复现第一周进度(3.25-3.31)

第一周总体来讲比较满意，课余时间利用的比较充分，热情也比较足，基本完成了原定目标，并且超额完成了后面的工作（基本完成了所有不需要日志就能开展的复现工作）。

1.原定目标

目标	完成情况
1.参考作者前文，了解语义提取流程	√
2.回顾python相关知识	√
3.学会git	×（还没用到）

1.参考作者前文，了解语义提取流程---用时：一下午

花了大概一下午的时间阅读了`DSN20-Cross App Interference Threats in Smart Homes Categorization, Detection and Handling`中语义提取的工作，得出的结论是：**这块先不做了。**因为流程繁琐复杂，工作量很大，料想到一开始入手这块会造成很大的心理压力，导致后面直接做不下去。

难点有：

- 流程复杂
- 工作量大
    1. **静态分析**：
    2. **路径搜索策略**：
    3. **符号输入**：
    4. 分析入口点和汇点：
    5. 生成控制流图：
    6. **约束生成**：
    7. **API**建模：
    8. **编译器定制**：
- 论文在讲这块时很多步骤都是概括性的，不明确的（有种作者也不想让你完全清楚每一个细节的感觉），换句话说，它的每一句话背后的工作量都很大。

  eg:

  "分析入口点和汇合点。在我们的实现中，分析入口点包括生命周期方法、安装、更新和卸载。分析接收器包括功能保护的设备命令和安全敏感的SmartThings api(如setLocationMode())。**我们考虑由104个功能[8]和21个SmartApp api(见附录A)保护的126个设备控制命令。**"

  "我们从每个赋值语句中建立数据约束。具体来说，**我们在编译器中编写回调方法来处理Groovy文档中定义的38种表达式类型[24]。**另一方面，我们还从条件语句构建谓词约束，即，if语句中的每个布尔表达式或switch语句中的每个case表达式都被转换为约束。我们还通过将每个三元表达式分解为两个分支来处理它们。"

"我们首先对**10个SmartApp APIs2建模**，这些APIs2根据它们的参数和功能来调度指定方法的执行。"

"为了对可能涉及约束构造的api进行建模，我们通过手动查看SmartThings开发人员文档[10]来对对象、方法和对象属性访问进行建模。API方法的返回值和不依赖于其他数据的对象属性也被标记为符号输入。**我们总共建模了173个API方法和94个对象属性访问，并根据每个方法或属性访问的参数和返回值重写了一个静态建模函数。**"

- 讲实话当初看到这块的时候心态就有点小崩，能够看出作者对smartthings整个流程框架、groovy语言编写的smartapp的整个执行过程、先前的的语义提取工作的大概方法都十分清楚了解，这些都不是目前的我所具备的。其次，语义提取这部分的工作量可以说一点都不比HAWatcher小。

综上所述，在看完这篇文章的那个晚上，我就立刻决定将这部分的工作放到可能的未来。

2.回顾python相关知识---用时：2天

学习方式:阅读csdn总结+python123

Python【面向对象】保姆式教学，零基础速成精英之路_phython面向对象速成-CSDN博客Python入门基础知识总结-CSDN博客

Python入门基础知识总结-CSDN博客

万字【Python基础】保姆式教学，零基础快速入门Python_西安科技大学研究生院-CSDN博客

Python123

这部分很快，主打一个回顾。

3.学会git

后面用到再说。

2.新完成的工作---用时：3天

目标	完成度
Semantic Analysis	√
Correlation Mining中的Hypothetical Correlation Generation部分（占大头）	√

- 因为我放弃先进行语义提取工作，所以在学习python后便马不停蹄直接开始HAWatcher的复现工作。
- 进度还是比较满意，可以说，基本完成了所有不需要日志就能开展的工作。
- 代码工作量大概两三百行代码，看着不多，实际上确实不多。。。。但其实还是挺耗心思的。具体工作是红框内的所有部分：
-

