复现第二周进度(4.1-4.7)

第一周基本完成了所有不需要日志数据的工作(semantic analysis、部分correlation Mining工作)，这周学长提供了数据，正好周末清明假期，没有其他事情，于是开始着手后面的部分。

学长的要求：假设智能家居平台是闭源的，我们拿不到自动化规则，测试的时候也假设不知道规则，就按假设检验的方法，先假设所有的可能性，再利用事件日志验证，最后和我们设置规则（作为ground truth）做比对

> 也就是一组不知道任何规则，需要假设生成所有自动化规则
>
> 一组已知自动化规则，不用假设
>
> 最后将这两种情况对比

因为已知自动化规则的较为简单一点，所以先从第二种情况着手。

这周完成了：

1.创建了已知规则、设备信息、等比较基础的前置工作

这块需要把给的数据和HAWatcher中的流程提前颗粒度对齐一下。。

2.Correlation Mining的Hypothetical Correlation Generation部分

这部分远没有之前自己想象的简单，需要考虑许多细节。因此也花了不少时间

3.**Correlation Mining的Hypothesis Testing部分**（即用日志验证所有假设相关性）

这部分最耗时（实现这个最耗时，这部分的程序跑起来也最耗时哈哈哈），因为这部分逻辑较为复杂，同时还要处理一些特殊情况，很多都是实现时才想到的。

同时花了一些时间理解里面的假设检验部分（翻了翻概率论的书、查了查资料、问了问gpt）

4.**Correlation Refining部分**

之前不理解作者加这个模块的原因，当自己亲自实现时才体会到为什么，**本质上这块就是前面Hypothesis Testing遗漏的小问题的查漏补缺**，这些小问题只有当你亲自走到这一步时才会注意到。

这部分根据提供的数据的特殊性我在方法上做出了一些修改，具体细节见后。

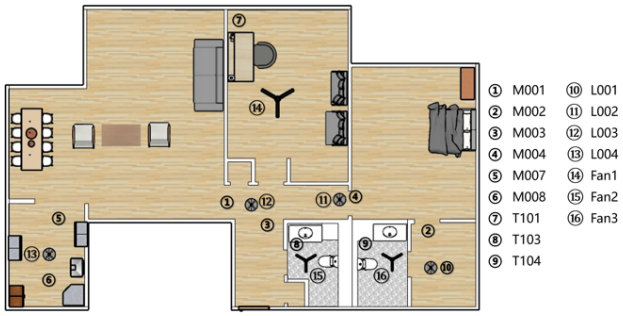
这周的代码量大概四五百行这样子。

1.数据格式分析

注：可能影响结果的参数：时间间隔d、前几个属性被认为相关、温度范围、假设检验部分

Evaluation

- Data generation: Dataset 2
  - User behavior log: HH125 apartment from WSU CASAS datasets
    - Devices: 8 Motion Sensors (M), 2 Door Sensor (D), 6 Temperature Sensors (T)
    - Two month



#	IF	Then
1	**M002** ON	**L001** ON
2	**M002** OFF	**L001** OFF
3	**M004** ON	**L002** ON
4	**M004** OFF	**L002** OFF
5	**M001** ON or **M003** ON	**L003** ON
6	**M001** OFF and **M003** OFF	**L003** OFF
7	**M007** ON or **M008** ON	**L004** ON
8	**M007** OFF and **M008** OFF	**L004** OFF
9	**T101** >= 32 ℃	**Fan1** ON
10	**T101** < 31 ℃	**Fan1** OFF
11	**T103** >= 25 ℃	**Fan2** ON
12	**T103** < 24 ℃	**Fan2** OFF
13	**T104** >= 25 ℃	**Fan3** ON
14	**T104** < 24 ℃	**Fan3** OFF

41

- WSU CASAS数据集是华盛顿州立大学（Washington State University, 简称WSU）为研究如环境辅助生活、智能环境和健康监测等领域而设计的一系列智能家居活动数据集。WSU CASAS数据集中的“HH125”公寓指的是从标记为“HH125”的智能家居环境中收集的特定数据集。
- 数据格式：日期（空格）时间戳（空格）设备（三空格）属性值/日期（空格）时间戳（空格）设备属性值

  前者时间戳后有空格的代表trigger;

  后者时间戳后没有空格，代表action
- 值得思考的点：对应规则8：

  2013-03-08 17:55:23.748960 M008 OFF
  ....

  ....

