1. **自己提出的问题的理解：**

问题1：P16说不需要将整个数据集加载到内存，这是Apriori的特点，但是我觉得既然要统计一个事务的置信度，应该没有必要必须把全部的数据集加载进去，可以一条一条读，那么有什么算法是必须要把整个数据集都放进去的嘛，为什么要这么做？

讨论后的理解：有的，Fp-growth算法，Apriori算法是基于宽度优先搜索的，它对每个层次的候选项集都需要扫描一遍整个数据，所以可以每次都只需要读入一条数据，节约了存储空间，但是因为每次读入都需要花费一定的IO时间，所以它的时间复杂度非常高。而频繁模式树增长算法（Frequent Pattern Tree Growth）即Fp-Growth算法采用分治的思想，将数据库中的频繁项集压缩到一棵频繁模式树中，同时保持项集之间的关联关系。然后将这棵压缩后的频繁模式树分成一些条件子树，每个条件子树对应一个频繁项，从而获得频繁项集，最后进行关联规则挖掘。因为需要将整个数据集存储到内存中，所以需要空间复杂度非常高，但是时间复杂度却比Apriori算法小很多。

参考资料：<https://www.cnblogs.com/stormtides/p/11890440.html>

问题2：我感觉Apriori太暴力了，有没有一些其他的优于Apriori的算法？

讨论后的理解：同第一个问题，Fp-Growth算法即为Apriori算法在时间复杂度上的优化算法，它使用分治的思想牺牲了空间复杂度来换取较低的时间复杂度，各有优缺点，所以在使用的时候可以根据实际情况进行选择。

1. **别人提出的问题的理解：**

问题1：P18 怎么得到Apriori算法是指数级的？

自己的理解：Apriori算法的目的即为求Fm，从F1开始，求F1的方法是从n个项目中选择符合最小支持度的项目，时间复杂度为O(n)，即O(),求F2的时候首先需要从F1中选择两个项目的组合，时间复杂度为O()，依次类推，将每个层次的时间复杂度加起来即为最终的总的时间复杂度：O（2n）

2、问题2：P22第二段，提到为了解决稀有项目问题，一个常见的解决办法是将数据分为几个小块，每个子集只包含那些出现频率比较相近的项目。关于这个分块依据（方法）不是很理解

自己的理解：因为在规则挖掘中，如果两个项目的出现频率差别非常大，比如电脑和牛奶，牛奶出现的频率非常高但是电脑出现的频率非常低，如果我们想用这样的数据集去挖掘关于电脑的规则，那么就必须把最小支持度设置的非常小才能让频繁项目集中出现电脑，这样的话如果奶酪和牛奶在数据集中并不频繁但是相比于电脑来说却非常的频繁，此时奶酪就会被认定为频繁项目，这就是稀有项目问题，问题的本质是项目之间的性质差别。此时如果使用将出现频率相似的数据分成一块，对每一块单独设置最小支持度，那么对于每一块来说，就不会出现因为项目间频率差别非常大而出现的稀有项目问题。

3、问题3：假定各个项目的MIS都由用户指定，那么参数φ应该怎么考虑呢？为什么要设置该参数呢（即，在满足各自MIS的情况下，支持度相差很大的事物为什么要过滤掉）？P25

自己的理解：这个问题和第二个问题本质上是一样的，因为需要保证每一块内的项目的出现频率相似，所以需要设定一个值，让每一块的出现频率最大的项目和最小的项目的最小支持度不会相差太大，避免了因为最小支持度相差太大而导致的将稀有项目和频繁项目同时作为频繁项目生成。这个参数的值一般人为指定，只要满足能够让性质相差不是很大的项目在个块中就好了。

1. 读书计划

1、本周完成的内容章节：2.1-2.4

2、下周计划：2.5-2.9

1. 读书摘要及理解或伪代码的具体实现

**(1)读书摘要：**

## 2.1 关联规则的基本概念

（1）支持度：

其中X和Y分别表示两个项集，n表示所有的T中事物的数目。

顾名思义，支持度也是就说这个事物集能够支持我们的预测的结果的程度，X和Y的交集就是我们研究的X和Y有多少的关联度，与n的比值就表示这个关联度在所有事务中占比大不大，如果不大说明这有可能只是偶然的关联，没什么参考价值，也就是这个事务集并不能支持我们的预测，也就是支持度低；如果支持度高，说明这个事务集中经常会出现X和Y相交的情况，这种情况下X和Y相交只是偶然的可能性就会非常小，也就是非常有参考价值。

（2）置信度

这个就比较直白了，就是X和Y的交集占X出现的总数的比值，高的话说明X经常和Y关联，低的话说明X只有一小部分是和Y关联的。

## 2.2 Apriori算法

### 2.2.1 频繁项目集的生成

（1）向下封闭属性：如果一个项集满足某个最小支持度要求，那么这个项集的所有非空子集必须满足这个最小支持度。

反过来说，也就是只要有一个项集的支持度不满足要求，那么以这个项集作为子集的所有项集都是不满足最小支持度要求的（因为只要一个项集I1出现的次数少，那么包括的它的项集I2肯定出现的次数更少）