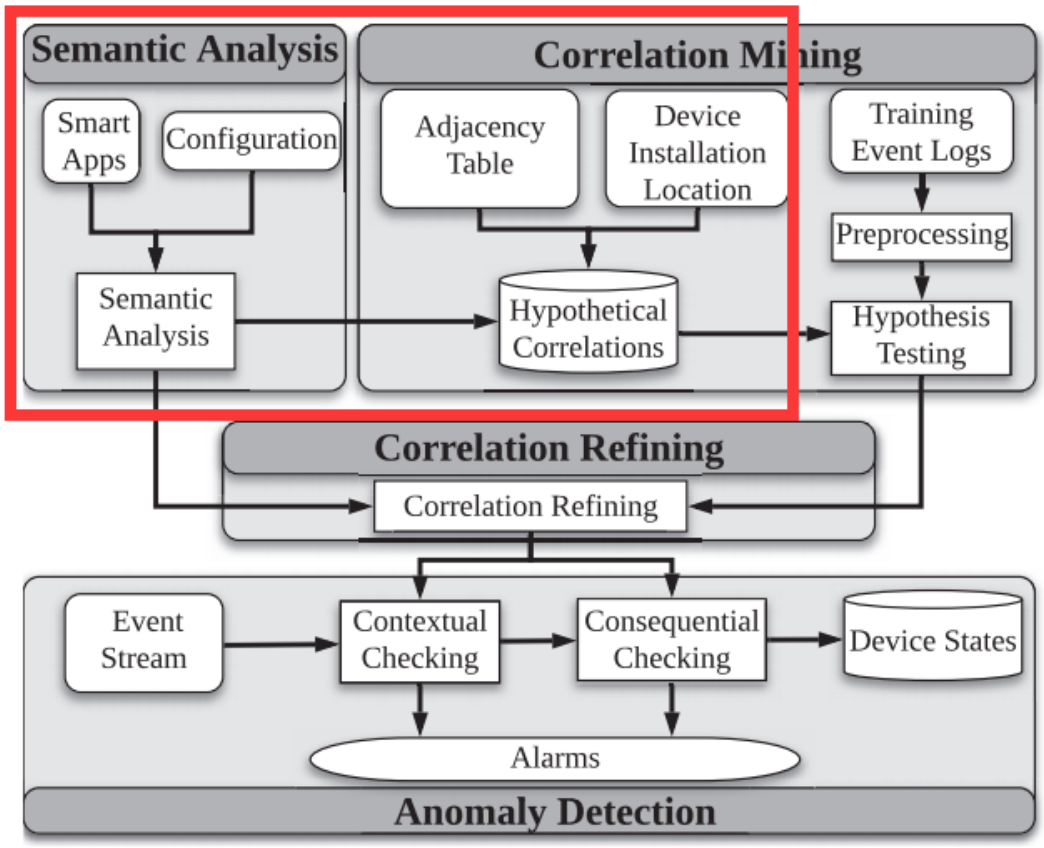


Figure 4: Architecture of HAWatcher.

1.Semantic Analysis（smartapp channel)

这部分是第一部分，因为涉及到很多如何定义数据结构的问题，所以心思主要花在这里，数据结构一旦定义好了，后面处理他们之间逻辑就ok了

当然这部分的假设是所有smartapp语义已经提取成rule了。

具体工作：

- 定义数据结构

  也就是如何表示相关性（e2e/e2s correlation）、tca rule

  定义了四个class: Rule;Event;State;correlation、编写函数将rule转化成correlation

  这里定义的rule的格式和作者前文的有所不同，为了方便处理

- 编写函数简化带有条件的相关性
- 编写函数，根据一个相关性生成更多相关性
- 编写测试用例

2.Hypothetical Correlation Generation（ physical and user activity channels)

这部分属于第二部分`Correlation Mining`中的最主要工作部分（不需要日志），主要就是将设备属性，利用Word2vec模型生成属性对表。本质上的工作就是**计算词和词的相似度**