  2013-03-08 17:56:47.009220 M007 OFF
  2013-03-08 17:56:47.018836 L004 OFF
- 学长的要求：假设智能家居平台是闭源的，我们拿不到自动化规则，测试的时候也假设不知道规则，就按假设检验的方法，先假设所有的可能性，再利用事件日志验证，最后和我们设置规则（作为ground truth）做比对

  也就是一组已知规则，假设所有规则

  一组已知规则，不用假设

  最后将这两种情况对比
- 有什么设备？：8 Motion Sensors (M), 2 Door Sensor (D), 6 Temperature Sensors (T),4 lights(L),3 fans(Fan)

  总计23个设备，M/D/T做trigger，L/Fan做action

  8 Motion Sensors (M):

  M001

M002

M003

M004

M007

M008

M011

M022

2 Door Sensor (D):

D001

D002

6 Temperature Sensors (T):

T101

T103

T104

T105

T106

T102

4 lights(L):

L001

L002

L003

L004

3 fans(Fan):

Fan1

Fan2

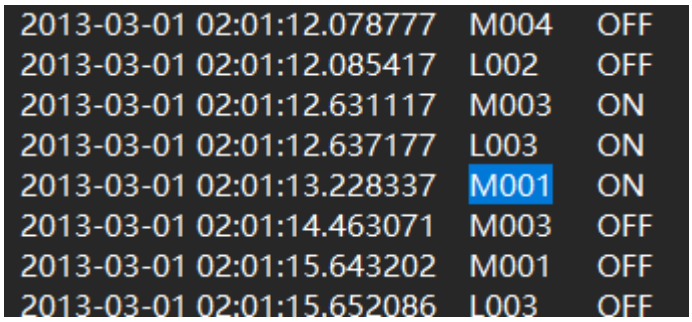
Fan3

2.流程分析

- 先做已知规则的测试和代码编写更合适
- 几个与原文不一样的点：
    - Semantic Analysis：
        - **有没有conditon?**

初步设想，对于M007 ON **and** MO08 ON可以将前者设为trigger，后者设为condion，或前者condion后者trigger

对于M007 ON **or** MO08 ON，可以拆成两个rule(错误的，下面这种情况在后续会被认为不正确)



correlation Mining

- Prepossessing Event Logs

  > 有两个原因需要预先处理事件日志:**1)原始事件日志受到重复传感器读数的干扰。**例如，一些电表定期报告类似(但略有波动)的读数。**2)器件的数值读数不能并入逻辑计算。**因此，我们设计了一个预处理方案，以消除冗余和数字到二进制转换。对于生成数值读数的每个设备，我们将整个训练数据集中的读数相加，并计算其平均值μ和标准差σ。落在[μ−3σ，μ + 3σ]范围之外的读数被排除为极值(即**3 -sigma规则**[64])然后，我们将**Jenks自然断续分类算法**[49]4应用于剩余的读数，并将其**分类为"低"或"高"。**接下来，对于每个设备的给定属性，(例如，连续的EIlluminance高)。**现在，关于设备的相同属性的两个时间上相邻的事件具有相反的值**

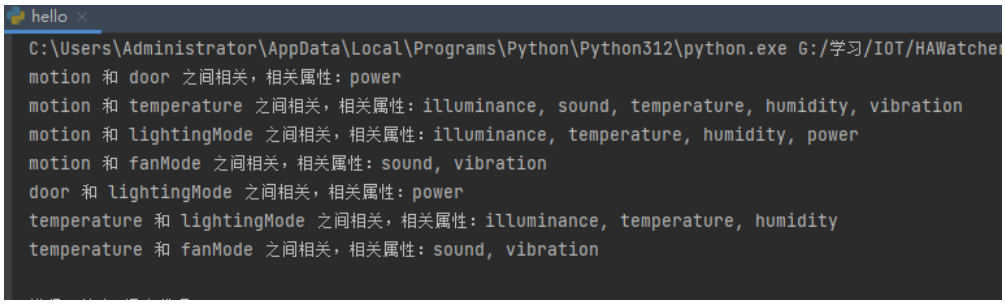
  分析数据可得，这些HAWatcher日志存在的问题本数据都没有。原因：

  本数据的日志只有变化时才会上传，

  并且，有数值读数的只有温度，温度没有异常值(除非自己插入)

