中山大學
SUN YAT-SEN UNIVERSITY

中山大学计算机学院

人工智能

本科生实验报告

(2022学年春季学期)

课程名称：Artificial Intelligence

教学班级	20 级计科一班	专业（方向）	计算机科学与技术
学号	20337056	姓名	李昊伟

补充说明：第三版，完成了测例3

第四版， 修改了实验报告

■ 给定的知识图谱所有三元组的一阶逻辑如下（测例3）：

edge(a,b),edge(b,c),edge(c,d),edge(b,a), connected(a,b), connected(b,c), connected(c,d), connected(b,a), connected(a,c),connected(a,d), connected(a,a), connected(b,d), connected(b,b)

一. 实验题目

编写程序，实现FOIL（First Order Inductive Learner）算法，对给定的知识图谱和目标谓词进行规则学习，并得到新的以目标谓词为关系的事实（用一阶逻辑表示）。

二. 实验内容

1. 算法原理

FOIL算法基于背景知识、正例和反例归纳结论。首先，根据背景知识生成可能作为条件的一阶逻辑表达式。 然后，对于这些表达式，逐个加入条件，测试覆盖的正例和反例数。利用这两个数字计算出信息增益。每一轮增加信息增益最大的一个表达式。每轮结束删除覆盖不了的正例，循环多轮，直到生成的表达式集合可以覆盖所有正例，不覆盖任何反例。

- FOIL算法是怎样work的？ 在助教的PPT中提到，FOIL算法属于归纳学习的范畴。首先，我们需要提出一些假设。在这里，所有的背景知识都可以作为假设的来源，我们可以假设，这些背景知识与目标之间存在因果的关系。然后，就像孟德尔设计实验一样，我们从背景知识中获得假设归纳出的结论，与正

例反例相对照。　　直观地讲，如果一个假设它覆盖的反例远远多于正例，那它显然是荒谬的。在这里，我们用信息增益来表示它是否是“荒谬”的程度。它的表达式是这样的：

$$\hat{m_+} \times [log_2 \frac{\hat{m_+}}{\hat{m_+} + \hat{m_-}} - log_2 \frac{m_+}{m_+ + m_-}]$$

算法循环多轮，每一轮增加信息增益最大的一个假设。每轮结束删除覆盖不了的正例，直到生成的假设序列可以覆盖所有正例，不覆盖任何反例，我们就认为，我们得到了理性正确的假设。

- FOIL算法有什么值得探讨的思路?
    - 从一般到特殊的推理思路。从一般到特殊指的是：对目标谓词或前提约束谓词中的变量依据背景知识赋予具体值。这个实现起来比较困难，我是建立所有背景知识出现过的姓名库，然后通过排列组合，生成所有可能的姓名组合。比如{a , b , c , d}要赋值给一个含有x , y , z的假设，就利用全排列公式

      $$A_4^3$$

      生成依次付给x , y , z的序列，再到背景知识当中寻找正确的赋值序列(用字典{x:"?" , y:"?" , z:"?"}表示) , 然后，就可以容易地找到它们的结论是否在正例、反例中了。
    - 归纳学习的思路。从例子中学习，就叫做归纳学习。反之，利用已经有的假设有目的地设计实验，叫做演绎。归纳连同演绎，是自然科学领域中的重要方式方法。

2 .输入说明

```
INPUT:(一次输入一个样例)

4
Father(Jack,Dell)
Father(Dell,Stephen)
Grandfather(Jack,Stephen)
Father(Dell,Seth)
Grandfather(x,y)

19
zero(A)
succ(A,B)
succ(B,C)
succ(C,D)
succ(D,E)
succ(E,F)
succ(F,G)
succ(G,H)
succ(H,I)
succ(I,J)
pred(B,A)
pred(C,B)
pred(D,C)
pred(E,D)
pred(F,E)
pred(G,F)
pred(H,G)
```

```
pred(I,H)
pred(J,I)
pred(x,y)

5
Sibling(Ann,Mike)
Couple(David,James)
Mother(James,Ann)
Mother(James,Mike)
Father(David,Mike)
Father(x,y)
```

