))]      # 把对角线元素去掉，去掉自己到的边      for i in range(len(labels)): G[i][i] = False      done = False     # done 标志      ord = 0      n\_edgetests = {} # 第ord次循环进行过的独立检验次数      # 这里的循环 mmax=1，只执行一遍      while done != True and any(G) and ord <= m\_max:          ord1 = ord + 1          n\_edgetests[ord1] = 0          done = True          G1 = G.copy()          # 提取出所有边          ind = [(i, j)                for i in range(len(G))                for j in range(len(G[i]))                if G[i][j] == True                ]          # 遍历所有边          for x, y in ind:              if priorAdj is not None:                  if priorAdj[x,y]==1 or priorAdj[y,x]==1:                      continue              # 并行测试 MRCIT 条件是否独立              if G[y][x] == True:             # 如果反向边存在                  # nbrs 是 x 为起点的所有不是 x--y 的边的索引                  nbrs = [i for i in range(len(G1)) if G1[x][i] == True and i != y]                  if len(nbrs) >= ord:        # 如果上面的边数大于等于ord                      if len(nbrs) > ord:                          done = False                      # itertools.combinations(nbrs, ord) 的作用是返回 nbrs 的大小为 ord 的所有子集                      # nbrs\_S 也就是所有以 x 为起点的 ord 条边的终点索引                      for nbrs\_S in set(itertools.combinations(nbrs, ord)):                          n\_edgetests[ord1] = n\_edgetests[ord1] + 1                          pval = indepTest.fit(x, y, list(nbrs\_S), \*\*kwargs)                          if pval >= alpha:                              G[x][y] = G[y][x] = False                              sepset[x][y] = set(nbrs\_S)    # 将没有通过检验的一组数据存在 sepset                              break          ord += 1      return {'sk': np.array(G),'sepset': sepset,}  # 剪枝算法：  # 利用 MRCITs 测试每个父子对条件作用对所有其他直接原因的独立性，对避免选择难以确定的阈值而新增的"多余"的边进行修剪  # 参数含义:  #   indepTest 条件独立性检验类  #   dag 无向图  #   m\_max model\_para['step3\_maxr']=3  #   alpha 显著性标准 0.05 对应 95% 置信水平  #   priorAdj 预处理返回的矩阵  def pruning(indepTest, dag, m\_max, alpha=0.05, priorAdj=None, \*\*kwargs):      for r in range(1, m\_max):          dag1 = dag.copy()          edges = np.where(dag == 1)              # edges 是一个两个元素的元组，edges[0]是所有的行索引，[1]是列索引          for k in range(len(edges[0])):              xi, xj = edges[0][k], edges[1][k]   # 取出 xi--xj 这条边              if priorAdj is not None:                  if priorAdj[xi,xj] == 1 or priorAdj[xj,xi] ==1:                      continue                  if priorAdj[xi,xj] == -1:                      dag1[xi, xj] = 0                      continue              ifdelete = dag.copy()               # 删掉这条边的 dag              ifdelete = dag.copy()              ifdelete[xi, xj] = 0              # 增枝：在 DAG 条件约束下，会贪婪地添加边              considerz = []              # parent 是 ifdelete 里面所有以 xi 为终点的边的起点的索引              for parent in list(np.where(ifdelete[:, xi] == 1)[0]):                  # 如果这个起点仍然和 xj 联通                  if reachable(ifdelete, parent, xj): considerz.append(parent)              for parent in list(np.where(ifdelete[:, xj] == 1)[0]):                  if reachable(ifdelete, parent, xi): considerz.append(parent)              considerz = list(set(considerz))    # 去掉重复元素              # 剪枝：通过 MRCITs 以测试每个父子对条件作用对所有其他直接原因的独立性，如果相应的父子是条件独立的，则删除“多余”边              if len(considerz) > r:                  if len(considerz) == 1:                      z = considerz[0]                      pvalue = indepTest.fit(xi, xj, z, \*\*kwargs)                      if pvalue > alpha:                          dag1[xi, xj] = 0                          continue                  else:                      for nbrs\_z in set(itertools.combinations(considerz, r)):                          pvalue = indepTest.fit(xi, xj, list(nbrs\_z), \*\*kwargs)                          if pvalue > alpha:                              dag1[xi, xj] = 0                              break          dag = dag1.copy()      return dag |

**3.3.9.3 算法相关概念介绍**

在框架学习中涉及到的PC算法，是一种因果发现算法。它在因果马尔可夫假设和因果忠实性假设下学习有向无环图。

**3.4 代码复现环境配置和必要的包**

为了使我们的代码能够顺利运行，我们通过本节介绍环境配置和必要的包。

一、安装R包（R>=3.6.3）

1. 通过https://cran.r-project.org/下载并安装R。

2. 打开R，首先尝试在控制台运行以下代码：

install.packages("/ossfs/workspace/pai-algo-dev/momentchi2\_0.1.5.tar", repos=NULL, type="source")

如果运行成功，R package安装结束。

3. 如果运行失败：通过步骤 “R上方菜单栏 - 程序包 - 安装程序包... - momentchi2 - 确定”安装。

二、导入必要的python包

在requirements.txt文件中，给出了建议的包的版本：

|  |
| --- |
| numpy==1.19.4  sklearn\_pandas==2.0.3  rpy2==3.4.3  pandas==1.1.5  scipy==1.4.1  lightgbm==2.3.1  pygam==1.1.1  astropy==4.2  easydict==1.9  scikit\_learn==0.24.2 |

需要根据电脑实际的python版本，安装对应版本的包。

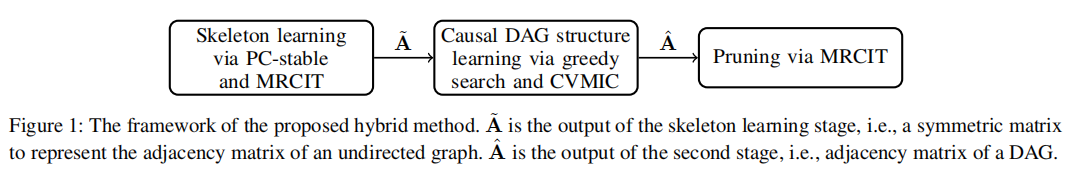
三、安装结束后，即可通过run.py运行代码，得到结果。

**3.5 代码复现结果展示**

我们100%完整复现了论文算法的各阶段算法、输出文件和各项评估参数。

在展示结果之前，我们首先介绍代码复现的具体工作内容，

原文的算法共有3个阶段，外加1类算法的各项评估参数的计算，如下图表：



|  |  |  |
| --- | --- | --- |
| **复现任务** | **原文展示** | **复现任务概述** |
| 1.第一阶段：框架学习 |  | 该任务作为算法的第1阶段，需要进行框架学习工作。  需要输出的结果，是表示无向图邻接矩阵的对称矩阵。 |
| 2.第二阶段：因果DAG结构学习 |  | 该任务作为算法的第2阶段，需要进行因果DAG结构学习工作。需要输出的结果，是DAG的邻接矩阵。 |
| 3.第三阶段：剪枝 |  | 该任务作为算法的第3阶段，需要进行剪枝。需要输出的结果，是剪枝后的结果。 |
| 4.关键参数计算与评估 |  | 该任务作为算法的关键参数计算与评估，需要对TP、TN、FP、FN、FD、MD、FPMD、Pre、Rec、SHD、运行时间等关键参数进行计算。 |

下面是我们的复现完成情况：

|  |  |  |
| --- | --- | --- |
| **复现任务** | **完成情况** | **结果展示方式** |
| 1.第一阶段：框架学习 | 成功复现 | 将结果输出到文件中“alram\_simulate\_skl.csv”，其中，与第二阶段的不同，已用黄色标出。（在本文3.5.2展示了清晰图片） |
| 2.第二阶段：因果DAG结构学习 | 成功复现 | 将结果输出到文件中“alram\_simulate\_dag2.csv”，其中，与第一阶段的不同，已用黄色标出；与第三阶段的不同，已用绿色标出；（在本文3.5.3展示了清晰图片） |
| 3.第三阶段：剪枝 | 成功复现 | 将结果输出到文件中“alram\_simulate\_dag.csv”，其中，与第二阶段的不同，已用绿色标出；（在本文3.5.4展示了清晰图片） |
| 4.关键参数计算与评估 | 成功复现 | 将结果输出到控制台中  （在本文3.5.5展示了清晰图片） |