原文：

> 对于物理通道相关性，我们考虑了与许多智能家居物联网设备相关的七个物理特性:照度、声音、温度、湿度、振动、功率和空气质量。为了确定两个物联网设备属性是否可以通过物理属性相关联，我们开发了一种基于NLP(自然语言处理)的方法。具体来说，对于抽象物联网设备的每个属性，我们从SmartThings的开发者网站[19]获取其描述，并将其解析为一个单独的单词列表。为了客观地评价属性与物理属性之间的关联度，我们使用Google预训练的word2vec模型[59]计算列表中每个单词与物理属性之间的语义相似度得分，并将得分最高的作为物理属性与属性之间的关联度得分。对于每个物理属性，我们选择得分最高的前十个属性，这些属性通过该物理属性被认为是相互关联的。

具体工作：

- 查阅word2vec模型，以及如何实现

  一文读懂：词向量 Word2Vec - 知乎 (zhihu.com)

  自然语言处理=======python利用word2vec实现计算词语相似度【gensim实现】 word2vec python-CSDN博客

  在Python中导入GoogleNews-vectors-negat_googlenews-vectors-negative300.bin-CSDN博客

> Word2Vec的主要作用是生成词向量，而词向量与语言模型有着密切的关系。Word2Vec的特点是能够将单词转化为向量来表示，这样词与词之间就可以定量的去度量他们之间的关系，挖掘词之间的联系。Word2Vec模型在自然语言处理中有着广泛的应用，包括词语相似度计算、文本分类、词性标注、命名实体识别、机器翻译、文本生成等。其主要目的是将所有词语投影到K维的向量空间，每个词语都可以用一个K维向量表示。
>
> 词向量就是用来将语言中的词进行数学化的一种方式，顾名思义，词向量就是把一个词表示成一个向量。 我们都知道词在送到神经网络训练之前需要将其编码成数值变量，常见的编码方式有两种：One-Hot Representation 和 Distributed Representation。
>
> Word2Vec模型的核心思想是通过词语的上下文信息来学习词语的向量表示。具体来说，Word2Vec模型通过训练一个神经网络模型，使得给定一个词语的上下文时，能够预测该词语本身（CBOW模型），或者给定一个词语时，能够预测其上下文（Skip-gram模型）。Word2Vec的训练模型本质上是只具有一个隐含层的神经元网络。它的输入是采用One-Hot编码的词汇表向量，它的输出也是One-Hot编码的词汇表向量。使用所有的样本，训练这个神经元网络，等到收敛之后，从输入层到隐含层的那些权重，便是每一个词的采用Distributed Representation的词向量。

这个模型是2013年提出的，跟现在的类型nlp方法比已经算不那么新了，但还是很牛逼，当结果跑出来的时候觉得很神奇，很实用。

具体来讲，我在看这块的时候重点不是具体原理，更关注的是如何调用，实现，那么简单来讲，Word2vec需要一个很大很大的数据集（互联网上任何有逻辑的信息都可以作为数据集），这个数据集基本包括了所有的英文单词。现将所有单词转化成One-Hot编码（比较low）的词汇表向量，然后训练。训练后神经网络中的权重，就是对应单词的Distributed Representation（比较nice，因为很稀疏，信息量更大）的词向量，之后每个单词只要有了对应的词向量，用那些传统的计算向量相似度的方法计算就OK了（比如欧氏距离、余弦相似度）。

