机器学习系统期末Project

因果推断框架搭建

小组成员：石青芸、杨乐宜、游宇菲

CONTENTS

PART.01

背景介绍&基本功能

因果推断的任务目标 、基本流程和常用方法

Step1：问题定义

目标：明确要研究什么因果关系

举例：对电商平台的客户数据，研究"客户高度不满意"是否导致"平台客户流失"

基本概念：处理效应，指接受处理vs不接受处理的差异，用来评判因果效应的影响

处理变量 (Treatment)
我们想研究其因果效应的变量

e.g. "客户高度不满意" 对应的变量HighlyDissatisfied (0/1)

结果变量 (Outcome)
我们关心的结果

e.g. “平台客户流失”对应的变量Churn (0/1)

协变量
可能影响处理结果的其他变量

e.g. 年龄、性别、购买历史等其他变量

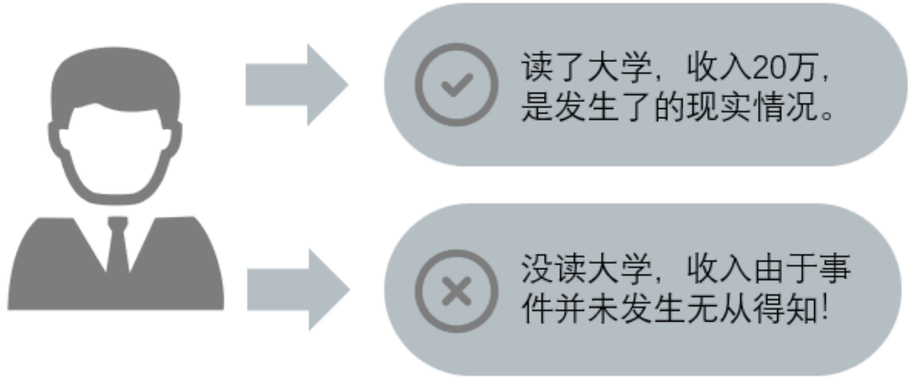
Step2：因果识别

识别的目标：希望我们观察到的关联就等于真实的因果效应。

核心困难：

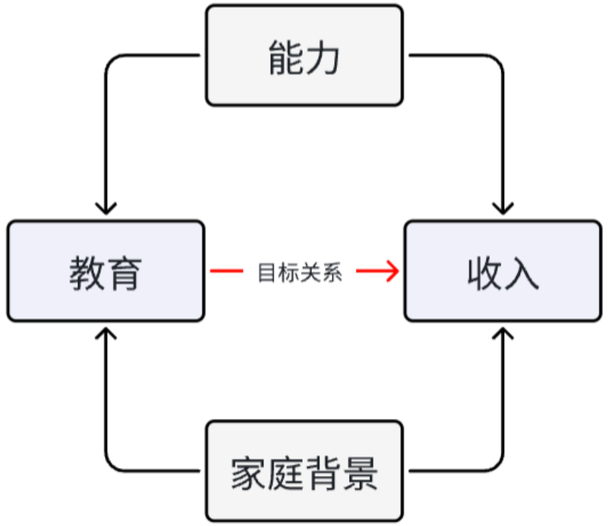
- 反事实问题：永远无法同时观察到同一个人在接受处理和不接受处理两种状态下的结果。
- 混杂因素干扰：观察到的关联，部分是真实因果效应，部分是混杂因子造成的虚假关联。

解决方案：找到一组阻断所有后门路径的变量，阻断后剩下的关联就是因果关系。{能力, 家庭背景}↓



后门路径创造了"虚假关联"：

- 高能力的人更可能读大学，也更可能有高收入
- 好家庭的孩子更可能读大学，也更可能有高收入



Step3：因果估计

目标：基于识别结果估计因果效应的大小。我们这里采取了逆概加权作为基本方法：

步骤1：估计倾向性评分

以前面的例子为例，这里需要用逻辑回归预测"接受教育的概率"作为倾向性评分

- 例如：小明(25岁,男,富裕家庭,高智商) → 读大学概率 = 0.8

```
ps = LogisticRegression().fit(X, w).predict_proba(X)[:,1]
```

步骤2：重新加权，使处理组和对照组在协变量上就"平衡"了，就像随机实验一样！

- 处理组：倾向性评分高的人（本来就容易读大学，协变量的影响大）权重小
- 对照组：倾向性评分高但没读大学的人（意外情况，协变量的影响小）权重大

```
ate = np.average(y[w==1], weights=wt_t[w==1]) - np.average(y[w==0], weights=wt_c[w==0])
# ate是平均处理效应
```