共有2个输入数据文件，分别为：

·数据文件：alarm.csv，含有表示实际图（true graph）的数据。

·数据文件：alram\_simulate.csv，含有需要读取的仿真数据。

共有3个输出数据文件，分别为：

·数据文件：alram\_simulate\_skl.csv

框架学习阶段（第一环节）的输出，表示无向图邻接矩阵的对称矩阵。

·数据文件：alram\_simulate\_dag2.csv

因果DAG结构学习（第二环节）的输出，表示DAG的邻接矩阵。

·数据文件：alram\_simulate\_dag.csv

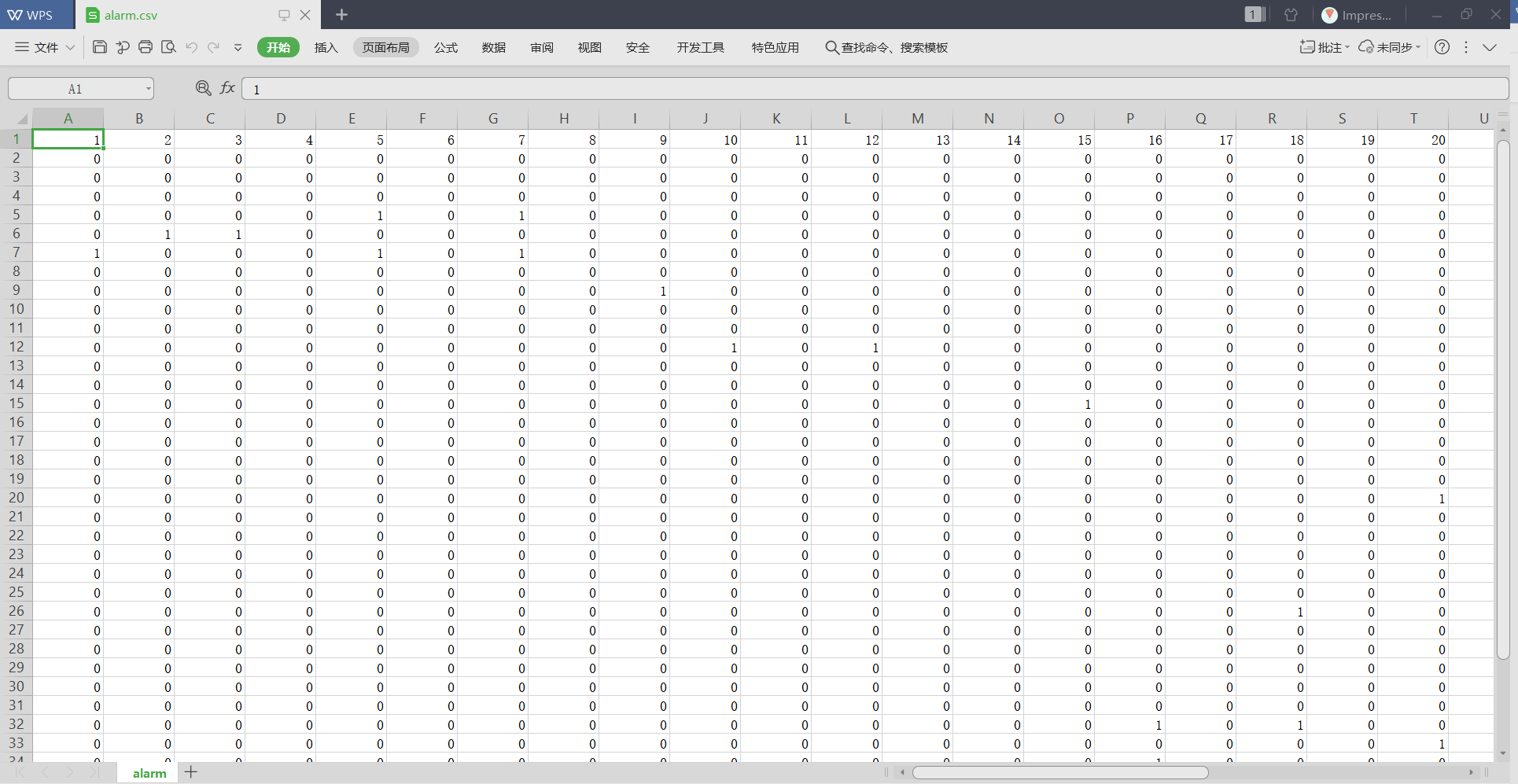
剪枝（第三环节）后的结果。

此外，除了上述三个文件外，还有控制台的输出，计算了算法的运行时间、以及TP、TN、FP、FN、SHD等关键参数。

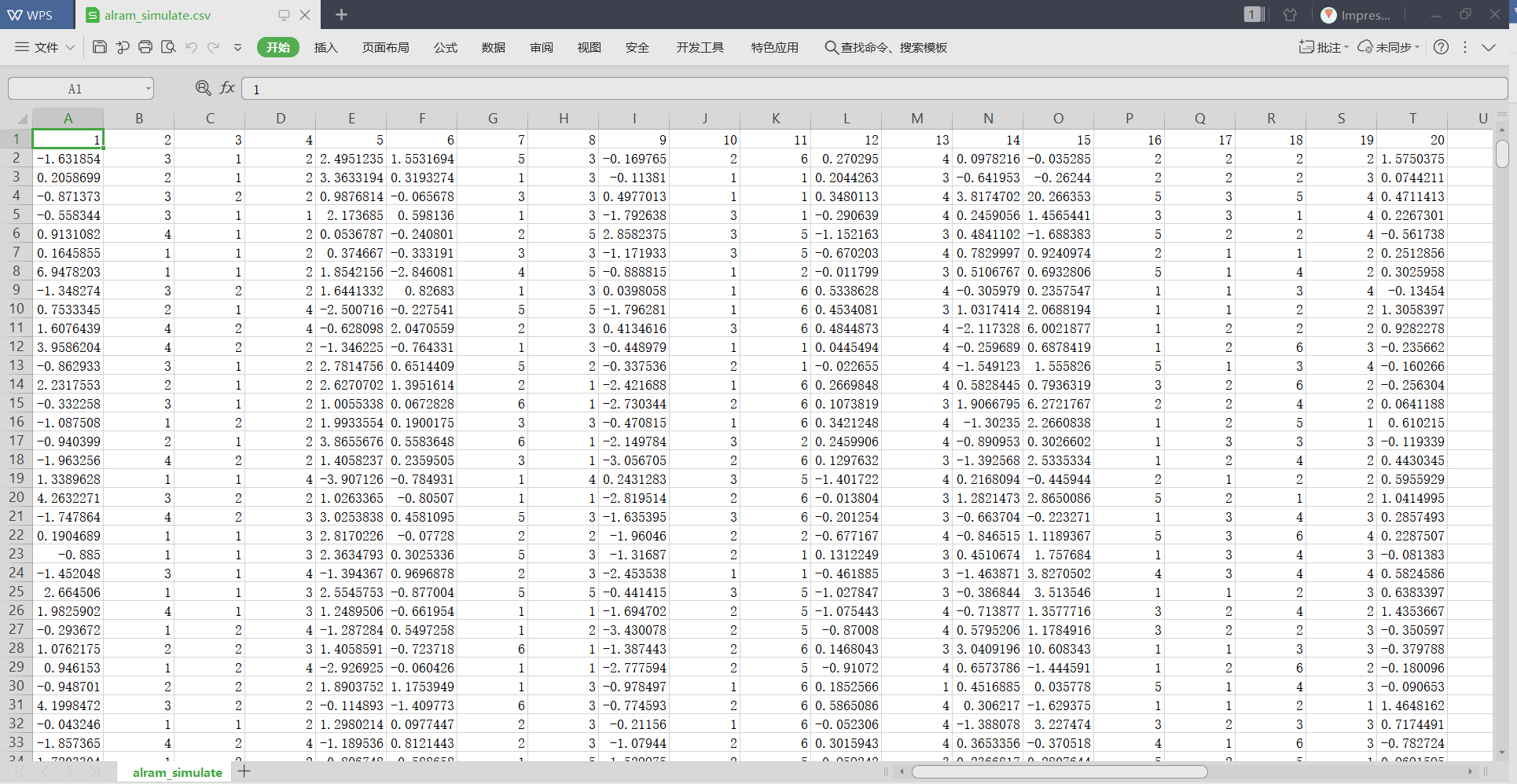
下面，我们分别展示了输入数据、输出数据的文件样例，并展示了连续的5次运行中，控制台的输出结果。

