菊安酱的机器学习第10期

菊安酱的直播间：https://live.bilibili.com/14988341

每周一晚8:00 菊安酱和你不见不散哦~(^o^)/~

更新日期：2019-1-7

作者：菊安酱

课件内容说明：

- 本文为作者原创，转载请注明作者和出处
- 如果想获得此课件及完整视频，可扫描下方二维码，回复"k"进群
- 若有任何疑问，请给作者留言。

完整版视频及课件：http://edu.cda.cn/course/966

12期完整版课纲

直播时间：每周一晚8:00

直播内容：

时间	期数	算法
2018/11/05	第1期	k-近邻算法
2018/11/12	第2期	决策树
2018/11/19	第3期	朴素贝叶斯
2018/11/26	第4期	Logistic回归
2018/12/03	第5期	支持向量机
2018/12/10	第6期	AdaBoost 算法
2018/12/17	第7期	线性回归
2018/12/24	第8期	树回归
2018/12/28	第9期	K-均值聚类算法
2019/01/07	第10期	Apriori 算法
2019/01/14	第11期	FP-growth 算法
2019/01/21	第12期	奇异值分解SVD

Apriori算法

一、关联分析概述

大型超市有海量的交易数据，作为精明的商家肯定不会放弃对这些海量数据的应用，他们希望通过对这些交易数据的分析，了解顾客的购买行为。我们可以通过聚类算法寻找购买相似物品的人群，从而为特定人群提供更具个性化的服务。但是对于超市来讲，更有价值的是找出商品之间的隐藏关联，从而可以用于商品定价、市场促销、存货管理等一系列环节，来增加营业收入。那如何从这种海量而又繁杂的交易数据中找出商品之间的关联呢？当然，你可以使用穷举法，但是这是一种十分耗时且计算代价高的笨方法，所以需要用更智能的方法在合理的时间范围内找到答案。因此，关联分析就此诞生了~

1. 关联分析

关联分析（association analysis）是一种在大规模数据集中寻找有趣关系的非监督学习算法。这种关系可以有两种形式：频繁项集或者关联规则。**频繁项集**（frequent item sets）是经常出现在一块的物品的集合，**关联规则**（association rules）暗示两种物品之间可能存在很强的关系。

关联分析的一个典型例子是购物篮分析。下面我们一起来看一个简单例子。

交易号码	商品
001	豆奶，莴苣
002	莴苣，尿布，啤酒，甜菜
003	豆奶，尿布，啤酒，橙汁
004	莴苣，豆奶，尿布，啤酒
005	莴苣，豆奶，尿布，橙汁

交易号码代表交易流水号，商品代表一个顾客一次购买的全部商品。

在这里需要明确几个定义：

- 事务：每一条交易称为一个事务。例如，在这个例子中包含了5个事务。
- 项：交易的每一个物品称为一个项，例如豆奶、尿布等。
- 项集：包含零个或者多个项的集合叫做项集，例如 {豆奶，莴苣}。
- k-项集：包含k个项的项集叫做k-项集。例如 {豆奶} 叫做1-项集，{豆奶，尿布，啤酒} 叫做3-项集。
- 前件和后件：对于规则{尿布}→{啤酒}，{尿布} 叫做前件，{啤酒} 叫做后件。