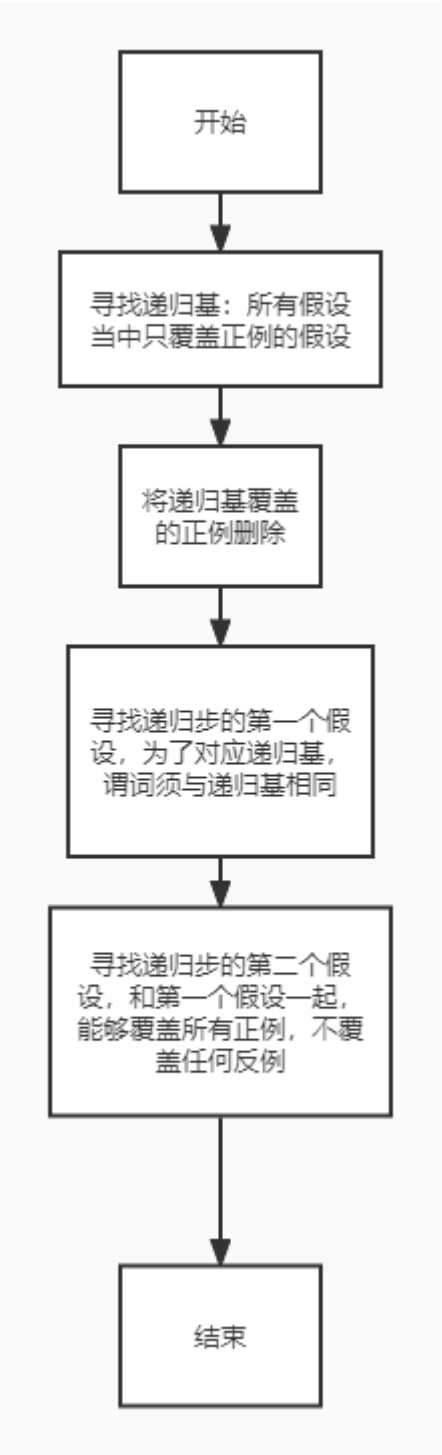
3. 代码详细解释(非递归版本)

```
# 解题函数
def solve(names , rules_set , target , back_ground , negative_examples ,
positive_examples):
    result = []
    # m01表示加条件之前的正例数，m12表示加条件之后的正例数
    # m02表示加条件之前的反例数，m12表示加条件之后的反例数
    # 初始化m01、m02参数，准备迭代
    m11 = len(positive_examples)
    m12 = len(negative_examples)
    m01 = m11
    m02 = m12
    old_gain = -9999
    while(1):
        new_rule = None
        # 保存最大信息增益对应的的m11、m12,最后判断推出条件用
        saved_m11 = 0
        saved_m12 = 0
        # 保存最大信息增益对应的正例、反例,因为要删除覆盖不了的正例
        saved_pos = positive_examples.copy()
        saved_neg = negative_examples.copy()
        # 开始筛选候选规则
        for it in rules_set:
            pos , neg = one_rule(names , target , result , it , back_ground ,
negative_examples , positive_examples)
            # 获得覆盖的正反例,得到对应的m11,m12
            m11 = len(pos)
            m12 = len(neg)
            # 计算信息增益
            gain = cal_gain(m01 , m02 , m11 , m12)
            # 判断是否是更大的信息增益
            if gain > old_gain:
                new_rule = it
                old_gain = gain
                saved_m11 = m11
                saved_m12 = m12
                saved_pos = pos
```

```
                saved_neg = neg
        positive_examples = saved_pos
        negative_examples = saved_neg
        result.append(new_rule)
        # 如果生成的表达式集合可以覆盖所有正例，不覆盖任何反例,可以退出
        if saved_m12 == 0 and saved_m11 == len(positive_examples):
            break
    return result
```

```
# 逐个解析每一个表达式
def one_rule(names , target , rules , rule , back_ground , negative_examples ,
positive_examples):
    new_rules = rules.copy()
    new_rules.append(rule)
    # 假定加入了这条规则,在背景中找到对应的事实以赋值
    examples_in_background = find_in_background(names , new_rules , back_ground)
    # 寻找覆盖的正反例
    pos_covered , neg_covered = find_covered(names , target ,
examples_in_background , negative_examples , positive_examples)
    new_pos_covered = []
    new_neg_covered = []
    # 去除重复的,因为推出号前面可能有多条规则,不同规则可能覆盖相同的正反例
    for it in neg_covered:
        if it not in new_neg_covered:
            new_neg_covered.append(it)
    for it in pos_covered:
        if it not in new_pos_covered:
            new_pos_covered.append(it)
    return new_pos_covered , new_neg_covered
```