Step4：因果验证

随机混杂因子测试

1. 添加一个随机生成的变量
2. 重新估计因果效应
3. 如果效应大幅改变 → 原估计可能有问题
4. 如果效应基本不变 → 原估计较可信

子样本稳定性测试

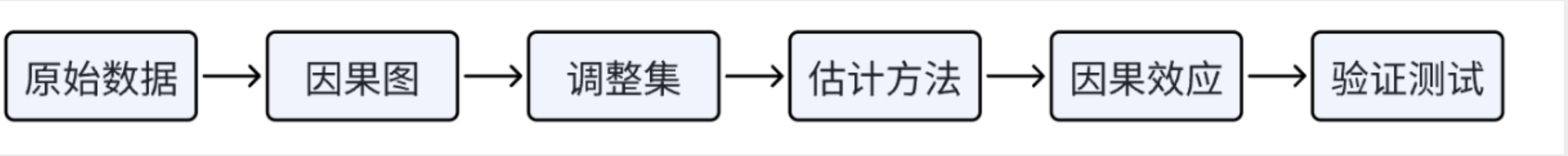
1. 用80%的数据重新估计
2. 比较新旧估计结果
3. 真实的因果效应应该在不同子样本中保持稳定

PART.02

亮点设计

在基础框架之上，我们还针对框架功能、模型选择等方面进行了改进。

完整的因果推断流程框架



因果图构建核心功能：

```
class CausalDAG:
    def __init__(self, dot_str: str):
        """从DOT字符串创建因果有向无环图"""

    @classmethod
    def from_dot(cls, dot_str: str):
        """从DOT格式创建因果图"""

    def to_png(self, path: str):
        """导出图形为PNG格式"""
```

识别调整集：

```
def select_backdoor(dag: CausalDAG, treat
    """
    选择后门调整集
    Args:
        dag: 因果DAG
        treat: 处理变量
        outcome: 结果变量
        observed_vars: 观测到的变量集合
    """
```

反驳测试：

```
def add_random_confounder(model, **kw):
    """添加随机混杂因子的反驳测试"""

def subset_refuter(model, frac=0.8, **kw):
    """子集反驳测试"""
```

```
def refute(self, method='random_common', **kw):
    """反驳测试入口"""
```

倾向性评分方法：

```
class PropensityScoreWeighting:
    def estimate(self, X, w, y, bootstrap_rounds=200, alpha=0.05):
        """倾向性评分加权估计因果效应"""
```

功能诊断：

```
def diagnostics(self, method='improved_ps_weighting', **kw):
    """
    运行诊断检查
    """
```

基于机器学习的效应估计法

1. 学习器 ： 倾向性评分方法使用逻辑回归，只能进行线性建模。用机器学习模型来估计因果效应。

S-Learner：单一模型策略
训练一个模型f即可预测处理效应ATE = f(X, T=1) - f(X, T=0)

T-Learner：双模型策略
考虑到了处理组和对照组的关系模式不同，因此分别训练一个模型

X-Learner：改进的T-Learner
在T-Learner的基础上，训练直接预测处理效应的模型，并加权组合

R-Learner：残差学习
直接学习处理效应函数，而不是先学结果再算差异

```
def estimate_effect(self, method='ps_weighting', **kw):
    """
    估计因果效应

    Parameters:
    -----------
    method : str
        估计方法，可选:
        - 'ps_weighting': 原始倾向评分加权
        - 'improved_ps_weighting': 改进的倾向评分加权
        - 's_learner': S-Learner元学习器
        - 't_learner': T-Learner元学习器
        - 'x_learner': X-Learner元学习器
        - 'r_learner': R-Learner元学习器
        - 'double_ml': 双重机器学习
        - 'causal_forest': 因果森林
    """
```

基于机器学习的效应估计法

2. 双重机器学习 (Double ML): 同时去除两个"偏差源"

- 用ML估计：E[Y|X] (结果模型)
- 用ML估计：E[T|X] (倾向性评分模型)
- 计算残差：Y_res = Y - E[Y|X], T_res = T - E[T|X]
- 因果效应 = Cov(Y_res, T_res) / Var(T_res)

3. 因果森林：估计个体层面的异质性处理效应

- 每个人可能有不同的处理效应
- 年轻人：教育效应 = +50%收入
- 中年人：教育效应 = +30%收入
- 老年人：教育效应 = +10%收入

```
def estimate_effect(self, method='ps_weighting', **kw):
    """
    估计因果效应

    Parameters:
    -----------
    method : str
        估计方法，可选:
        - 'ps_weighting': 原始倾向评分加权
        - 'improved_ps_weighting': 改进的倾向评分加权
        - 's_learner': S-Learner元学习器
        - 't_learner': T-Learner元学习器
        - 'x_learner': X-Learner元学习器
        - 'r_learner': R-Learner元学习器
        - 'double_ml': 双重机器学习
        - 'causal_forest': 因果森林
    """
```

模型比较功能及优化

传统的Bootstrap置信区间计算（有放回抽样模拟估计值分布）中，每次重采样都会重新进行完整的模型选择，导致：

- 计算时间指数增长（例如：100轮bootstrap × 4个模型 × 5折交叉验证 = 2000次模型训练）
- 在某些情况下出现无法中断的"伪死循环"