啤酒与尿布的故事

关联分析中最有名的例子是"啤酒与尿布"。你打开百度搜索一下，就会发现很多人都在对“啤酒与尿布”的故事津津乐道，可以说100个人就有100个版本的 “啤酒与尿布”的故事。
故事的时间跨度从上个世纪80年代到本世纪初，甚至连故事的主角和地点都会发生变化——从美国跨越到欧洲。认真地查了一下资料，我们发现沃尔玛的 “啤酒与尿布”案例是正式刊登在1998年的《哈佛商业评论》上面的，这应该算是目前发现的最权威报道。
“啤酒与尿布”的故事产生于20世纪90年代的美国沃尔玛超市中，沃尔玛的超市管理人员分析销售数据时发现了一个令人难于理解的现象：在某些特定的情况下，“啤酒”与“尿布”两件看上去毫无关系的商品会经常出现在同一个购物篮中，这种独特的销售现象引起了管理人员的注意，经过后续调查发现，这种现象出现在年轻的父亲身上。
在美国有婴儿的家庭中，一般是母亲在家中照看婴儿，年轻的父亲前去超市购买尿布。父亲在购买尿布的同时，往往会顺便为自己购买啤酒，这样就会出现啤酒与尿布这两件看上去不相干的商品经常会出现在同一个购物篮的现象。如果这个年轻的父亲在卖场只能买到两件商品之一，则他很有可能会放弃购物而到另一家商店，直到可以一次同时买到啤酒与尿布为止。
沃尔玛发现了这一独特的现象，开始在卖场尝试将啤酒与尿布摆放在相同的区域，让年轻的父亲可以同时找到这两件商品，并很快地完成购物；而沃尔玛超市也可以让这些客户一次购买两件商品、而不是一件，从而获得了很好的商品销售收入，这就是“啤酒与尿布”故事的由来。

2. 频繁项集的评估标准

频繁项集是经常出现在一块的物品的集合，那么问题就来了：第一，当数据量非常大的时候，我们无法凭肉眼找出经常出现在一起的物品，这就催生了关联规则挖掘算法，比如 Apriori、PrefixSpan、CBA 等。第二是我们缺乏一个频繁项集的标准。比如10条记录，里面A和B同时出现了三次，那么我们能不能说A和B一起构成频繁项集呢？因此我们需要一个评估频繁项集的标准。

常用的频繁项集的评估标准有支持度,置信度和提升度三个。

2.1 支持度

支持度就是几个关联的数据在数据集中出现的次数占总数据集的比重。或者说几个数据关联出现的概率。如果我们有两个想分析关联性的数据X和Y，则对应的支持度为:

$$Support(X,Y) = P(XY) = \frac{number(XY)}{num(AllSamples)}$$

比如上面例子中，在5条交易记录中{尿布, 啤酒}总共出现了3次，所以

$$Support(\text{尿布，啤酒}) = \frac{3}{5} = 0.6$$

以此类推，如果我们有三个想分析关联性的数据X，Y和Z，则对应的支持度为:

$$Support(X,Y,Z) = P(XYZ) = \frac{number(XYZ)}{num(AllSamples)}$$

一般来说，支持度高的数据不一定构成频繁项集，但是支持度太低的数据肯定不构成频繁项集。 另外，支持度是针对项集来说的，因此，可以定义一个最小支持度，而只保留满足最小支持度的项集，起到一个项集过滤的作用。

2.2 置信度

置信度体现了一个数据出现后，另一个数据出现的概率，或者说数据的条件概率。如果我们有两个想分析关联性的数据X和Y，X对Y的置信度为

$$Confidence(X \rightarrow Y) = P(X|Y) = P(XY)/P(Y)$$

比如上面例子中，豆奶对莴苣的置信度=豆奶和莴苣同时出现的概率/莴苣出现的概率，则有

$$P(\text{豆奶 * 莴苣}) = \frac{3}{5} = 0.6$$
$$P(\text{莴苣}) = \frac{4}{5} = 0.8$$
$$Confidence(\text{豆奶} \rightarrow \text{莴苣}) = \frac{P(\text{豆奶 * 莴苣})}{P(\text{莴苣})} = \frac{0.6}{0.8} = 0.75$$

也可以以此类推到多个数据的关联置信度，比如对于三个数据X，Y，Z,则X对于Y和Z的置信度为：

$$Confidence(X \rightarrow YZ) = P(X|YZ) = P(XYZ)/P(YZ)$$

为什么使用支持度和置信度？支持度是一种重要度量，因为支持度很低的规则可能只是偶然出现。从商务角度来看，低支持度的规则多半也是无意义的，因为对顾客很少同时购买的商品进行促销可能并无益处。因此，支持度通常用来删去那些无意义的规则。此外，支持度还具有一种期望的性质，可以用于关联规则的有效发现。

另一方面，置信度度量通过规则进行推理具有可靠性。对于给定的规则*X*→*Y*，置信度越高，*Y*在包含*X*的事务中出现的可能性就越大。置信度也可以估计*Y*在给定*X*下的条件概率。