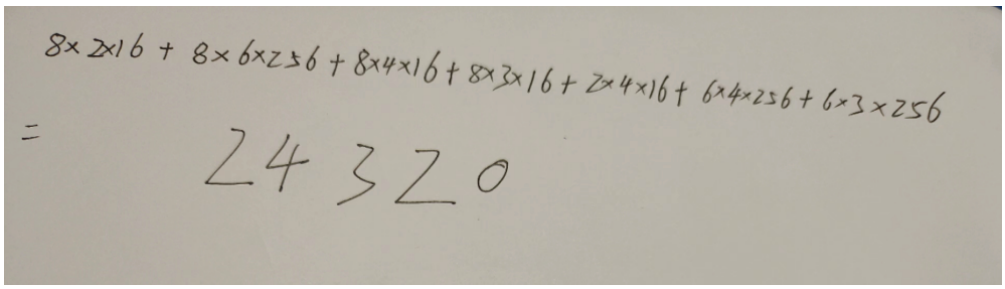
- Hypothetical Correlation Generation
  - 很明显不需要那么多的属性了，设备只有五种，每种只有一个属性，所以只有五个属性
  - 属性少了，所以计算前几个属性相关（原来是10个）也要改变，比如1或2
  - 没有switch属性
  - For user activity channel correlations, 只有motion，而motion属性在计算是已经和所有属性有关，所以不用特殊处理
  - 因为没有conditon，所以虚拟运动传感器也不需要考虑了
  - **相关性生成上，一个比较大的问题是温度**，因为按原文将数值分类为低或高，比较好处理，但是CASAS无法将温度进行分类，所以须保留数值，比如'<24'、'>=32'，所以这块会生成非常多的假设相关性，**这里假设可能的温度范围是20-35度**

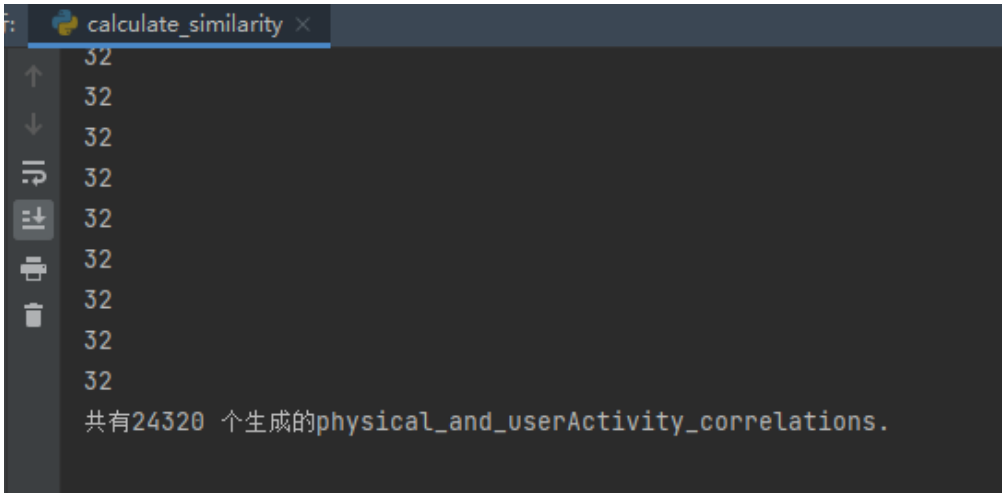
    一个带有温度的属性对可以生成256个相关性，根据



和8 Motion Sensors (M), 2 Door Sensor (D), 6 Temperature Sensors (T),4 lights(L),3 fans(Fan)

则预估生成24320个相关性（结果确实如此）





需要注意的是，Hypothetical Correlation Generation生成的相关性和语义分析部分生成的相关性一定会有重复的。

- **并且我觉得，考虑到CASAS的数据集较为简单，能够造成物理设备之间因共同事件相互关联的情况比较少，所以physical_and_userActivity_correlations的作用微乎其微，后面可以用结果验证下**

- Hypothesis Testing
  - 原文的时间间隔d=60s，也就是说前事件发生六十秒后，后事件发生就认为正常。但WSU CASAS数据集中的后事件与前事件都在一秒内发生，也就是前事件发生后，后事件马上发生
  - 由于上面的原因，一个设备的开event或者关event只可能导致一个后事件，不会导致两个或多个后事件
  - > 值得强调的是，假设的相关性不一定是正确的。这就是为什么我们需要假设检验，即使用事件日志验证假设相关性的过程。给定一个假设的相关性，我们遍历事件日志以找到所有与它的前面匹配的事件，并将它们中的每一个作为测试用例。然后，我们检查假设相关的后验事件或状态是否与事件日志中记录的物理基

> 础事实一致。例如，EMotion active的事件实例构成了假设关联↓EMotion active→switch(Light) on⊮的测试用例。如果在情绪激活后的短时间内出现开关(灯)打开，则视为成功。在我们的执行中，d = 60s，这足以等待反馈事件的到来，但又不足以接受与情感活动无关的事件。注意SmartThings的调度粒度是每分钟级别[1]。
>
> 检查这些测试用例可以看作是**一系列独立的伯努利试验**。我们使用**单尾检验[42]**来评估每个假设相关性的正确性。**对于给定的相关性，我们将备选假设Hα设置为"相关性成功且概率高于P0"。对应的零假设H0为"相关成功的概率不高于P0"。**
>
> **我们按照惯例选择95%的基准概率[27]，这意味着只有当原假设的p值小于5%时，相关性才能被接受。**