**3.5.1 原始输入数据**

alarm.csv文件：

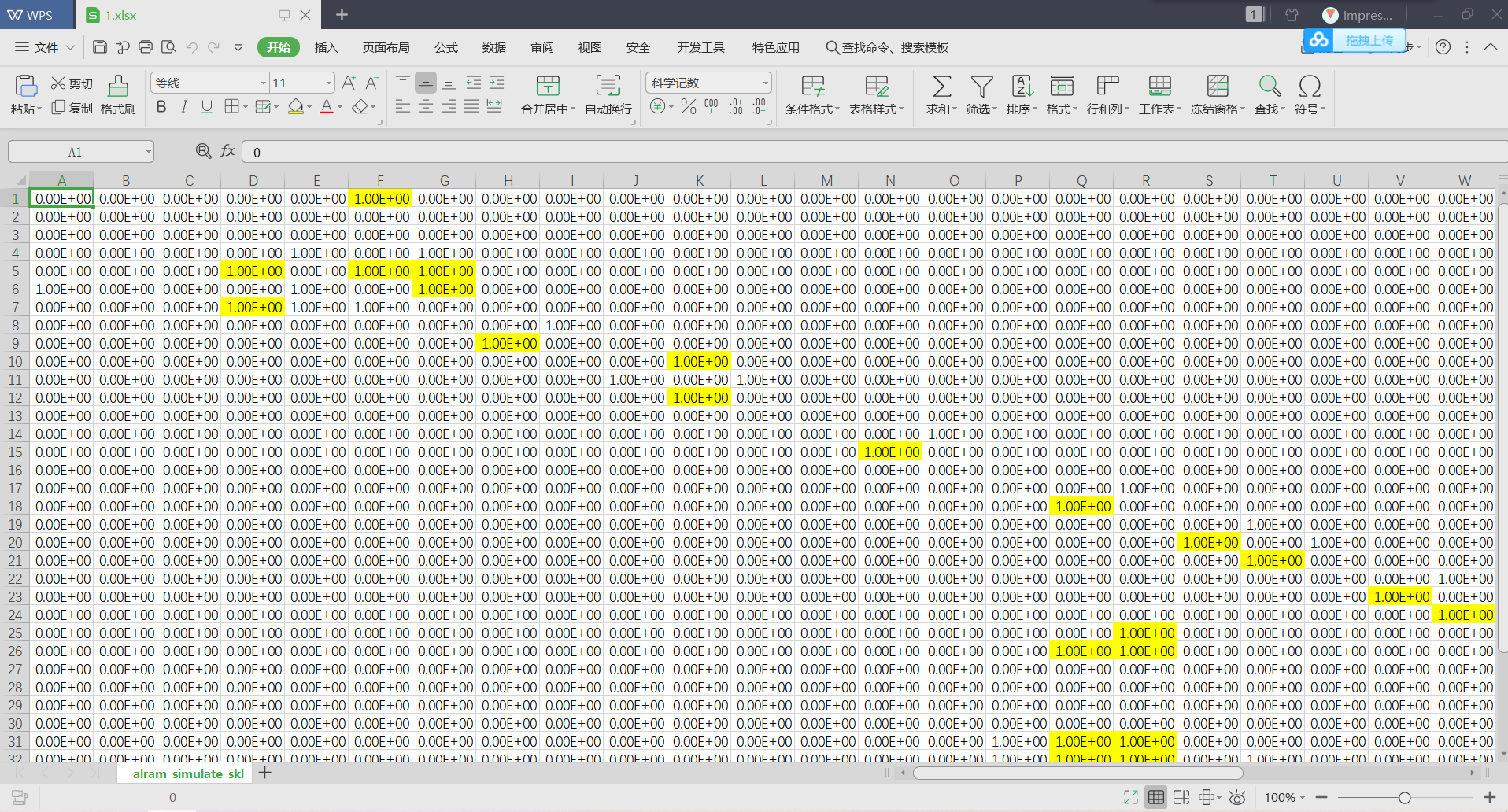


alram\_simulate.csv文件：



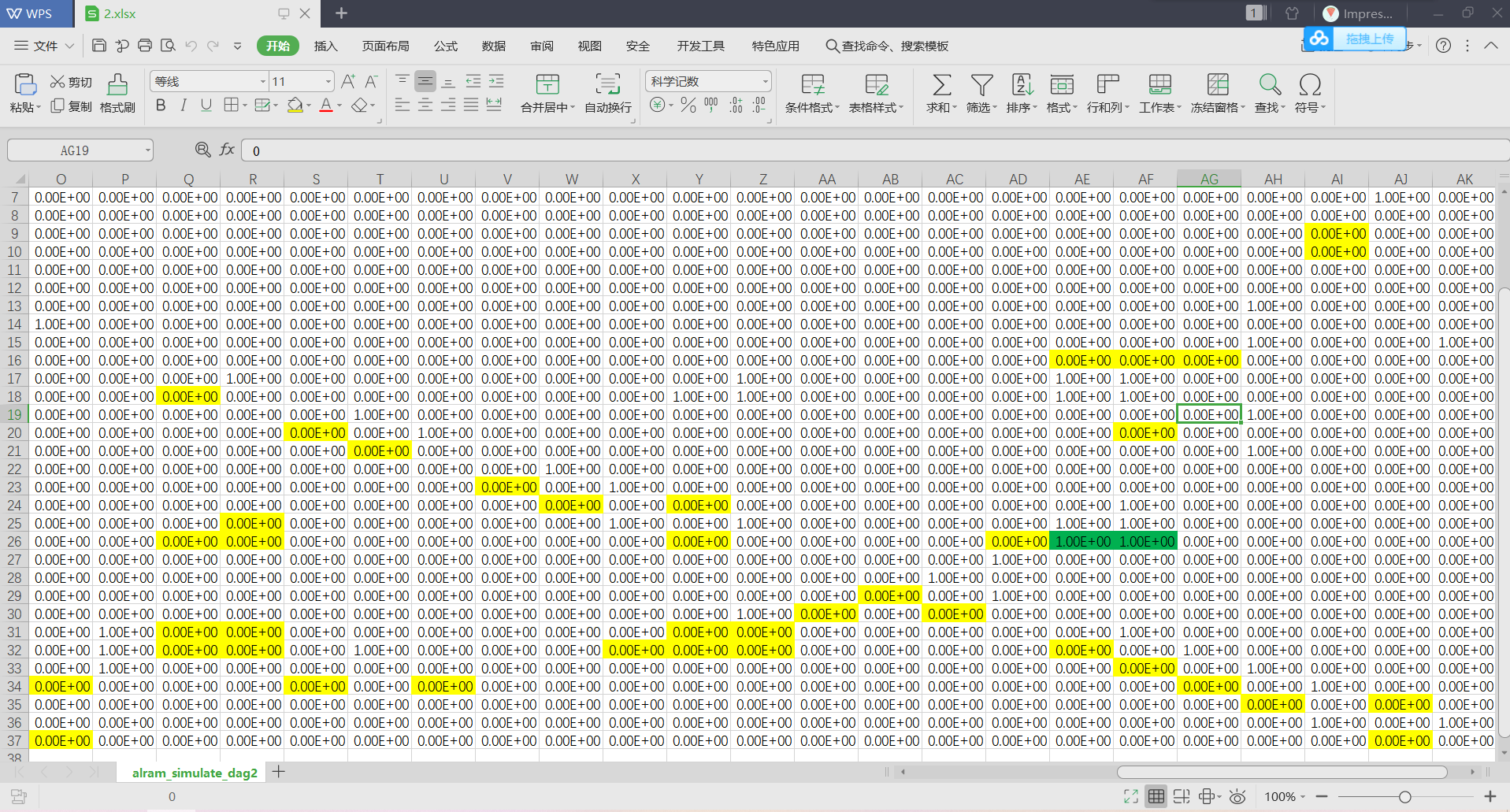
**3.5.2 算法第一阶段（框架学习）输出数据**

alram\_simulate\_skl.csv文件：表示无向图邻接矩阵的对称矩阵。



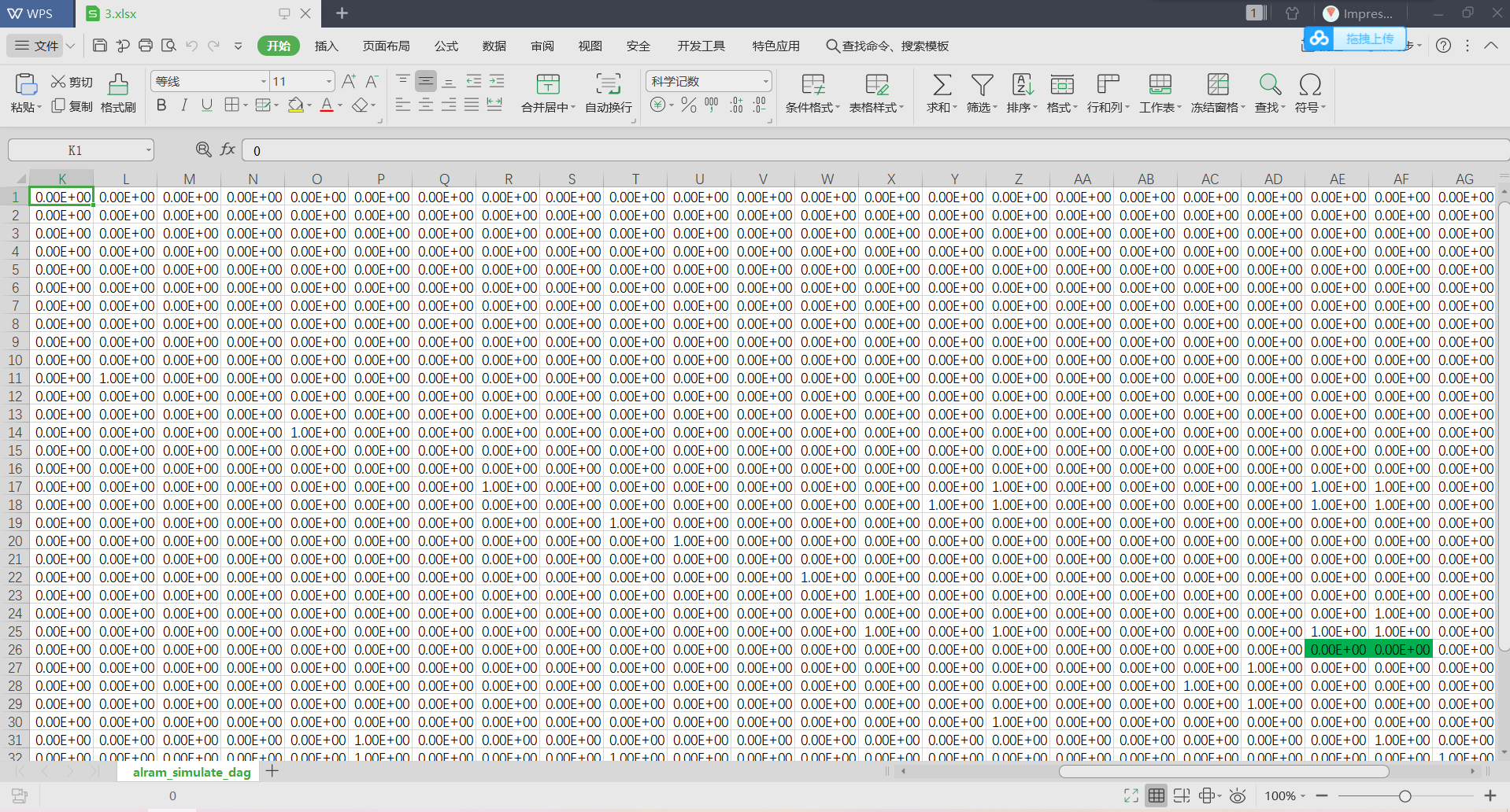
**3.5.3 算法第二阶段（因果DAG结构学习）输出数据**

alram\_simulate\_dag2.csv文件：表示DAG的邻接矩阵。



**3.5.4 算法第三阶段（剪枝）输出数据**

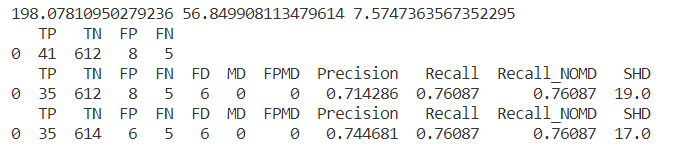
alram\_simulate\_dag.csv文件：表示剪枝后的结果。

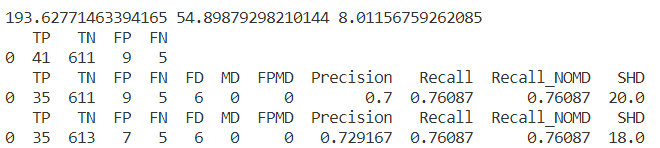


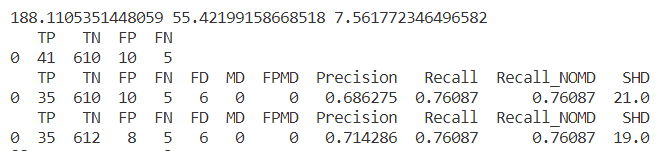
**3.5.5 运行时间与关键参数**

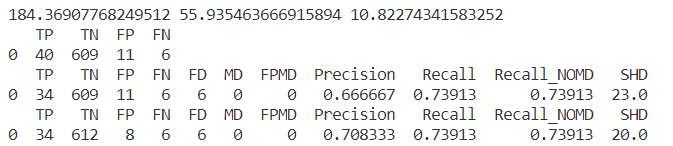
连续的5次运行中，控制台的输出结果：

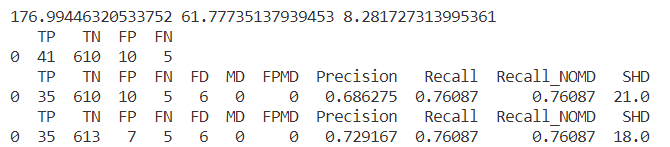
（注：每次运行需要约5-10分钟）











**3.6 团队代码与文档仓库**

我们本次作业全部代码和文档，已经放在：

<https://github.com/ChestnutSilver/Hybrid-Causal-Structure-Learning>

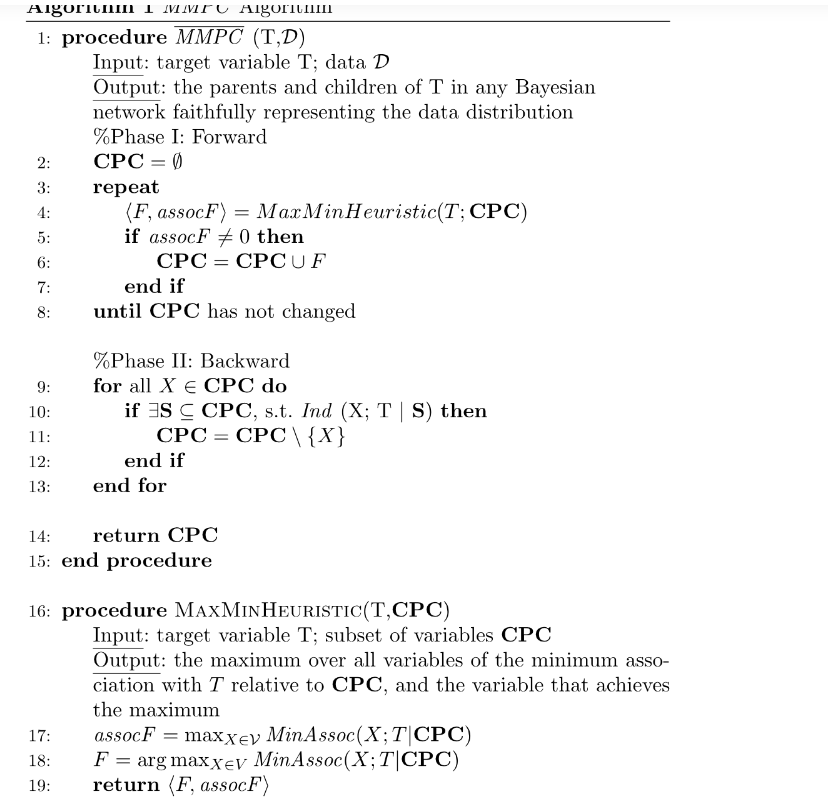
其中包括：

1.代码复现源代码；2.相关说明文档；3.算法输入与输出样例文件；

4.进一步探索的另一种相关算法的代码复现源代码；5.本作业文档。

第四部分 其他相关算法的分析与探讨

MMHC是现行的比较成功的一种混合贝叶斯结构学习算法，其主要思想是：先建立一个贝叶斯网络的骨架，再通过贪心算法确定最终结构。



上图为建立贝叶斯网络的伪代码。

第一步：前向，填充目标变量T的父子节点表CPC，用最大最小启发式方法，最大最小的意思是，在建立CPC表格时，使得在给定CPC表时，待选节点与目标变量最小联系最大化，于是得到初步CPC表。

第二步：后向，在CPC中，给定其中的变量S，若目标变量T与CPC表中元素X是独立的，则去除X（证明非父子关系）。

建立了CPC表后，然后该算法是利用了爬山搜索算法，在缩小了范围的搜索空间里进行搜索。但是缺点是仍然有陷入局部最优的可能

爬山算法是从当前的节点开始，和周围的邻居节点的值进行比较。 如果当前节点是最大的，那么返回当前节点，作为最大值（即山峰最高点）；反之就用最高的邻居节点来，替换当前节点，从而实现向山峰的高处攀爬的目的。如此循环直到达到最高点。

爬山算法是一种局部的启发式贪婪搜索过程，因为是贪婪式的，只能保证局部最大，因此不一定能有最优解。

另一种形式的伪代码：