同时，应当小心解释关联分析的结果。由关联规则作出的推论并不必然蕴涵因果关系。它只表示规则前件和后件同时出现的一种概率。

2.3 提升度

提升度表示含有Y的条件下同时含有X的概率，与X总体发生的概率之比，也就是X对Y的提升度与X总体发生的概率之比，即:

$$Lift(X \rightarrow Y) = P(X|Y)/P(X) = Confidence(X \rightarrow Y)/P(X)$$

提升度体先了X和Y之间的关联关系, 提升度大于1则$X \Leftarrow Y$是有效的强关联规则， 提升度小于等于1则$X \Leftarrow Y$是无效的强关联规则 。一个特殊的情况，如果X和Y独立，则有$Lift(X \Leftarrow Y) = 1$，因为此时$P(X|Y) = P(X)$。

一般来说，要选择一个数据集合中的频繁数据集，则需要自定义评估标准。最常用的评估标准是用自定义的支持度，或者是自定义支持度和置信度的一个组合。

3. 关联规则发现

给定事务的集合*T*，关联规则发现是指找出支持度大于等于$minsup$并且置信度大于等于 $minconf$ 的所有规则，其中 $minsup$ 和 $minconf$ 是对应的支持度和置信度阈值。挖掘关联规则的一种原始方法是：计算每个可能规则的支持度和置信度。但是这种方法的代价很高，令人望而却步，因为可以从数据集提取的规则的数目达指数级。更具体地说，从包含d个项的数据集提取的可能规则的总数为:

$$R = 3^d - 2^{d+1} + 1$$

【公式证明过程】可参考：https://blog.csdn.net/wxbmelisky/article/details/48268833

对于我们前面聚德小例子，里面一共有6中商品，提取的可能规则数为：$3^6 - 2^7 + 1 = 602$，也就是说对于只有6种商品的小数据集都需要计算602条规则的支持度和置信度。使用*minsup*= 20%和*minconf*=50%，80%以上的规则将被丢弃，使得大部分计算是无用的开销。为了避免进行不必要的计算，事先对规则剪枝，而无须计算它们的支持度和置信度的值将是有益的。 因此，大多数关联规则挖掘算法通常采用的一种策略是，将关联规则挖掘任务分解为如下两个主要的子任务。