- 检查这些测试用例可以看作是**一系列独立的伯努利试验**，原因：
  1. **离散事件**：伯努利试验是最简单的随机试验，它涉及只有两种可能结果的单次试验，例如抛硬币试验中的"正面"或"反面"。在智能家居系统中，每个测试案例可以看作是一个类似的二元事件，即设备行为要么符合假设相关性（成功），要么不符合（失败）。
  2. **独立性**：伯努利试验的一个关键特性是每次试验的结果是独立的，即前一次试验的结果不会影响后一次试验。在智能家居系统中，如果事件日志足够多，且每个事件的发生不受其他事件的影响，那么每个测试案例也可以被认为是独立的。
- 假设检验部分的代码主要使用了Python的`scipy.stats`模块中的`binom_test`函数来进行。这个过程基于统计学的原理，旨在判断我们观察到的事件之间的相关性是否具有统计学意义。在这个例子中，我们关注的是`EMotion_active`事件发生后，在一定时间内（60秒），是否会导致`Eswitch(Light)_on`事件的发生。下面我将详细解释代码和背后的原理。

假设检验的代码解释：

```
from scipy.stats import binom_test

# 假设的成功概率阈值
P0 = 0.5  # 根据你的假设具体情况调整

# 执行单尾二项式检验
p_value = binom_test(success, total, P0, alternative='greater')
```

- `binom_test`函数的参数包括：
  - `success`：在我们的检验案例中，指的是在给定的时间窗口内（例如60秒），成功观察到`Eswitch(Light)_on`事件发生的次数。
  - `total`：这是观察到的`EMotion_active`事件的总次数，即总的测试案例数。
  - `P0`：这是我们的零假设（null hypothesis）H0下，事件相关成功的概率。在这里，它是基于领域知识或先前观察设定的一个阈值。
  - `alternative='greater'`：这表示我们的备择假设（alternative hypothesis）Hα是事件相关成功的概率大于P0的情况。

原理解释：

假设检验是用于决定样本数据是否支持某个假设的统计方法。在这个场景中，我们使用了二项检验（binomial test），它适用于评估观察到的成功次数在统计上是否显著地偏离了假设的成功概率（P0）。

- **零假设（H0）**：认为`EMotion_active`事件与`Eswitch(Light)_on`事件的发生之间没有统计学上的相关性，或者说，后者发生的概率不超过P0。
- **备择假设（Hα）**：认为`EMotion_active`事件发生后，`Eswitch(Light)_on`事件发生的概率高于P0。

在进行了`binom_test`之后，我们会得到一个`p_value`（p值），它表示在零假设为真的情况下，观察到的结果（或更极端的结果）出现的概率。如果这个概率很低（通常小于5%），我们就有足够的证据拒绝零假设，接受备择假设，即认为两个事件之间存在统计学上的相关性。

结果判断：

```
print(f"Successes: {success}, Total: {total}, P-value: {p_value}")

# 检查假设是否可以被接受
if p_value < 0.05:
    print("The hypothetical correlation is statistically
significant.")
else:
    print("The hypothetical correlation is not statistically
significant.")
```

- 如果`p_value < 0.05`，则表示我们有足够的证据拒绝零假设，认为`EMotion_active`和`Eswitch(Light)_on`之间存在统计学意义上的相关性。
- 如果`p_value >= 0.05`，则表示我们没有足够的证据拒绝零假设，也就是说，两个事件之间的相关性可能仅仅是偶然发生的，没有统计学上的意义。

  - Anomaly Detection

    同Hypothesis Testing一样，时间间隔d得改，改成一秒内没变化就有问题

- 需要保存每个设备及其对应的属性及其对应的可能的值

3.进度记录

4.3

编写known_rules.py:创建所有已知规则

4.4

编写device_information.py:

构建23个设备的对应类型、属性、值范围、当前值

编写MAIN.py：主函数，串联整个工作流程

```
following_state=State(subject='T103', attribute='temperature', constraint='>=25'), 'e2s' correlation:
pre_event=Event(subject='Fan2', attribute='fanMode', constraint='OFF',extraConstraint='None')
condition=None
following_state=State(subject='T103', attribute='temperature', constraint='<24'), 'e2s' correlation:
pre_event=Event(subject='Fan3', attribute='fanMode', constraint='ON',extraConstraint='None')
condition=None
following_state=State(subject='T104', attribute='temperature', constraint='>=25'), 'e2s' correlation:
pre_event=Event(subject='Fan3', attribute='fanMode', constraint='OFF',extraConstraint='None')
condition=None
following_state=State(subject='T104', attribute='temperature', constraint='<24')]
32 smartapp correlations in total.
```