函数Ind(X；T|Z)：如果X和T在Z条件下独立，则为真。

函数Assoc(X；T|Z)是给定Z时X和T的关联（依赖性）强度的估计。MMPC 实现中，独立性的条件测试是 G2 测试：如果测试返回的双尾 p 值小于标准的 5% 阈值，则为独立，否则为依赖的。G2 测试返回的 p 值越小，关联越高。也可以使用任何其他可靠的独立性测试和关联度量。

假设Ind(X；T|Z)⇔(Assoc(X；T|Z)=0)。

伪代码如下：

|  |
| --- |
| procedure MMPC (T,D)  Input: target variable T; data D  Output: the parents and children of T any Bayesian network faithfully representing the data distribution  %Phase I: Forward  CPC = Ø  repeat  <F, assoc)>= MaxMinHeuristic(T; CPC)  if assocF≠ 0 then  CPC= CPCUF  end if  until CPC has not changed  %Phase II: Backward  for all X∈CPC do  if 存在S包含于CPC, s.t. Ind(X;T|S) then  CPC= CPC \{X}  end if  end for  return CPC  end procedure |
| procedure MAXMINHEURISTIC(T,CPC)  Input: target variable T; subset of variables CPC  输入：目标变量T； 变量 CPC 的子集  Output: the maximum over all variables of the minimum association with T relative to CPC, and the variable that achievesthe maximum  输出：与 T 相关的最小关联的所有变量的最大值（相对于 CPC），以及达到最大值的变量  assocF = max x∈v MinAssoc(X;T|CPC)  F = argmax x∈v MinAssoc(X;T|CPC)  return (F, assocF)  end procedure |

MMHC的优点:由于是混合式学习方法，很显然，能够集两种典型算法的优点于一身(基于约束的方法，基于搜索评分的方法)。基于约束的方法的缺点是当面临大量的节点是，付出的运算代价很大。而搜索评分的缺点是搜索的空间也是巨大的。当两种方法融合在一起后，建立的基本的联系，然后缩小了搜索的范围，自然比以前的方法更好。

与论文中的CVMIC相比，虽然二者都能处理混合型变量，且在构建DAG时均采用了贪婪式搜索。MMHC采用了启发式函数的评分搜索，MMHC是基于评分搜索的贝叶斯网络结构学习算法，其效果不仅依赖于搜索算法, 也依赖于评分准则, 采用不同的搜索算法对学习效果具有不同的影响.而CVMIC对离散变量的概率通过频率计算，而连续变量的概率通过核密度估计（KDE）估计，每次增加一条潜在增益最大的边，最后通过MRCITs进行剪枝，以测试每个父子对条件作用对所有其他直接原因的独立性，如果相应的父子是条件独立的，则删除“多余”边。两者在构建DAG时十分的类似，两者也没有明显的优劣，不同情况下各有各的优势。