- 频繁项集产生：

  其目标是发现满足最小支持度阈值的所有项集，这些项集称作频繁项集（frequent itemset）。

- 规则的产生：

  其目标是从上一步发现的频繁项集中提取所有高置信度的规则，这些规则称作强规则（strong rule）。

通常，频繁项集产生所需的计算开销远大于产生规则所需的计算开销。那有没有办法可以减少这种无用的计算呢？有。下面这两种方法可以降低产生频繁项集的计算复杂度：

（1）减少候选项集的数目M。

（2）减少比较次数。替代将每个候选项集与每个事务相匹配，可以使用更高级的数据结构，或者存储候选项集或者压缩数据集，来减少比较次数。

这些策略将在Apriori算法基本思想中进行讨论。

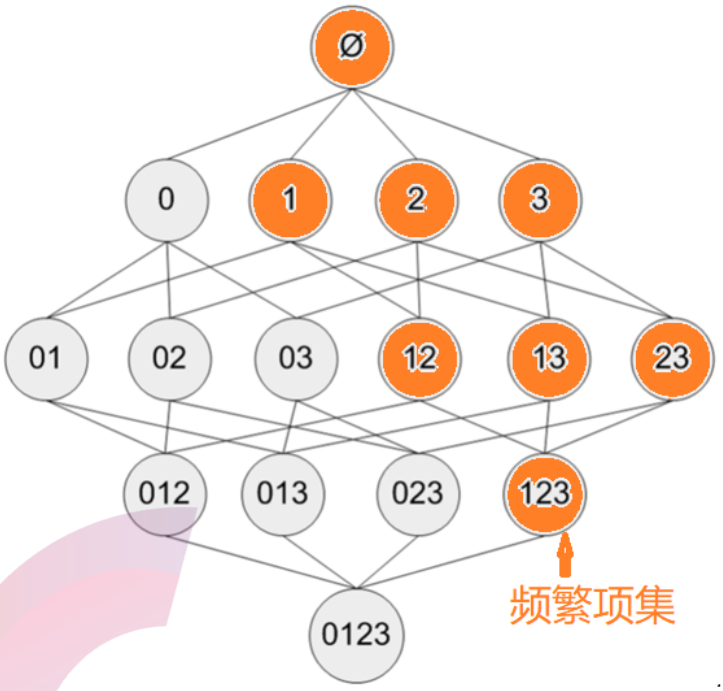
二、Apriori算法原理

先验原理 如果一个项集是频繁的，则它的所有子集一定也是频繁的。

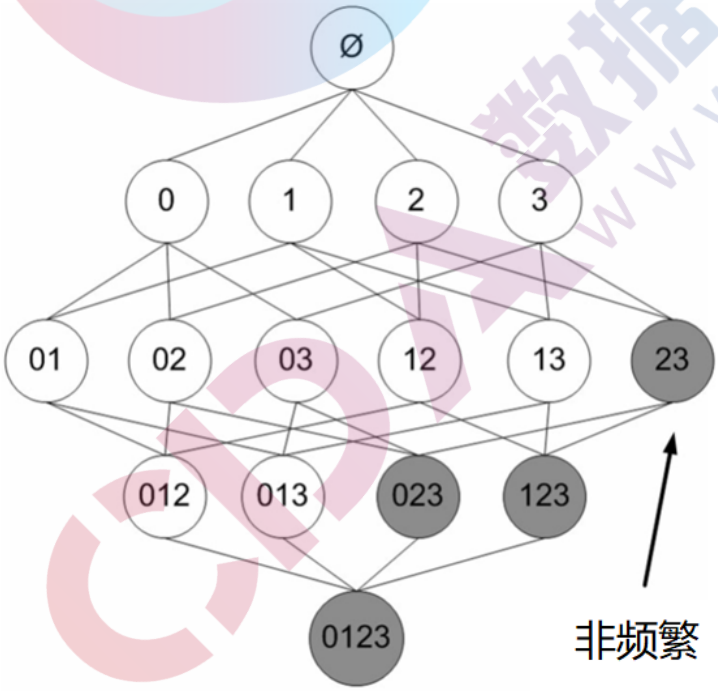
假设我们在经营一家商品种类并不多的杂货店，我们对那些经常在一起被购买的商品非常感兴趣。我们只有4种商品：商品0，商品1，商品2和商品3。那么所有可能被一起购买的商品组合都有哪些？这些商品组合可能只有一种商品，比如商品0，也可能包括两种、三种或者所有四种商品。我们并不关心某人买了两件商品0以及四件商品2的情况，我们只关心他购买了一种或多种商品。

下图显示了物品之间所有可能的组合。为了让该图更容易懂，图中使用物品的编号0来取代物品0本身。另外，图中从上往下的第一个集合是ϕ，表示空集或不包含任何物品的集合。物品集合之间的连线表明两个或者更多集合可以组合形成一个更大的集合。