编写添加calculate_similarity.py:

- ## 编写generate_correlations()函数:

  读取excel，生成相关性（工作量较大）

```
motion 和 door 之间相关，相关属性: power
motion 和 temperature 之间相关，相关属性: illuminance, sound, temperature, humidity, vibration
motion 和 lightingMode 之间相关，相关属性: illuminance, temperature, humidity, power
motion 和 fanMode 之间相关，相关属性: sound, vibration
door 和 lightingMode 之间相关，相关属性: power
temperature 和 lightingMode 之间相关，相关属性: illuminance, temperature, humidity
temperature 和 fanMode 之间相关，相关属性: sound, vibration
24320 physical_and_userActivity_correlations in total.
```

4.5，4.6

编写eventLogs.py:

eventLogs.py负责

- 读取日志
- 完成Hypothesis Testing部分

  先上结果，后面细说。

```
following_state=State(subject='M002', attribute='motion', constraint='OFF'), 'e2s' correlation:
pre_event=Event(subject='L002', attribute='lightingMode', constraint='ON',extraConstraint='None')
condition=None
following_state=State(subject='M004', attribute='motion', constraint='ON'), 'e2s' correlation:
pre_event=Event(subject='L002', attribute='lightingMode', constraint='OFF',extraConstraint='None')
condition=None
following_state=State(subject='M004', attribute='motion', constraint='OFF')]
22 smartapp correlations in total.

motion 和 door 之间相关，相关属性：power
motion 和 lightingMode 之间相关，相关属性：illuminance, temperature, humidity, power
motion 和 fanMode 之间相关，相关属性：sound, vibration
door 和 lightingMode 之间相关，相关属性：power
temperature 和 fanMode 之间相关，相关属性：sound, vibration
5888 physical_and_userActivity_correlations in total.

The function took 130.4954183101654 seconds to run.
350 physical_and_userActivity_correlations after hypothesis testing.
```

这部分要考虑的东西很多，首先要学会处理特殊情况：

- 有condition的
- 有温度的，比如单个日志中温度传感器的值'23'应该满足correlation中类似'<=24'的constraint

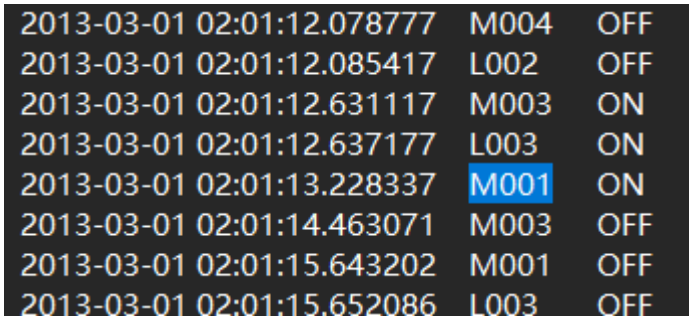
目前实验的是smartapp correlations

存在两个问题：

1.or的问题：有or的规则数据结构处理不当，导致逻辑不对

```
'e2e' correlation:
pre_event=Event(subject='M001', attribute='motion', constraint='ON',extraConstraint='None')
condition=None
following_event=Event(subject='L003', attribute='lightingMode', constraint='ON',extraConstraint='None')
success_count=1579,total_count=2783
success_rate=0.5673733381243262

'e2e' correlation:
pre_event=Event(subject='M003', attribute='motion', constraint='ON',extraConstraint='None')
condition=None
following_event=Event(subject='L003', attribute='lightingMode', constraint='ON',extraConstraint='None')
success_count=3369,total_count=4558
success_rate=0.7391399736726635
```



修改方法：

```
#5.1
trigger= {"subject": "M001", "attribute": "motion", "constraint": "ON",
"extraConstraint": None}
condition=None
action = {"subject": "L003", "attribute": "lightingMode", "constraint": "ON",
"extraConstraint": None}
rule = semantic_analysis.Rule(trigger, condition, action)
knownRules.append(rule)
```

改为

```
#5.1
trigger= {"subject": "M001", "attribute": "motion", "constraint": "ON",
"extraConstraint": None}
condition={"subject": "L003", "attribute": "lightingMode", "constraint": "OFF"}
action = {"subject": "L003", "attribute": "lightingMode", "constraint": "ON",
"extraConstraint": None}
rule = semantic_analysis.Rule(trigger, condition, action)
knownRules.append(rule)
```