第五部分 论文算法的实际应用与意义

通过阅读文献，我们进一步了解了作者论文中，混合型数据的混合因果结构学习算法的实际应用与意义。

在作者的论文中，介绍了HCM算法在德国信用数据集上发挥的作用。它最初的目标是根据金融历史和个人状况的属性将人们划分为良好或不良信用风险。数据集中，包含21个变量，其中7个为连续变量，14个为分类变量，其中连续变量包括信贷金额、分期付款率；分类变量包括自有财产、婚姻状况和性别。

根据学习到的因果图，发现了信用风险的直接原因包括储蓄债券、信用历史、现有支票账户的状态、月持续时间等，其中大多数与金融有关。此外，还发现了金融和人口属性的分离是明确的，这两部分通过信贷目的和持续时间进行沟通，恢复的因果关系符合人们的共同理解和领域知识。

混合型数据的因果发现方法，在近些年来不断被提出，例如：一种面向金融事件的混合型因果关系发现方法与流程，为解决金融事件中因果关系的无法快速甄别和发现的问题，提供了一种面向金融事件的混合型因果关系发现方法。

该方法采用的步骤是：

s1.获取金融事件文本数据集；

s2.从所述金融事件文本数据集中抽取文本事件因果关系；

s3.从所述金融事件文本数据集中提取文本事件的关键词并利用百度指数上转化为随时间动态变化的搜索指数曲线；

s4.运用格兰杰因果性或因果传递熵算法确定两搜索指数曲线对应的文本事件对之间的因果关系；

s5.用最大似然估计方法融合步骤s2和s4分别得到的因果关系，从而生成金融事件因果关系发现。

还有，基于神经网络的混合数据的因果发现，这种方法在混合加性噪声模型(ANM-MM)的假设下,使用梯度下降法最优化改进的损失函数得到混合数据的抽象因果分布参数,然后将分布参数看作是原因变量和结果变量之间的隐变量,通过比较原因变量和分布参数之间的HilberSchmidt独立性来确定二元变量的因果方向。

我们不难感受到，对于混合型数据的混合因果关系的发现，在金融、医疗、环境、交通领域正在发挥着作用。例如，在医疗领域，基于NCICR的因果推断算法获得了病历资料中变量之间的因果关系网络,明确了因果方向,并对潜在变量有一定的识别。从因果推断的角度出发,对癌症预后治疗的影响因素进行因果关系建模,提高对判断患者预后的准确率,同时也为医学观测数据的因果识别提供了新的方法；在交通领域，创建能够接受多类型或混合类型数据输入的神经网络模型MDNN，有利于鉴别道路交通事故的多发地点，这将对降低交通事故数量产生巨大的有益影响。

总而言之，混合型数据的混合因果关系的发现，作为一个正在被研究的重要方向，越来越显示着它的意义与价值，它应用与生产生活的多个领域，能够为我们做出很多新的突破和贡献。

第六部分 作业总结与心得体会

在本次作业中，小组中的每一位同学都付出了特别多的努力。

从阅读原文，梳理原文的文章内容和结构，再到对文章算法可行性证明的理解与复述，再到最关键的代码复现、结果分析，最后是文中算法的优势评估、实际价值的探讨与作业文档的撰写，大家都尽己所能，为作业的完成做出了贡献。