根据先验原理，假如 {1,2,3} 是频繁项集，那么它的所有子集（下图中橘色项集）一定也是频繁的。



这个先验原理直观上并没有什么帮助，但是反过来看就有用了，也就是说如果一个项集是非频繁项集，那么它的所有超集也是非频繁的（如下图所示）。

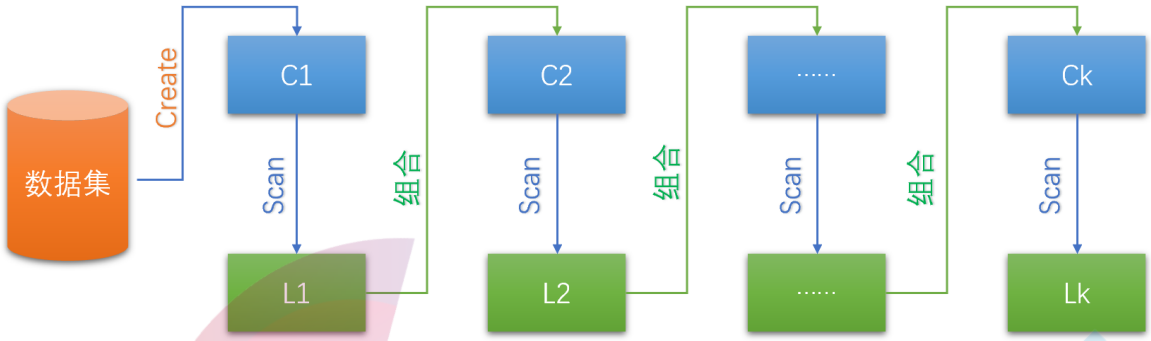


上图中，已知阴影项集 {2,3} 是非频繁的。利用这个知识，我们就知道项集{0,2,3} ，{1,2,3}以及{0,1,2,3}也是非频繁的。这也就是说，一旦计算出了{2,3}的支持度，知道它是非频繁的之后，就不需要再计算{0,2,3}、{1,2,3}和{0,1,2,3}的支持度，因为我们知道这些集合不会满足我们的要求。使用该原理就可以避免项集数目的指数增长，从而在合理时间内计算出频繁项集。

三、使用Apriori算法来发现频繁项集

关联分析的目的包括两项：发现频繁项集和发现关联规则。首先需要找到频繁项集，然后才能获得关联规则。

Apriori 算法过程



C1，C2，...，Ck分别表示1-项集，2-项集，...，k-项集；

L1，L2，...，Lk分别表示有k个数据项的频繁项集。

Scan表示数据集扫描函数。该函数起到的作用是支持度过滤，满足最小支持度的项集才留下，不满足最小支持度的项集直接舍掉。

下面我们用python代码来实现这一过程~

1. 生成候选项集

在使用python来对整个程序编码之前，需要创建一些辅助函数。

数据集扫描的伪代码如下：

```
对数据集中的每条交易记录transaction
对每个候选项集can：
    检查can是否是transaction的子集：
    如果是，增加can的计数
对每个候选项集：
    如果支持度不低于最小值，则保留该项集
    返回所有频繁项集列表
```

首先，我们创建一个简单数据集，来帮助我们建模

函数1：创建一个用于测试的简单数据集

```
def loadDataSet():
    dataSet = [[1,3,4],[2,3,5],[1,2,3,5],[2,5]]
    return dataSet
```

运行函数，结果如下：

```
dataSet = loadDataSet()
dataSet
```

函数2：构建第一个候选集合C1。

由于算法一开始是从输入数据集中提取候选项集列表，所以这里需要一个特殊的函数来处理——frozenset类型。frozenset是指被"冰冻"的集合，也就是用户不能修改他们。这里必须要用frozenset而非set类型，是因为后面我们要将这些集合作为字典键值使用。

该函数流程是：首先创建一个空列表，用来储存所有不重复的项值。接下来遍历数据集中所有的交易记录，对每一条交易记录，遍历记录中的每一个项。如果该项没有在C1中出现过，那么就把它添加到C1中，这里需要注意的是，并非简单地添加物品项，而是添加只包含该物品项的一个集合（此处用集合或者列表都可以）。循环完毕后，对整个C1进行排序并将其中每个单元素集合映射到frozenset()，最后返回frozenset的列表。

```
"""
函数功能：生成第一个候选集合C1
参数说明：
    dataSet：原始数据集
返回：
    frozenset形式的候选集合C1
"""
def createC1(dataSet):
    C1 = []
    for transaction in dataSet:
        #print(transaction)
        for item in transaction:
            #print(item)
            if not {item} in C1:
                #print({item})
                C1.append({item})
                #print(C1)
    C1.sort()
    return list(map(frozenset, C1))
```