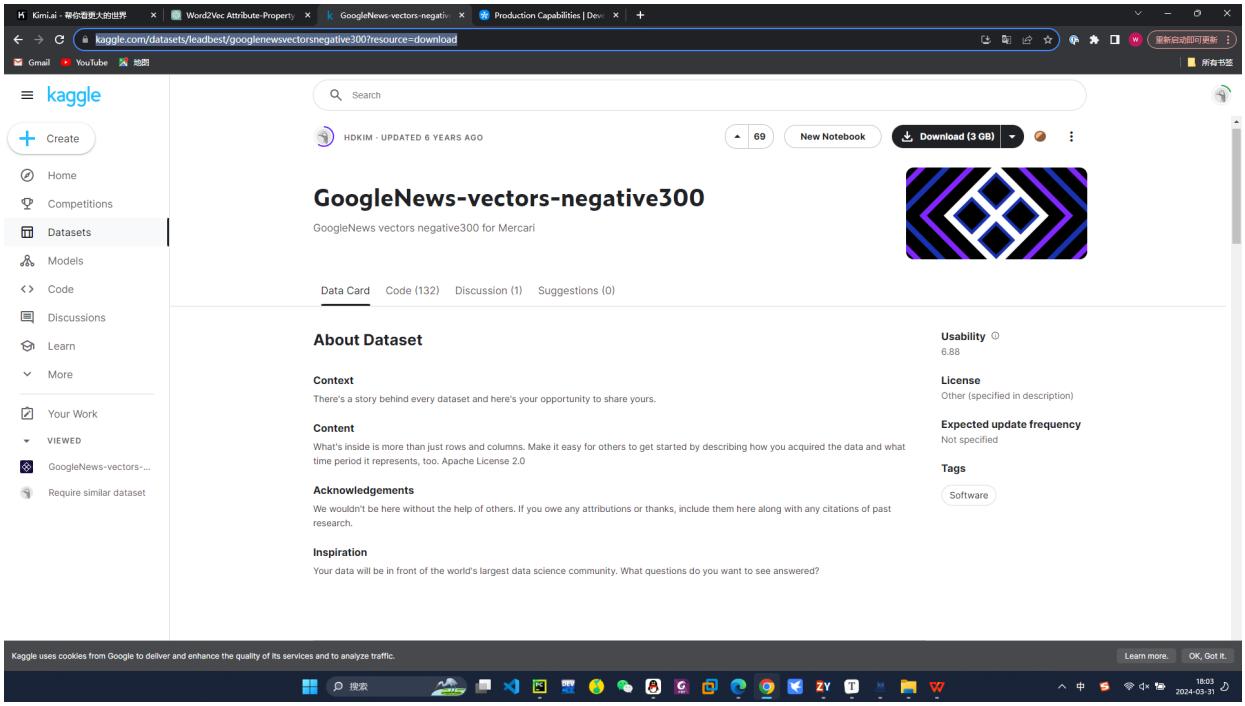
那么当然我是必不可能自己训练的（乐），重点是应用，所以和大部分人一样，只需要提前下载别人预训练好的模型，用就ok了，

那么互联网上最出名的已经训练好的数据集是`GoogleNews-vectors-negative300.bin.gz`

> `GoogleNews-vectors-negative300.bin.gz` 是一个包含预训练词向量的文件，这些词向量由Google的Word2Vec算法生成。这个具体的文件是基于Google新闻数据集训练的，包含大约300万个词汇或短语的300维向量表示。

下载链接：

https://www.kaggle.com/datasets/leadbest/googlenewsvectorsnegative300?resource=download