4. 创新与改进：递归专用版FOIL



思路: (只能实现最简单的递归) 首先，找到递归基。递归基不一定覆盖所有正例，但一定不覆盖负例。 然后， 寻找递归步，限制第一个条件在背景知识生成的谓词中找，因为必须对应递归基。因为是递归，第二个条件的谓词和目标谓词相同，且找到x,y的关系。

寻找递归基

```
def find_base(names , rules_set , target , back_ground , negative_examples ,
positive_examples):
    for it in rules_set:
        result = []
        pos , neg = one_rule(names , target , result , it , back_ground ,
negative_examples , positive_examples)
```

```
            if len(neg) == 0:
                return it , pos
```

寻找递归步

```
    for it in rules_set_afterbase:
        # 递归步的假设不能和递归基一样，而且不能直接是目标结论，否则就成了A(x,y)->A(x,y)
        if it != base_rule and it != (target , 'x' , 'y'):
            new_rule_set.append(it)
    rules_set_afterbase = new_rule_set
    # solve函数同非递归版本
    result_recurse = solve(names , rules_set_afterbase , target , back_ground + positive_examples , negative_examples , positive_examples)
```

5. 结果

```
PS D:\人工智能\E07\6_20337056_v2\code> python .\foil.py
4
Father(Jack,Dell)
Father(Dell,Stephen)
Grandfather(Jack,Stephen)
Father(Dell,Seth)
Grandfather(x,y)
================================================
Father(x,z) ∧ Father(z,y) -> Grandfather(x,y)
Grandfather(Jack,Seth)
Grandfather(Jack,Stephen)
================================================
PS D:\人工智能\E07\6_20337056_v2\code>
```

```
================================================
succ(y,x) -> pred(x,y)
pred(B,A)
pred(C,B)
pred(D,C)
pred(E,D)
pred(F,E)
pred(G,F)
pred(H,G)
pred(I,H)
pred(J,I)
================================================
PS D:\人工智能\E07\6_20337056_v2\code>
```

```
5
Sibling(Ann,Mike)
Couple(David,James)
Mother(James,Ann)
Mother(James,Mike)
Father(David,Mike)
Father(x,y)
================================================
Couple(x,z) ∧ Mother(z,y) -> Father(x,y)
Father(David,Ann)
Father(David,Mike)
================================================
PS D:\人工智能\E07\6_20337056_v2\code>
```

```
connected(b,b)
¬connected(c,a)
¬connected(c,b)
¬connected(c,c)
¬connected(d,a)
¬connected(d,b)
¬connected(d,c)
¬connected(d,d)
connected(x,y)
================================================
edge(x,y) -> connected(x,y) ; edge(x,z) ∧ connected(z,y) -> connected(x,y)
================================================
PS D:\人工智能\E07\6_20337056_v3\code>
```

三. 实验感想

这个实验乍看起来原理不是很难，但是实现起来却不容易，主要是细碎的地方太多，需要自己动手实现许多细节，而且一不留神就错了。但是，通过动手实现FOIL算法,更好的理解了将规则总结出来的原理，代码能力也提高了。