最终，我们在大量查找分析资料、查找并阅读开源代码的基础上加以分析和理解，完成了本次代码复现工作，并将代码和结果呈现在本文当中。

这一次作业，为我们提供了一次新颖的作业经历，团队中的大多数成员，都是第一次从英文文献中一步步梳理结构，并进行代码复现的；这次作业与以往的课程作业不同，它对我们阅读查找文献资料、代码分析的能力进行了一次很有意义的启发式的帮助。对于这次作业，也可以说是收获满满。

特别值得一提的是，我们还对另一种现行的比较成功的混合贝叶斯结构学习算法进行了分析和复现，并探讨了该算法与论文所提出的算法的异同和优缺点。同时，我们将这种算法的源代码置于本作业的附录中。

在未来，我们还可以继续深入探讨混合型数据因果关系对我们生产生活的影响与意义，可以预想的是，它作为一种在生活中普遍存在的因果关系，处处影响着我们的生活；而对于其因果关系相关算法的研究，也一定会越来越发挥出重要的价值。

致 谢

向为我们提供这样一次新颖、有意义的作业经历的老师致谢！

同时，本次作业，也要感谢向作业组的每一位同学共同付出与倾力协作。

参考文献

[1]刘昕. 基于结构方程模型的因果分析算法研究及应用[D].电子科技大学,2019.

[2]王双成,李小琳,侯彩虹.用于因果分析的混合贝叶斯网络结构学习[J].智能系统学报,2007(06):82-89.

[3]李岩. 多类型数据的事件因果关系研究[D].国防科学技术大学,2014.

[4]耿家兴. 基于希尔伯特施密特独立性验证的因果推断方法研究[D].南华大学,2020.DOI:10.27234/d.cnki.gnhuu.2020.000235.

[5]张浩,郝志峰,蔡瑞初,温雯.基于互信息的适用于高维数据的因果推断算法[J].计算机应用研究,2015,32(02):382-385.

[6]王东明,陈都鑫.因果推断：起源和发展[J].控制工程,2022,29(03):464-473.DOI:10.14107/j.cnki.kzgc.20210524.

[7]王舒鸿,崔欣,姚守宇.统计相关还是真实因果?——基于“因果推断”的新兴研究范式[J].金融与经济,2018(08):21-30.DOI:10.19622/j.cnki.cn36-1005/f.2018.08.004.

[8]周琦. 基于归因指标的混合数据因果推断方法及应用[D].南华大学,2021.DOI:10.27234/d.cnki.gnhuu.2021.000352.

[9]耿家兴,万亚平,李洪飞.基于神经网络的混合数据的因果发现[J].计算机技术与发展,2020,30(05):26-31.

[10]张克. 基于混合数据神经网络的交通事故多发地点鉴别方法研究[D].重庆大学,2020.DOI:10.27670/d.cnki.gcqdu.2020.002964.

[11]马真真. 基于因果推断的肺癌患者生存时间预测方法研究[D].南华大学,2021.DOI:10.27234/d.cnki.gnhuu.2021.000720.

[12]谢晓芬,许欢,吴佳隆,郭冰,肖雄,周峻民,杨淑娟,赵星.因果推断方法在大气污染与人群健康效应研究中的应用[J].中华预防医学杂志,2021,55(11):1364-1370.

[13]蔡瑞初,陈薇,张坤,郝志峰.基于非时序观察数据的因果关系发现综述[J].计算机学报,2017,40(06):1470-1490.

[14]高帆. 基于贝叶斯网络的因果知识发现算法研究[D].国防科技大学,2020.DOI:10.27052/d.cnki.gzjgu.2020.000128.

附录

注：我们还对另一种现行的比较成功的，混合贝叶斯结构学习算法进行了分析和复现，以下附上该算法的复现代码。

（对于选定论文的代码复现，已经写在本文前面，不再在附录中赘述）。

1.accessory.py

|  |
| --- |
| from itertools import combinations, chain  import numpy as np  import pingouin as pg  from numba import njit  from scipy.stats import chi2, norm  from sklearn.linear\_model import LinearRegression  def powerset(iterable):      "powerset([1,2,3]) --> () (1,) (2,) (3,) (1,2) (1,3) (2,3) (1,2,3)"      s = list(iterable)      return chain.from\_iterable(combinations(s, r) for r in range(len(s)+1))  @njit(fastmath=True)  def bic(data, arities, cols):#启发式搜索时的一种评分方式，对应离散变量      strides = np.empty(len(cols), dtype=np.uint32)      idx = len(cols) - 1      stride = 1      while idx > -1:          strides[idx] = stride          stride \*= arities[cols[idx]]          idx -= 1      N\_ijk = np.zeros(stride)      N\_ij = np.zeros(stride)      for rowidx in range(data.shape[0]):          idx\_ijk = 0          idx\_ij = 0          for i in range(len(cols)):              idx\_ijk += data[rowidx, cols[i]] \* strides[i]              if i != 0:                  idx\_ij += data[rowidx, cols[i]] \* strides[i]          N\_ijk[idx\_ijk] += 1          for i in range(arities[cols[0]]):              N\_ij[idx\_ij + i \* strides[0]] += 1      bic = 0      for i in range(stride):          if N\_ijk[i] != 0:              bic += N\_ijk[i] \* np.log(N\_ijk[i] / N\_ij[i])      bic -= 0.5 \* np.log(data.shape[0]) \* (arities[cols[0]] - 1) \* strides[0]      return bic  def bic\_g(data, arities, cols):#启发式搜索时的一种评分函数，对应非离散变量      data = data.to\_numpy()      y = data[:, cols[0]]      if len(cols) == 1:          resids = np.mean(y) - y      else:          X = data[:, cols[1:]]          reg = LinearRegression().fit(X, y)          preds = reg.predict(X)          resids = y - preds      sd = np.std(resids)      numparams = len(cols) + 1  # include intercept AND sd (even though latter is not a free param)      bic = norm.logpdf(resids, scale=sd).sum() - np.log(data.shape[0]) / 2 \* numparams      return bic  def local\_score(data, arities, cols, score='default'):#根据输入选择评分函数      '''      :param weight: weight for data      :param data: numbered version of data set      :param cols: the index of node and its parents, the first element represents the index of the node and the following elements represent the indices of its parents      :param score\_function: name of score function, currently support bic, nal, bic\_g      :return: local score of node (cols[0]) given its parents (cols[1:])      '''      if len(data) == 0:          return np.nan      else:          if score == 'default':              score = 'bic\_g' if arities is None else 'bic'          try:              ls = globals()[score](data, arities, np.asarray(cols))          except Exception as e:              raise Exception('score function ' + str(                  e) + ' is undefined or does not fit to data type. Available score functions are: bic (BIC for discrete variables) and bic\_g (BIC for continuous variables).')          return ls  # statistical test  #在前向和后向建立PCP（candidate parents and children）时，需要进行变量间的独立性检测，以此来删减或加入边，此函数就是用来进行独立性检测  def independence\_test(p\_value, tar, pc, can, data, arities, varnames, test='g-test', threshold=0.05):      '''      statistical independence test      :param p\_value: a dictionary contains the maximum p-value of CI tests for each variable      :param pc: parents and children set of the target variable      :param can: candidate variable for the pc set of the target variable      :param data: input data (numpy array)      :param arities: number of distinct value for each variable      :param varnames: variable names      :param prune: whether use prune method      :param test: type of statistical test (currently support g-test)      :param threshold: threshold for statistical test to determine independence      :return: a dictionary contains the maximum p-value of CI test for each variable      '''      for can\_var in can:          if can\_var not in p\_value.keys():              p\_value[can\_var] = 0          for con in powerset(pc[0:-1]):              # avoid checking the separation set that has been checked in previous iterations              con = list(con)              if len(pc) != 0:                  con.append(pc[-1])              cols = np.array([varnames.index(x) for x in [tar, can\_var] + con])              if test == 'g-test':                  G, dof = it\_counter(data, arities, cols)                  p = chi2.sf(G, dof)                  p\_value[can\_var] = max(p, p\_value[can\_var])              elif test == 'z-test':                  # under construction                  r = pg.partial\_corr(data=data, x=tar, y=can\_var, covar=con)['r'][0]                  z = np.sqrt(data.shape[0] - len(con) - 3) \* np.arctanh(r)                  p = 2 \* min(norm.cdf(z), norm.cdf(-z))                  p\_value[can\_var] = max(p, p\_value[can\_var])              else:                  raise Exception('statistical test ' + test + ' is undefined, currently supported tests are: g-test')              if p\_value[can\_var] > threshold:                  break      return p\_value  @njit(fastmath=True)  def it\_counter(data, arities, cols):      strides = np.empty(len(cols), dtype=np.uint32)      idx = len(cols) - 1      stride = 1      while idx > -1:          strides[idx] = stride          stride \*= arities[cols[idx]]          idx -= 1      N\_ijk = np.zeros(stride)      N\_ik = np.zeros(stride)      N\_jk = np.zeros(stride)      N\_k = np.zeros(stride)      for rowidx in range(data.shape[0]):          idx\_ijk = 0          idx\_ik = 0          idx\_jk = 0          idx\_k = 0          for i in range(len(cols)):              idx\_ijk += data[rowidx, cols[i]] \* strides[i]              if i != 0:                  idx\_jk += data[rowidx, cols[i]] \* strides[i]              if i != 1:                  idx\_ik += data[rowidx, cols[i]] \* strides[i]              if (i != 0) & (i != 1):                  idx\_k += data[rowidx, cols[i]] \* strides[i]          N\_ijk[idx\_ijk] += 1          for j in range(arities[cols[1]]):              N\_ik[idx\_ik + j \* strides[1]] += 1          for i in range(arities[cols[0]]):              N\_jk[idx\_jk + i \* strides[0]] += 1          for i in range(arities[cols[0]]):              for j in range(arities[cols[1]]):                  N\_k[idx\_k + i \* strides[0] + j \* strides[1]] += 1      G = 0      for i in range(stride):          if N\_ijk[i] != 0:              G += 2 \* N\_ijk[i] \* np.log(N\_ijk[i] \* N\_k[i] / N\_ik[i] / N\_jk[i])      dof = max((arities[cols[0]] - 1) \* (arities[cols[1]] - 1) \* strides[1], 1)      return G, dof  # convert the dag to bnlearn format  #将最终得到的有向图转为贝叶斯网络格式  def to\_bnlearn(dag):      output = ''      for var in dag:          output += '[' + var          if dag[var]['par']:              output += '|'              for par in dag[var]['par']:                  output += par + ':'              output = output[:-1]          output += ']'      return output |