运行函数，查看结果：

```
C1=createC1(dataSet)
C1
```

函数3：生成满足最小支持度的频繁项集L1。

C1是大小为1的所有候选项集的集合，Apriori算法首先构建集合C1，然后扫描数据集来判断这些只有一个元素的项集是否满足最小支持度的要求。那些满足最低要求的项集构成集合L1。

```
"""
函数功能：生成满足最小支持度的频繁项集L1
参数说明：
    D:原始数据集
```

```
    Ck:候选项集
    minSupport:最小支持度
返回:
    retList: 频繁项集
    supportData: 候选项集的支持度
"""
def scanD(D, Ck, minSupport):
    ssCnt = {}
    for tid in D:
        for can in Ck:
            if can.issubset(tid):  #判断can是否是tid的子集，返回的是布尔型数据
                if can not in ssCnt.keys():
                    ssCnt[can] = 1
                else:
                    ssCnt[can] += 1
    numItems = float(len(D))
    retList= []
    supportData = {}          #候选集项Ck的支持度字典(key:候选项， value:支持度)
    for key in ssCnt:
        support = ssCnt[key] / numItems
        supportData[key] = support
        if support >= minSupport:
            retList.append(key)
    return retList, supportData
```

将数据集dataSet带入函数，查看运行结果：

```
L1, supportData = scanD(dataSet, C1, 0.5)
L1
supportData
```

```
In [11]: L1, supportData = scanD(dataSet, C1, 0.5)

In [12]: L1

Out[12]: [frozenset({1}), frozenset({2}), frozenset({3}), frozenset({5})]

In [13]: supportData

Out[13]: {frozenset({1}): 0.5,
          frozenset({2}): 0.75,
          frozenset({3}): 0.75,
          frozenset({5}): 0.75,
          frozenset({4}): 0.25}
```

从运行结果中可以看出，设定最小支持度为0.5时，frozenset({4})支持度为0.25<0.5，就被舍弃了，没有放入到L1中。

2. 组织完整的Apriori算法

这个Apriori算法的伪代码如下：

```
当集合中项的个数大于0时：
    构建一个k-项集组成的列表
    检查数据确保每个项集都是频繁的
    保留频繁项集并构建(k+1)-项集组成列表
```

函数1：输入参数为频繁项集Lk与项集元素个数k，输出为Ck.

```
def aprioriGen(Lk, k):
    Ck = []
    lenLk = len(Lk)
    for i in range(lenLk):
        for j in range(i + 1, lenLk):
            #前k-2个项相同时，将两个集合合并
            L1 = list(Lk[i])[:k - 2]
            L1.sort()
            L2 = list(Lk[j])[:k - 2]
            L2.sort()
            if L1 == L2:
                Ck.append(Lk[i] | Lk[j])
    return Ck
```

【解惑】这里的k-2获取会让人有点困惑。加入我们要利用 {0}、{1}、{2} 构建 {0,1}、{0,2}、{1,2} 时实际上试讲单个项组合到一块。现在利用 {0,1}、{0,2}、{1,2}来创建三元素项集，如果两两组合的话，会得到{0,1,2}、{0,1,2}、{0,1,2}三个一模一样的结果。也就是说，同样的结果集合会重复3次。而我们需要的是尽可能少的遍历列表。现在，如果比较集合 {0,1}、{0,2}、{1,2}的第一个元素并只对第一个元素相同的集合（即 {0,1}、{0,2}）进行并集操作的话，就可以得到{0,1,2}，在这个过程中就执行了一次操作！这样就不需要遍历列表来寻找非重复值啦。

函数2：根据数据集和支持度，返回所有的频繁项集，以及所有项集的支持度。

```
def apriori(D, minSupport = 0.5):
    C1 = createC1(D)
    L1, supportData = scanD(D, C1, minSupport)
    L = [L1]
    k = 2
    while (len(L[k-2]) > 0):
        Ck = aprioriGen(L[k-2], k)
        #print("ck:",Ck)
        Lk, supK = scanD(D, Ck, minSupport)
        supportData.update(supK)
        #print("lk:", Lk)
        L.append(Lk)
        k += 1
    return L, supportData
```

运行函数，查看结果：

```
dataset = loadDataSet()
L, supportData = apriori(dataset, minSupport=0.5)
```

In [27]:
```
dataset = loadDataSet()
L, supportData = apriori(dataset, minSupport=0.5)
```