2.温度的问题：有温度的correlation普遍成功率低，因为温度计时器会不定时报告读数，此时若风扇已经打开则不会重新打开

解决方法：添加条件，开风扇的条件是风扇目前关着，关风扇的条件是风扇目前开着

修改方法：

```
#9
trigger= {"subject": "T101", "attribute": "temperature", "constraint": ">=32",
"extraConstraint": None}
condition=None
action = {"subject": "Fan1", "attribute": "fanMode", "constraint": "ON",
"extraConstraint": None}
rule = semantic_analysis.Rule(trigger, condition, action)
knownRules.append(rule)
```

改为

```
#9
trigger= {"subject": "T101", "attribute": "temperature", "constraint": ">=32",
"extraConstraint": None}
condition={"subject": "Fan1", "attribute": "fanMode", "constraint": "OFF"}
action = {"subject": "Fan1", "attribute": "fanMode", "constraint": "ON",
"extraConstraint": None}
rule = semantic_analysis.Rule(trigger, condition, action)
knownRules.append(rule)
```

> （上面两个问题的解决办法还是我早上在床上想到的…）

- 下面是physical_and_userActivity_correlations假设检验计算成功率后的一些结果，有些是不合理的：

```
The function took 762.005373954773 seconds to run.
numbers that success_rate>0.9:4351
```

```
'e2s' correlation:
pre_event=Event(subject='T102', attribute='temperature', constraint='>=25',extraConstraint='None')
condition=None
following_state=State(subject='L001', attribute='lightingMode', constraint='OFF')
success_count=147,total_count=147
success_rate=1.0
```

上面这个情况代表着温度高不高和灯的亮不亮有关，显然这是完全不太符合常理的。

```
'e2s' correlation:
pre_event=Event(subject='Fan3', attribute='fanMode', constraint='ON',extraConstraint='None')
condition=None
following_state=State(subject='T102', attribute='temperature', constraint='<26')
success_count=73,total_count=73
success_rate=1.0

'e2s' correlation:
pre_event=Event(subject='Fan3', attribute='fanMode', constraint='OFF',extraConstraint='None')
condition=None
following_state=State(subject='T102', attribute='temperature', constraint='<26')
success_count=72,total_count=72
success_rate=1.0
```

```
'e2s' correlation:
pre_event=Event(subject='Fan3', attribute='fanMode', constraint='ON',extraConstraint='None')
condition=None
following_state=State(subject='T102', attribute='temperature', constraint='<31')
success_count=73,total_count=73
success_rate=1.0
```

事实上，从原数据中能看出T102其实一直都小于26度，这就导致这种情况的发生

- 下面的情况也值得思考：

```
'e2s' correlation:
pre_event=Event(subject='T101', attribute='temperature', constraint='>=32',extraConstraint='None')
condition=None
following_state=State(subject='Fan1', attribute='fanMode', constraint='ON')
success_count=425,total_count=432
success_rate=0.9837962962962963
p_value=0.00020029887379276884

'e2s' correlation:
pre_event=Event(subject='T101', attribute='temperature', constraint='>=33',extraConstraint='None')
condition=None
following_state=State(subject='Fan1', attribute='fanMode', constraint='ON')
success_count=102,total_count=102
success_rate=1.0
p_value=0.005343277621351436
```

直观感受，在假设检验后correlation Mining是必要的（这块错怪作者了，原来觉得correlation Mining这块非常鸡肋，起不到应该有的效果，比如自动化规则改变时可以更新相关性），考虑这样一种情况：以前温度传感器大于23度开风扇，后来用户修改自动化规则，改为温度传感器大于20度开风扇，那么就应该将后面的相关性覆盖前面的（如果不修改，21度的时候风扇会不开）。

an e2e correlation $\mathbb{C}_s = \langle \mathcal{E}_a^{\alpha(A)} \rightarrow \mathcal{E}_b^{\beta(B)} \rangle$ extracted from a smart app *covers* a correlation $\mathbb{C}_h = \langle \mathcal{E}_c^{\gamma(C)} \rightarrow \mathcal{E}_d^{\delta(D)} \rangle$ that passes hypothesis testing if they meet two conditions: 1) they have the same posterior event (i.e., $\mathcal{E}_b^{\beta(B)} = \mathcal{E}_d^{\delta(D)}$); and 2) $\mathcal{E}_a^{\alpha(A)}$ (logically) implies $\mathcal{E}_c^{\gamma(C)}$ (i.e., $\mathcal{E}_a^{\alpha(A)} \Rightarrow \mathcal{E}_c^{\gamma(C)}$). If $\mathbb{C}_s$ covers $\mathbb{C}_h$, the latter is removed. In the example mentioned above, a smart app derived e2e correlation $\langle \mathcal{E}_{open}^{contact} \wedge \mathcal{S}_{sunset}^{location} \rightarrow \mathcal{E}_{on}^{switch(PorchLight)} \rangle$ covers the mined correlation $\langle \mathcal{E}_{open}^{contact} \rightarrow \mathcal{E}_{on}^{switch(PorchLight)} \rangle$ because they have the same posterior event and $(\mathcal{E}_{open}^{contact} \wedge \mathcal{S}_{sunset}^{location}) \Rightarrow \mathcal{E}_{open}^{contact}$; thus, the latter correlation is removed.