2.hc.py

|  |
| --- |
| from rpy2.robjects.packages import importr  from accessory import local\_score, to\_bnlearn  base, bnlearn = importr('base'), importr('bnlearn')  def check\_cycle(vi, vj, dag):#检查得到的DAG中是否有环      # whether adding or orientating edge vi->vj would cause cycle. In other words, this function check whether there is a direct path from vj to vi except the possible edge vi<-vj      underchecked = [x for x in dag[vi]['par'] if x != vj]      checked = []      cyc\_flag = False      while underchecked:          if cyc\_flag:              break          underchecked\_copy = list(underchecked)          for vk in underchecked\_copy:              if dag[vk]['par']:                  if vj in dag[vk]['par']:                      cyc\_flag = True                      break                  else:                      for key in dag[vk]['par']:                          if key not in checked + underchecked:                              underchecked.append(key)              underchecked.remove(vk)              checked.append(vk)      return cyc\_flag  def hc(data, arities, varnames, pc=None, score='default'):#一种混合算法，爬山算法和启发式搜索。爬山算法是一种贪婪式的搜索方法，所以得到的图只是局部最优。启发式函数搜索是一种通过启发式函数进行评分，增加或删除边，缺点是搜索空间巨大      '''      :param data: the training data used for learn BN (numpy array)      :param arities: number of distinct value for each variable      :param varnames: variable names      :param pc: the candidate parents and children set for each variable      :param score: score function, including:                     bic (Bayesian Information Criterion for discrete variable)                     bic\_g (Bayesian Information Criterion for continuous variable)      :return: the learned BN (bnlearn format)      '''      if score == 'default':          score = 'bic\_g' if arities is None else 'bic'      # initialize the candidate parents-set for each variable      candidate = {}      dag = {}      cache = {}      for var in varnames:          if pc is None:              candidate[var] = list(varnames)              candidate[var].remove(var)          else:              candidate[var] = list(pc[var])          dag[var] = {}          dag[var]['par'] = []          dag[var]['nei'] = []          cache[var] = {}          cache[var][tuple([])] = local\_score(data, arities, [varnames.index(var)], score)      diff = 1      while diff > 0:          diff = 0          edge\_candidate = []          for vi in varnames:              # attempt to add edges vi->vj              for vj in candidate[vi]:                  cyc\_flag = check\_cycle(vi, vj, dag)                  if not cyc\_flag:                      par\_sea = tuple(sorted(dag[vj]['par'] + [vi]))                      if par\_sea not in cache[vj]:                          cols = [varnames.index(x) for x in (vj, ) + par\_sea]                          cache[vj][par\_sea] = local\_score(data, arities, cols, score)                      diff\_temp = cache[vj][par\_sea] - cache[vj][tuple(dag[vj]['par'])]                      if diff\_temp - diff > 1e-10:                          diff = diff\_temp                          edge\_candidate = [vi, vj, 'a']              for par\_vi in dag[vi]['par']:                  # attempt to reverse edges from vi<-par\_vi to vi->par\_vi                  cyc\_flag = check\_cycle(vi, par\_vi, dag)                  if not cyc\_flag:                      par\_sea\_par\_vi = tuple(sorted(dag[par\_vi]['par'] + [vi]))                      if par\_sea\_par\_vi not in cache[par\_vi]:                          cols = [varnames.index(x) for x in (par\_vi, ) + par\_sea\_par\_vi]                          cache[par\_vi][par\_sea\_par\_vi] = local\_score(data, arities, cols, score)                      par\_sea\_vi = tuple([x for x in dag[vi]['par'] if x != par\_vi])                      if par\_sea\_vi not in cache[vi]:                          cols = [varnames.index(x) for x in (vi, ) + par\_sea\_vi]                          cache[vi][par\_sea\_vi] = local\_score(data, arities, cols, score)                      diff\_temp = cache[par\_vi][par\_sea\_par\_vi] + cache[vi][par\_sea\_vi] - cache[par\_vi][                          tuple(dag[par\_vi]['par'])] - cache[vi][tuple(dag[vi]['par'])]                      if diff\_temp - diff > 1e-10:                          diff = diff\_temp                          edge\_candidate = [vi, par\_vi, 'r']                  # attempt to delete edges vi<-par\_vi                  par\_sea = tuple([x for x in dag[vi]['par'] if x != par\_vi])                  if par\_sea not in cache[vi]:                      cols = [varnames.index(x) for x in (vi, ) + par\_sea]                      cache[vi][par\_sea] = local\_score(data, arities, cols, score)                  diff\_temp = cache[vi][par\_sea] - cache[vi][tuple(dag[vi]['par'])]                  if diff\_temp - diff > 1e-10:                      diff = diff\_temp                      edge\_candidate = [par\_vi, vi, 'd']          if edge\_candidate:              if edge\_candidate[-1] == 'a':                  dag[edge\_candidate[1]]['par'] = sorted(dag[edge\_candidate[1]]['par'] + [edge\_candidate[0]])                  candidate[edge\_candidate[0]].remove(edge\_candidate[1])                  candidate[edge\_candidate[1]].remove(edge\_candidate[0])              elif edge\_candidate[-1] == 'r':                  dag[edge\_candidate[1]]['par'] = sorted(dag[edge\_candidate[1]]['par'] + [edge\_candidate[0]])                  dag[edge\_candidate[0]]['par'].remove(edge\_candidate[1])              elif edge\_candidate[-1] == 'd':                  dag[edge\_candidate[1]]['par'].remove(edge\_candidate[0])                  candidate[edge\_candidate[0]].append(edge\_candidate[1])                  candidate[edge\_candidate[1]].append(edge\_candidate[0])      dag = bnlearn.model2network(to\_bnlearn(dag))      return dag |