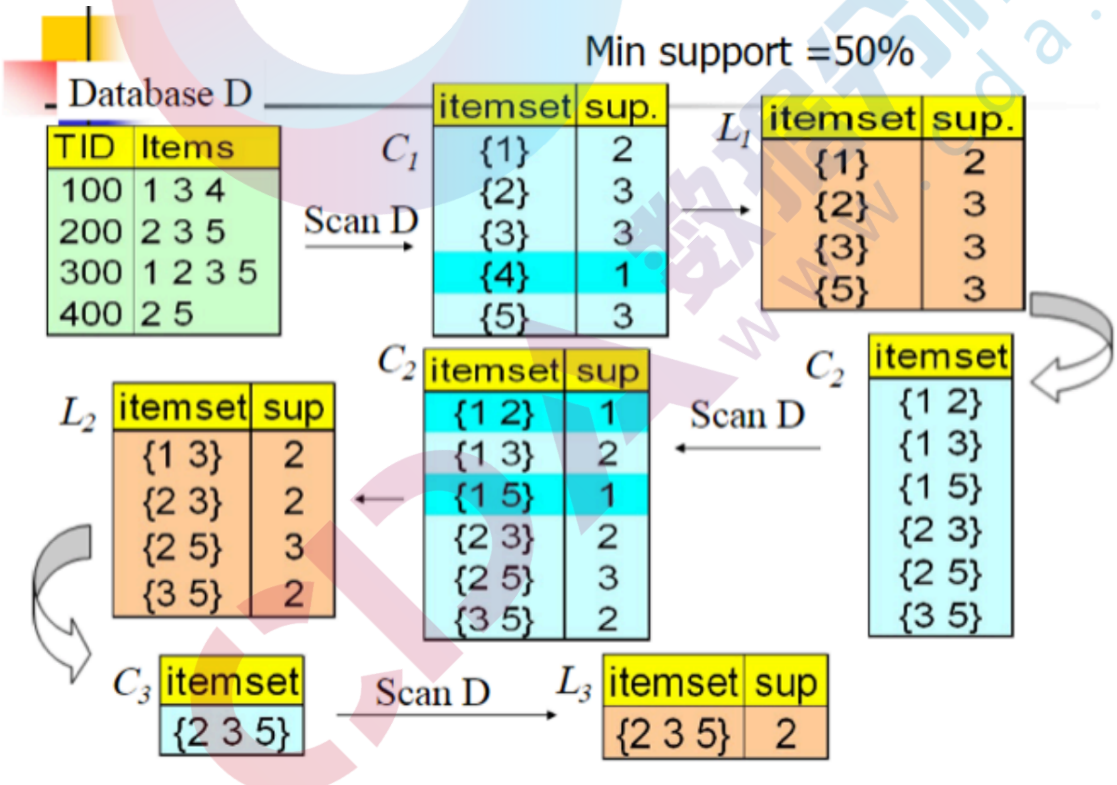
In [28]:
```
L
```

Out[28]:
```
[[frozenset({1}), frozenset({3}), frozenset({2}), frozenset({5})],
 [frozenset({1, 3}), frozenset({2, 3}), frozenset({3, 5}), frozenset({2, 5})],
 [frozenset({2, 3, 5})],
 []]
```

In [29]:
```
supportData
```

Out[29]:
```
{frozenset({1}): 0.5,
 frozenset({3}): 0.75,
 frozenset({4}): 0.25,
 frozenset({2}): 0.75,
 frozenset({5}): 0.75,
 frozenset({1, 3}): 0.5,
 frozenset({2, 3}): 0.5,
 frozenset({3, 5}): 0.5,
 frozenset({2, 5}): 0.75,
 frozenset({1, 2}): 0.25,
 frozenset({1, 5}): 0.25,
 frozenset({2, 3, 5}): 0.5}
```

其实上面这一系列函数实现的功能可以用下图表示：



【完整版】四、挖掘关联规则

【完整版】五、案例：发现美国国会投票中的模式

【完整版】六、案例：发现毒蘑菇的相似特征

其他

- 菊安酱的直播间：https://live.bilibili.com/14988341
- 下周一（2019/1/14）将讲解**FP-growth算法**，欢迎各位三连哈~
- 如有问题，可以给我留言哦~