- 因为属性太少、描述不准确等问题。。有的结果其实比较不太合常理，比如temperature

```
motion 和 door 之间相关，相关属性：power
motion 和 temperature 之间相关，相关属性：illuminance, sound, temperature, humidity, vibration
motion 和 lightingMode 之间相关，相关属性：illuminance, temperature, humidity, power
motion 和 fanMode 之间相关，相关属性：sound, vibration
door 和 lightingMode 之间相关，相关属性：power
temperature 和 lightingMode 之间相关，相关属性：illuminance, temperature, humidity
temperature 和 fanMode 之间相关，相关属性：sound, vibration
24320 physical_and_userActivity_correlations in total.
```

上面用word2vec计算得到的温度和灯开不开有关，放到实际情况中就很不符合常理，所以我手动修改了邻接表，删掉了一些不太正确的。

手动修改后的相关性

	motion	door	temperature	lightingMode	fanMode
motion		power		illuminance, temperature, humidity, power	sound, vibration
door	power			power	
perature					sound, vibration
ghtingMo	illuminance, temperature, hum	power			
fanMode	sound, vibration		sound, vibration		

```
motion 和 door 之间相关，相关属性：power
motion 和 lightingMode 之间相关，相关属性：illuminance, temperature, humidity, power
motion 和 fanMode 之间相关，相关属性：sound, vibration
door 和 lightingMode 之间相关，相关属性：power
temperature 和 fanMode 之间相关，相关属性：sound, vibration
5888 physical_and_userActivity_correlations in total.
```

修改之后的相关性更科学。

- 之前初步的想法是通过假设检验保留 成功率大概在0.9以上的。

但会出现下面这种情况

无论M002是ON/OFF，D002都是关着的状态，这种相关性被认为是正确的，况且除去success_count还有一百多个D002是开着的状态。

```
'e2s' correlation:
pre_event=Event(subject='M002', attribute='motion', constraint='ON',extraConstraint='None')
condition=None
following_state=State(subject='D002', attribute='door', constraint='CLOSE')
success_count=3238,total_count=3414
success_rate=0.9484475688342121
p_value=4.908741028554857e-25

'e2s' correlation:
pre_event=Event(subject='M002', attribute='motion', constraint='OFF',extraConstraint='None')
condition=None
following_state=State(subject='D002', attribute='door', constraint='CLOSE')
success_count=3240,total_count=3415
success_rate=0.9487554904831625
p_value=2.2637958001868405e-25
```

所以假设检验保留 成功率修改为在0.95以上

- 还有这种情况：

  ```
  'e2s' correlation:
  pre_event=Event(subject='Fan3', attribute='fanMode', constraint='ON',extraConstraint='None')
  condition=None
  following_state=State(subject='T101', attribute='temperature', constraint='>=20')
  success_count=73,total_count=73
  success_rate=1.0
  p_value=0.02364956658822994

  'e2s' correlation:
  pre_event=Event(subject='Fan3', attribute='fanMode', constraint='OFF',extraConstraint='None')
  condition=None
  following_state=State(subject='T101', attribute='temperature', constraint='>=20')
  success_count=72,total_count=72
  success_rate=1.0
  p_value=0.024894280619189416
  ```

  不管Fan3开还是不开，T101温度始终大于20度。

  这种就很鸡肋，你不能说他错，但是没什么用

后面还有换成22度23度（一直到30度）的，也不错，但没什么用，后续可以考虑把他们筛掉，但是无伤大雅。

```
'e2s' correlation:
pre_event=Event(subject='Fan3', attribute='fanMode', constraint='ON',extraConstraint='None')
condition=None
following_state=State(subject='T101', attribute='temperature', constraint='>=22')
success_count=73,total_count=73
success_rate=1.0
p_value=0.02364956658822994

'e2s' correlation:
pre_event=Event(subject='Fan3', attribute='fanMode', constraint='OFF',extraConstraint='None')
condition=None
following_state=State(subject='T101', attribute='temperature', constraint='>=22')
success_count=72,total_count=72
success_rate=1.0
p_value=0.024894280619189416
```