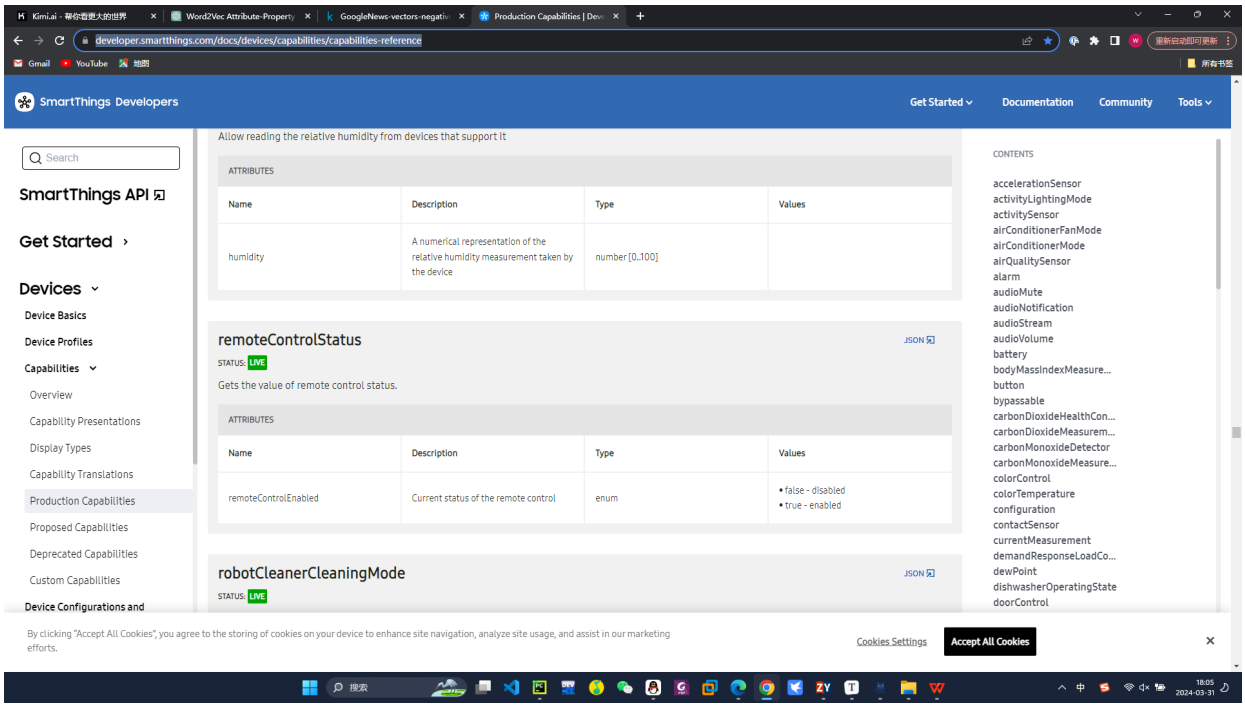
（原来大部分人用的下载链接已经下架了，这个链接还是我在一个博客的评论底下找到的。。。）