（2）Apriori算法的频繁项目集的生成分两步：先扎到候选项集，再从候选项集中找到所有的频繁项目集。而候选项集的生成使用Candidata-gen函数，这个函数非常简单粗暴：将所有的长度为k的频繁项目集Fk进行组合，找到所有长度为k+1的候选项目集Ck+1。

注意：在生成c的时候使用的是前面k-1项相同的f1和f2，为什么要前面k-1项相同呢，举个例子，假如要生成的是{1,2,3,4}，那么由向下封闭属性可知：{1,2,3}，{1,2,4}，{1,3,4}，{2,3,4}也必然是频繁项目集，这个时候如果没有前面的限制，那么我们可以由{1,2,3}和{1,2,4}来生成{1,2,3,4}，也可以由{1,2,3}和{1,3,4}来生成{1,2,3,4}，可以的方式有很多种，那么我们直接约束他只使用第一种方式，一是不会重复，而是大大节省资源。

### 2.2.2 关联规则的生成

如果(f-a)->a是一条关联规则，那么(f-asub)->asub也是一条关联规则。

原因很简单，因为分子不变，分母变小了（项多了就小了）。

生成算法使用上面这条规律，一边生成m-后件项集，一边生成m-后件关联规则：从1-后件项集开始，同时先生成1-后件关联规则，由上面那条规则可知，只要一个m-后件项集不能成为关联规则，那么以它为子集的项集也不能成为关联规则，所以可以从Hm递推出Hm+1。然后再在Hm+1中寻找m+1后件关联规则就好了。注意k是需要从1到最大长度遍历的。

## 2.4 多最小支持度的关联规则挖掘

为每个项目都设定同样的最小支持度显然是不现实的，因为有的项目可能频繁出现，有的项目经常性的成对出现，而如果我们研究的是稀有项目，也就是相比于其他的项目，它们出现的频率非常少，这个时候就需要为每个项目设置不同的最小项目支持度(Minimum Item Support,MIS)，不仅可以解决稀有项目的关联规则挖掘问题，同时还可以避免非常多的因为最小支持度和支持度非常高但是参考意义不大的项目集。

### 2.4.1 扩展模型

为了不产生同时包括频繁项目和稀有项目的项集作为频繁项目集，同时不影响向下封闭属性，引入支持度差别限制（Support Difference Constraint）：对于每一个项集，需满足：

其中0<=φ<=1,对每个项集都相同。

### 2.4.2 挖掘算法（MS-Apriori算法）

(1)生成L，L的作用只是在生成F1和C2的时候使用。首先因为在生成F1时，如果一个项目在L（1）之前同时不满足MIS，那么它不管是在生成F1还是C2的时候都不可能用到。但是如果一个不满足自己的MIS的项目在L（1）之后，那么它有可能因为满足前面项目的MIS而可能出现在C2中，正因为这个原因，所以C2需要由L而不是F1来生成。所以生成L的方法如书本所示。

1. **代码实现：  
   Apriori算法：**

C=[]

F=[]

T=["西瓜","茄子","奶酪","冰箱","手机","电脑","牛奶","杯子"]

datas=[["西瓜","茄子","奶酪"],

        ["电脑","手机"],

        ["西瓜","牛奶","杯子"],

        ["手机","冰箱","电脑"],

        ["茄子","牛奶","奶酪","杯子"],

        ["奶酪","牛奶"],

        ["冰箱","奶酪","牛奶"],

        ["牛奶","杯子"],

        ["西瓜","奶酪"],

        ["手机","电脑"],

        ["手机"],

        ["电脑"]]

MIS=0.3

def init\_pass(T):

    res=[]

    for data in datas:

        if len(data)==1:

            res.append(data)

    return res

def candidate\_gen(F,k):

    C=[]

    for f1 in F:

        for f2 in F:

            if f1[:k-1]==f2[:k-1] and f1[k-1]<f2[k-1]:

                f1.append(f2[k-1])

                f=0

                for i in range(2\*\*k-1,2\*\*k):

                    num=0

                    com=[]

                    for j in range(i):

                        if (i>>j)%2:

                            num+=1

                            com.append(f1[j])

                    if num==k-1 and C[k].count(com)==0:

                        f=1

                        break

                if f==0:

                    C.append(f1)

    return

def Apriori(T):

    total=0

    cnt = {}

    cnt.fromkeys(T[, 0])

    for data in datas:

        total+=len(data)

        for item in data:

            cnt[item]+=1

    C.append(init\_pass(T))

    F.append([])

    for item in C[0]:

        if cnt[item[0]]\*1.0/total>=MIS:

            F[0].append(item)

    k = 0

    cntc={}

    while len(F[k])!=0:

        C.append(candidate\_gen(F,k))

        for t in T:

            for c in C:

                if c.count(T)>0:

                    if cntc.get(c)==None:

                        cntc[c]=1

                    else:

                        cntc[c]+=1

            if cntc[c]\*1.0/total>=MIS:

                F[k].append(c)