> Hypothesis Testing部分比较耗时，运行时间大概两分钟左右

4.7

1.编写correlation refining部分

考虑上面提到的情况：

```
'e2s' correlation:
pre_event=Event(subject='T101', attribute='temperature', constraint='>=32',extraConstraint='None')
condition=None
following_state=State(subject='Fan1', attribute='fanMode', constraint='ON')
success_count=425,total_count=432
success_rate=0.9837962962962963
p_value=0.00020029887379276884

'e2s' correlation:
pre_event=Event(subject='T101', attribute='temperature', constraint='>=33',extraConstraint='None')
condition=None
following_state=State(subject='Fan1', attribute='fanMode', constraint='ON')
success_count=102,total_count=102
success_rate=1.0
p_value=0.005343277621351436
```

```
'e2s' correlation:
pre_event=Event(subject='Fan3', attribute='fanMode', constraint='ON',extraConstraint='None')
condition=None
following_state=State(subject='T101', attribute='temperature', constraint='>=22')
success_count=73,total_count=73
success_rate=1.0
p_value=0.02364956658822994

'e2s' correlation:
pre_event=Event(subject='Fan3', attribute='fanMode', constraint='OFF',extraConstraint='None')
condition=None
following_state=State(subject='T101', attribute='temperature', constraint='>=22')
success_count=72,total_count=72
success_rate=1.0
p_value=0.024894280619189416
```

- 什么情况下删除相关性a和b？：

1.a与b的following_state一模一样，pre_event设备属性一模一样，但constriant相反

第一轮筛完之后

2.a与b的following_state一模一样，pre_event设备属性一模一样（特指温度），但constriant存在同类且也有的更严苛，则删掉不严苛的。（也就是原文的逻辑=>）

单独对physical_and_userActivity_correlations进行refine，350个correlations最后只保留了42个，其中还有一部分是跟smartapp correlations重复的

```
The function took 138.9063262939453 seconds to run.
350 physical_and_userActivity_correlations after hypothesis testing.

44
42
42 physical_and_userActivity_correlations after correlation refining.
```

42个里主要涉及有：

1.与门有关的。这个自动化规则里没有

eg:

```
'e2s' correlation:
pre_event=Event(subject='D002', attribute='door', constraint='CLOSE',extraConstraint='None')
condition=None
following_state=State(subject='M004', attribute='motion', constraint='OFF')
success_count=728,total_count=750
success_rate=0.9706666666666667
p_value=0.0036903073180318035
```

2.

单独对smartapp correlation进行refine，没有一个被筛掉。

- 下一步，合并smartapp correlation和physical_and_userActivity_correlations，统一进行refine

  因为两个相关性中存在一些一模一样的相关性，所以相同的只保留一份。

> ### 使用列表推导式保持顺序
>
> 如果你想保持原始列表中元素的顺序，可以使用列表推导式加上一个条件判断来实现。这种方法首先遍历第一个列表的所有元素，然后遍历第二个列表的元素，仅当这个元素不在第一个列表中时，才将它添加到结果列表中。
>
> ```
> pythonCopy codelist1 = [1, 2, 3, 4]
> list2 = [3, 4, 5, 6]
>
> # 保持顺序的同时去除重复元素
> merged_list = list1 + [item for item in list2 if item not in list1]
>
> print(merged_list)
> ```
>
> 这种方法保持了第一个列表的完整性和顺序，并将第二个列表中的非重复元素按其在原列表中的顺序追加到结果列表的末尾。

可以使用上面这种方法去重，也可以使用集合的方式，但集合不能保证有序。

结果

```
condition=None
following_state=State(subject='M004', attribute='motion', constraint='OFF')]
22 smartapp correlations in total.

motion 和 door 之间相关，相关属性：power
motion 和 lightingMode 之间相关，相关属性：illuminance, temperature, humidity, power
motion 和 fanMode 之间相关，相关属性：sound, vibration
door 和 lightingMode 之间相关，相关属性：power
temperature 和 fanMode 之间相关，相关属性：sound, vibration
5888 physical_and_userActivity_correlations in total.

The function took 158.01357889175415 seconds to run.
350 physical_and_userActivity_correlations after hypothesis testing.

64 correlations after correlation refining.
```

总结

总体来讲，这一周给我最大的感受就是很多细节上的问题只有当你真正开始做才会遇到，之前觉得自己完全理解论文了，但在实现时才发现没有想象的那么简单，。比如，很多时候我需要手动检查相关性，才能发现一些问题并没有处理好，这时候往往又需要回头修改代码等等，检查相关性这块特别耗精力。

最后，复现论文有种重走作者长征路的感觉。

下一步工作

1.实现最后的模块----异常检测。到时候估计又会发现这周的一些小问题哈哈。

2.在假设不知道任何自动化规则的前提下把整个流程重新走一遍，与已知规则的情况做一个对比。

预计下周将完成所有工作。