ok,那么有了数据集，下面只需要调用就ok了，这里需要调用一个gensim库，里面有已经编写好的利用预训练数据计算相似度的函数。

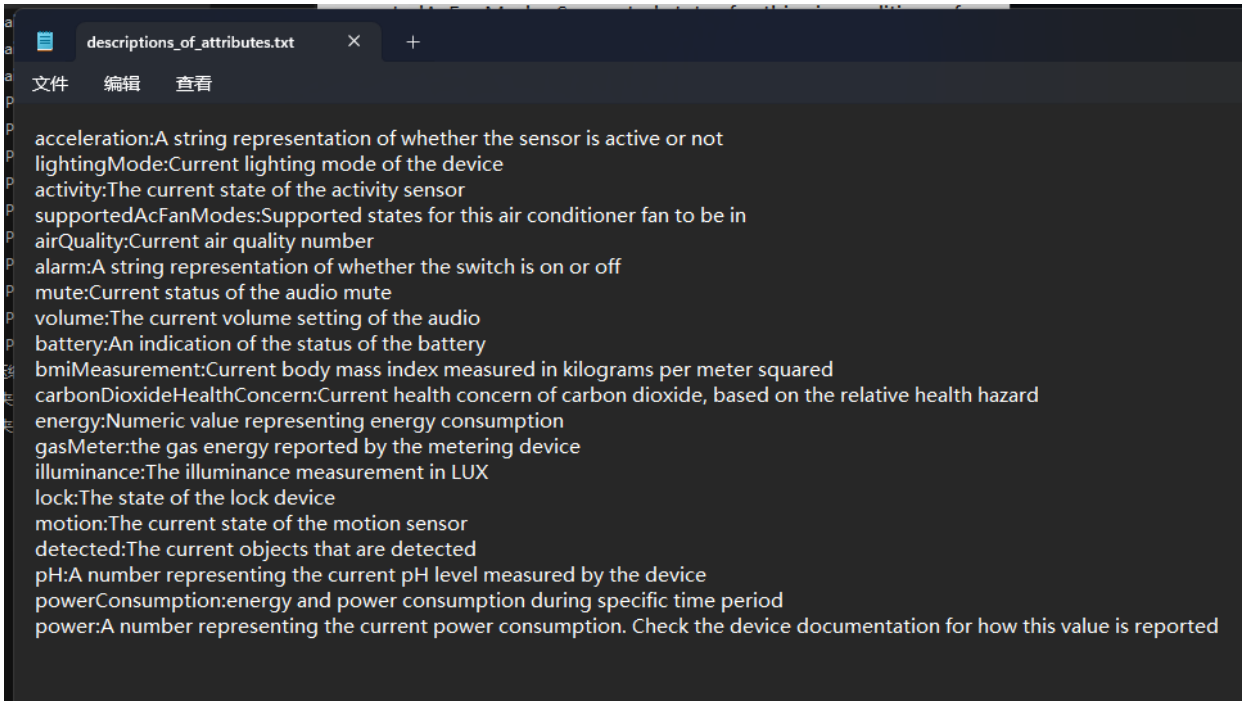
- 收集属性和对应描述

参考链接：https://developer.smartthings.com/docs/devices/capabilities/capabilities-reference