我们提供三种策略：

- full：每轮都重新模型选择（与传统等价，但保留为最高精度选项）
- lightweight：减少模型交叉检验次数至3（节省大量计算）
- reuse：直接复用已存在模型，无需重新选择（最快）

```
def compare_methods(self, methods=None, **kw):
    """
    比较多种估计方法

    Parameters:
    -----------
    methods : list or None
        要比较的方法列表，如果为None则使用所有方法
    **kw : dict
        传递给估计器的额外参数

    Returns:
    --------
    results : dict
        每种方法的估计结果
    """
```

PART.03

结果展示

数据集介绍

我们选用了ecommerce-customer-churn-analysis-and-prediction（电商客户流失分析与预测）数据集作为实例，目标是通过已有客户行为和属性数据，预测哪些客户可能会流失（Churn），探索影响流失的关键因素。

数据集本身提供了合适的标签（ChurnFlag）和干预变量（SatisfactionFlag / ComplainFlag），适合做策略评估和因果推断：

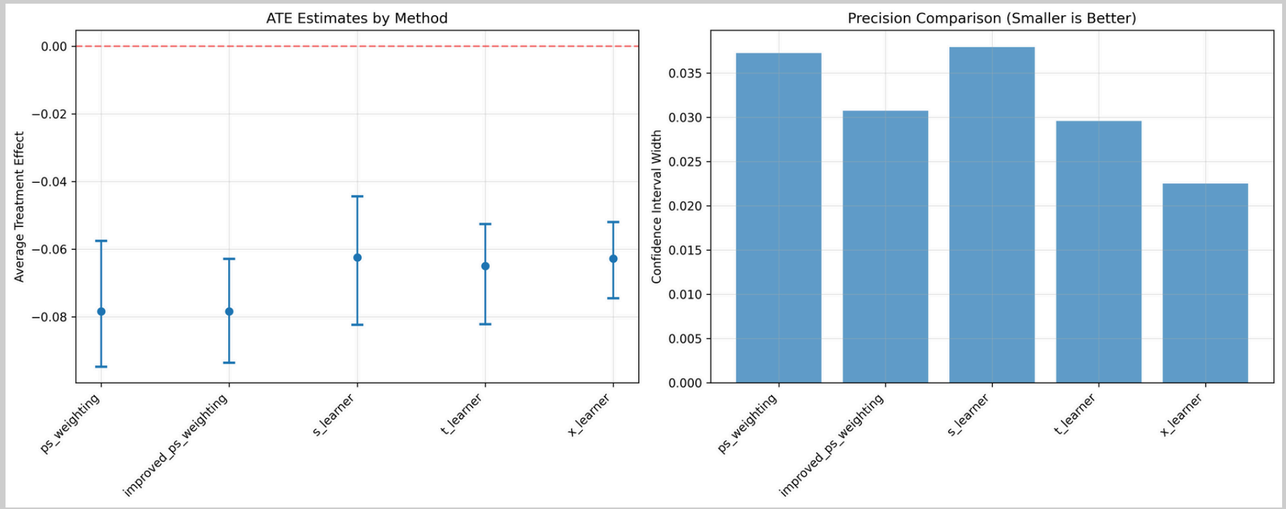
- 目标变量 ChurnFlag 是流失（1）或未流失（0）
- 处理变量包括：
  - SatisfactionFlag：表示用户是否满意，作为“干预”变量
    - SatisfactionScore = 1 → HighlyDissatisfied=1（表示极度不满意）
    - SatisfactionScore = 2~5 → HighlyDissatisfied=0（表示相对满意）
  - ComplainFlag：表示用户是否有投诉，也是一个潜在的“处理”变量

实例运行结果

```python
cm = CausalModel(df, dag, treatment="HighlyDissatisfied", outcome="Churn")
adj = cm.identify_effect()            # 取出后门集
ate, ci = cm.estimate_effect()         # PSW 估计
print("ATE:", ate, "CI:", ci)
print("Random common cause refute:", cm.refute('random_common'))
print("Subset refute:", cm.refute('subset', frac=0.8))
```

Python

```
ATE: -0.07826779088996595 CI: (np.float64(-0.09868405898102761), np.float64(-0.059235883107810736))
Random common cause refute: {'orig': np.float64(-0.07826779088996595), 'new': np.float64(-0.07845397854044216)}
Subset refute: {'orig': np.float64(-0.07826779088996595), 'new': np.float64(-0.08132372871648685)}
```



- 反直觉的负效应: ATE为负值(-0.0783)表明"高度不满意"实际上降低了客户流失率约7.83个百分点
- 统计显著性：95%置信区间为(-0.0987, -0.0592)不包含0，说明这个负效应在统计上是显著的
- 可能的解释: 高度不满意的客户可能更倾向于投诉或寻求解决方案，而不是直接流失，公司可能对表达不满的客户提供了更好的客户服务或补偿措施
- 稳健性检验: 两种检验方法的结果都与原始估计相近，说明结果相对稳健

THANK YOU

感谢您的观看