3.mmpc.py

|  |
| --- |
| from accessory import independence\_test  import operator  # forward phase of MMPC  def mmpc\_forward(tar, pc, can, data, arities, varnames, prune, test, threshold):      #也就是伪代码中所说的前向建立CPC的过程，在MMPC算法的第1阶段利用Max-min策略启发式地使变量依次进入目标节点T的候选父子节点集CPC      #Max-Min策略启发式的选择使变量T相对于以候选父子节点集CPC为条件集时的最小依赖最大的变量.      #除了候选父子节点集CPC节点集中的所有其他变量在给定以CPC的子集为条件集时都与目标变量T独立时, 第1阶段停止.      '''      forward phase of mmpc      :param tar: target variable      :param pc: parents and children set of the target variable      :param can: candidate variable for the pc set of the target variable      :param data: input data (numpy array)      :param arities: number of distinct value for each variable      :param varnames: variable names      :param prune: whether use prune method      :param test: type of statistical test (currently support g-test)      :param threshold: threshold for statistical test to determine independence      :return: pc set and candidate pc set      '''      # run until no candidate variable for current variable      p\_value = {}      for can\_var in can[tar]:          p\_value[can\_var] = 0      while can[tar]:          # run conditional independence test between each candidate varialbe and target variable          p\_value = independence\_test(p\_value, tar, pc, can[tar], data, arities, varnames, test, threshold)          # print(p\_value)          # update pc set and candidate set          pc, can = update\_forward(p\_value, tar, pc, can, prune, threshold)      return pc, can  # backward phase of MMPC  def mmpc\_backward(tar, pc, can, data, arities, varnames, prune, test, threshold):      #即伪代码所描述的后向过程，移去CPC中第1阶段进入的本不该有的变量, 如果目标变量T的候选父子节点集CPC中存在变量X, 使Ind (X;T|S) 成立, 其中S⊆CPC, 则将变量X从CPC中移去.      '''      backward phase of mmpc      :param tar: target variable      :param pc: parents and children set of the target variable      :param can: candidate variable for the pc set of the target variable      :param data: input data (numpy array)      :param arities: number of distinct value for each variable      :param varnames: variable names      :param prune: whether use prune method      :param test: type of statistical test (currently support g-test)      :param threshold: threshold for statistical test to determine independence      :return: pc set and candidate pc set      '''      # transfer the variable in pc set to candidate set except the last one      can[tar] = pc[0: -1]      pc\_output = []      pc\_output.append(pc[-1])      can[tar].reverse()      while can[tar]:          # run conditional independence test between each candidate varialbe and target variable          p\_value = independence\_test({}, tar, pc\_output, can[tar], data, arities, varnames, test, threshold)          # update pc set and candidate set          pc\_output, can = update\_backward(p\_value, tar, pc\_output, can, prune, threshold)      return pc\_output, can  def update\_forward(p\_value, tar, pc, can, prune, threshold):#将PCP更新成在前向得到的结果      '''      add the variable with lowest p-value to pc set and remove it from the candidate set      :param p\_value: a dictionary contains the maximum p-value of CI tests for each variable      :param tar: target variable      :param pc: parents and children set of the target variable      :param can: candidate variable for the pc set of the target variable      :param prune: whether use prune method      :param threshold: threshold for statistical test to determine independence      :return: updated pc set and candidate variables      '''      sorted\_p\_value = sorted(p\_value.items(), key=operator.itemgetter(1))      if sorted\_p\_value[0][1] < threshold:          pc.append(sorted\_p\_value[0][0])          can[tar].remove(sorted\_p\_value[0][0])          p\_value.pop(sorted\_p\_value[0][0], None)      # remove independent variables from candidate set      independent\_can = [x for x in sorted\_p\_value if x[1] > threshold]      for ind in independent\_can:          can[tar].remove(ind[0])          p\_value.pop(ind[0])          # prune the target variable from the candidate set of the candidate variable if they are independent          if prune:              if tar in can[ind[0]]:                  can[ind[0]].remove(tar)      return pc, can  def update\_backward(p\_value, tar, pc, can, prune, threshold):#将CPC更新成后向得到的      '''      pc and candidate set update function for backward phase      :param p\_value: a dictionary contains the maximum p-value of CI tests for each variable      :param tar: target variable      :param pc: parents and children set of the target variable      :param can: candidate variable for the pc set of the target variable      :param prune: whether use prune method      :param threshold: threshold for statistical test to determine independence      :return: updated pc set and candidate variables      '''      # initialise the output candidate set      can\_output = []      # signal of import variable      sig\_import = 1      for can\_var in can[tar]:          if p\_value[can\_var] <= threshold:              if sig\_import:                  pc.append(can\_var)                  sig\_import = 0              else:                  can\_output.append(can\_var)          else:              if prune:                  if tar in can[can\_var]:                      can[can\_var].remove(tar)      can[tar] = can\_output      return pc, can  # symmetry check for pc set  def symmetry(pc):#检查pc的对称性，由于 PC 关系应该是对称的，破坏对称性表明存在误报。 MMPC 检查是否所有 X ∈ MMPC(T, D) 的 T ∈ MMPC(X, D) ;如果不是这种情况，它会从其输出中删除 X。      for var in pc:          pc\_remove = []          for par in pc[var]:              if var not in pc[par]:                  pc\_remove.append(par)          if pc\_remove:              for par in pc\_remove:                  pc[var].remove(par)      return pc |

4.mmhc.py

|  |
| --- |
| from mmpc import mmpc\_forward, mmpc\_backward, symmetry  from hc import hc  import time  import numpy as np  def mmhc(data, test = None, score = None, prune = True, threshold = 0.05):#mmhc总的部分，包含了mmpc前向，后向和对称检验的过程。      #注意，代码中的can就是伪代码中所说的CPC(candidate parents and children)      '''      mmhc algorithm      :param data: input data (pandas dataframe)      :param test: type of independence test (currently support g-test (for discrete data), z-test (for continuous data))      :param score: type of score function (currently support bic (for both discrete and continuous data))      :param prune: whether use prune method      :param threshold: threshold for CI test      :return: the DAG learned from data (bnlearn format)      '''      # initialise pc set as empty for all variables      pc = {}      # initialise the candidate set for variables      can = {}      for tar in data:          can[tar] = list(data.columns)          can[tar].remove(tar)      # preprocess the data      varnames = list(data.columns)      if all(data[var].dtype.name == 'category' for var in data):#全是离散型变量则用g\_test，评分函数用bic,否则用z\_test和bic\_g          arities = np.array(data.nunique())          data = data.apply(lambda x: x.cat.codes).to\_numpy()          if test is None:              test = 'g-test'          if score is None:              score = 'bic'      elif all(data[var].dtype.name != 'category' for var in data):          arities = None          if test is None:              test = 'z-test'          if score is None or score == 'bic':              score = 'bic\_g'      else:          raise Exception('Mixed data is not supported.')      # run MMPC on each variable      start = time.time()      for tar in varnames:          # forward phase          pc[tar] = []          pc[tar], can = mmpc\_forward(tar, pc[tar], can, data, arities, varnames, prune, test, threshold)          # backward phase          if pc[tar]:              pc[tar], can = mmpc\_backward(tar, pc[tar], can, data, arities, varnames, prune, test, threshold)      # check the symmetry of pc set      # when the number of variables is large, this function may be computational costly      # this function can be merged into the pruning process during forward and backward mmpc by transmitting the whole      # pc set into mmpc\_forward and mmpc\_backward      pc = symmetry(pc)      print('MMPC phase costs %.2f seconds' % (time.time() - start))      # run hill-climbing      start = time.time()      dag = hc(data, arities, varnames, pc, score)      print('HC phase costs %.2f seconds' % (time.time() - start))      return dag |

5.evaluation.py

|  |
| --- |
| from rpy2.robjects.packages import importr  base, bnlearn = importr('base'), importr('bnlearn')  # compute the F1 score of a learned graph given true graph  def f1(dag\_true, dag\_learned):#比较真实结果和学习得到的结果      '''      :param dag\_true: true DAG      :param dag\_learned: learned DAG      :return: the F1 score of learned DAG      '''      compare = bnlearn.compare(bnlearn.cpdag(dag\_true), bnlearn.cpdag(dag\_learned))      return compare[0][0] \* 2 / (compare[0][0] \* 2 + compare[1][0] + compare[2][0]) |