里面有所有属性和对应描述：



原文收集了73个，我为了测试只收集了20个，后面如果要用到再收集。

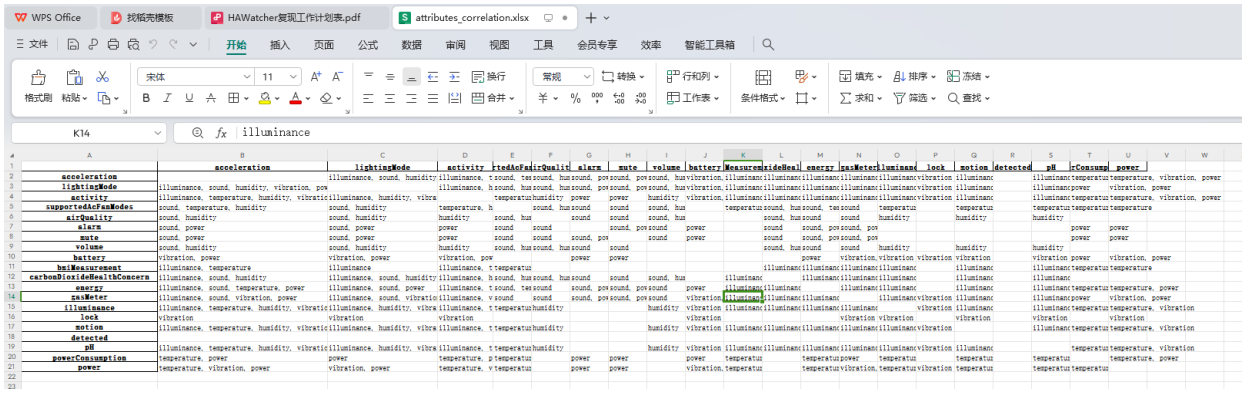


至此所有准备工作完成。

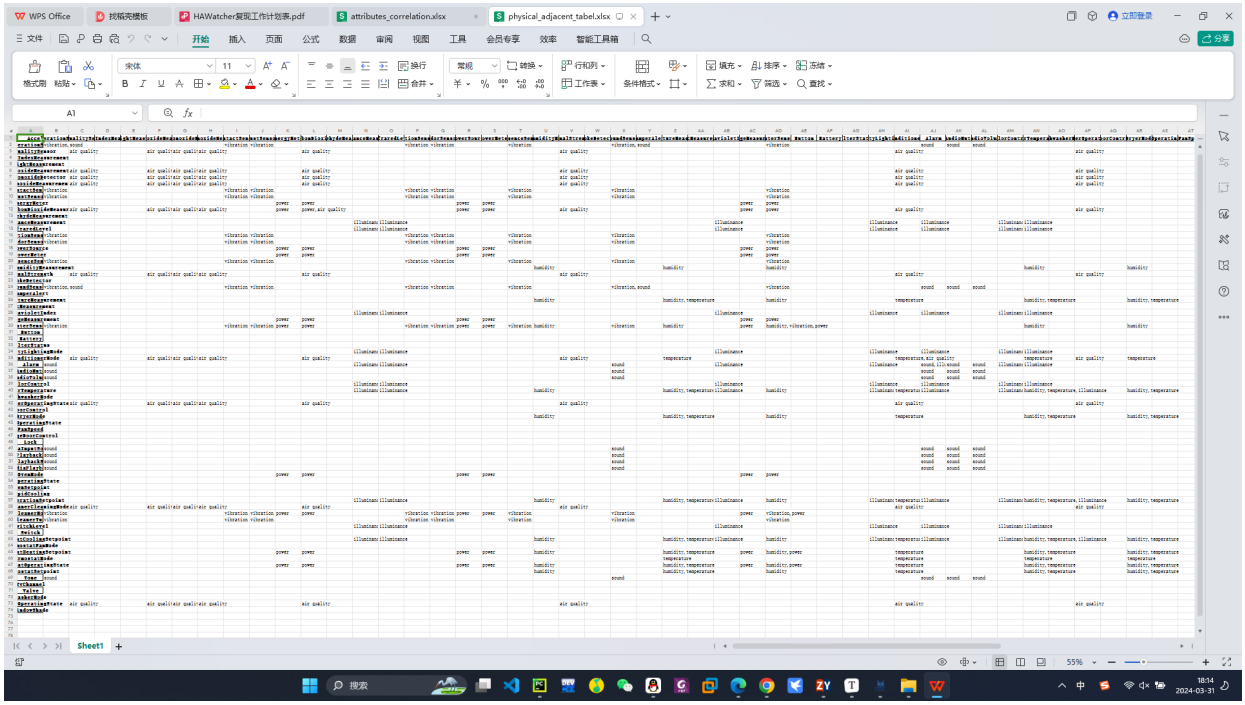
这部分的具体过程就不说了，下面展示一下结果：

```
排序后:
Top attributes for illuminance:
illuminance: 1.0 acceleration: 0.47674795985221863 activity: 0.47674795985221863 mo

Top attributes for sound:
mute: 0.4219237267971039 volume: 0.4219237267971039 lightingMode: 0.336089432239532

Top attributes for temperature:
illuminance: 0.40263831615448 acceleration: 0.35520923137664795 activity: 0.3552092

Top attributes for humidity:
lightingMode: 0.28572434186935425 illuminance: 0.278033047914505 supportedAcFanMode

Top attributes for vibration:
acceleration: 0.5514553785324097 activity: 0.5514553785324097 motion: 0.55145537853

Top attributes for power:
powerConsumption: 1.0 power: 1.0 energy: 0.48809513449668884 gasMeter: 0.4880951344
```

结果可以说是非常的nice，符合常理，比如`sound`和`mute`相似度最高,`vibration`和`accelaration`相似度最高等等，说明这个模型起到了他应该起的作用，下面是我收集的20个属性生成的表格：



下面这个是作者在github上传的表格



作者的比较稀疏，原因很简单：**我只收集了20个属性，作者是73个，在论文的流程中要选择一个property中得分最高的前十个属性作为相关属性，我的属性少，比例大，自然就稠密了**

3.总结

- 原来的每个目标的计划时间确实是定的长了，抛开前面的语义提取工作，HAWatcher整个的复现工作其实逻辑还是比较清晰的。
- **经过这一周，我预估，抛开前面的语义提取工作，整个的复现大概不到一个月就能完成。**算上这一周，再来三周完成剩下的需要日志的工作，应该是可以做到的，（在这里也给自己push一下）
- 接下来的工作都需要日志，具体工作:

  1.分析日志，根据日志的具体格式把原来的数据结构改改（这块肯定是要改的），并且根据实际数据收集对应的属性描述等等。

  2.在事件中插入异常事件（当然自己不会像作者一样那么全面，数据多，主要是要把最终结果跑通了）

  3.Prepossessing Event Logs、Hypothesis Testing、Correlation Refining、anomly detection这些流程的复现工作

  4.完成以上所有的工作，就可以进行测试了，测试完就齐活了

End.