关联分析

课前准备

- 下载Anaconda软件，请点击这里进行下载。
- 安装efficient-apriori库。
  - `pip install efficient-apriori`

本节要点

- 关联分析的相关概念与意义。
- 关联分析的步骤与实现。
  - 寻找频繁项集。
  - 生成关联规则。

啤酒与尿布

在美国有婴儿的家庭中，通常是母亲在家中照看婴儿，父亲去超市为婴儿购买尿布。当丈夫在为孩子购买尿布的同时，也通常购买自己爱喝的啤酒。因此，沃尔玛超市发现这一规律后，将啤酒与尿布放在相同的区域，使得父亲可以同时买到这两件商品，从而提高啤酒与尿布的销售量。

日常中，在我们去超市进行购物的时候，也经常见到超市将某些商品捆绑在一起进行销售。然而，这些捆绑的依据是什么，超市又是如何发现这些规律的呢？

关联分析

关联分析定义

实际上，超市这种销售的行为不是偶然的，而是长期从顾客的大量订单中分析，从而得出的结论。**关联分析**，就是从大规模数据中，发现对象之间隐含关系与规律的过程，也称为**关联规则学习**。例如，{啤酒 -> 尿布}就是一个关联规则。

应用场景

- 超市购物分析
- 图书购买分析
- 服装搭配分析
- 交通事故分析
- 疾病症状分析
- 社交关系分析

关联分析相关概念

项与项集

项，指我们分析数据中的一个对象。而项集就是由若干项构成的集合。

例如，在如下水果购物清单中：

购物清单
苹果，香蕉，葡萄
苹果，桔子
苹果，火龙果
葡萄，桔子
香蕉，葡萄，梨
葡萄，梨，火龙果

苹果，香蕉等每一个水果对象，都是一个项。而一个或更多水果（项）构成的集合，就是项集。例如，{葡萄}，{香蕉，梨}都是项集。

支持度

支持度为某项集在数据集中出现的频率。即项集在记录中出现的次数，除以数据集中所有记录的数量。

$$support(A) = \frac{count(A)}{count(dataset)} = P(A)$$

支持度体现的是某项集的频繁程度，只有某项集的支持度达到一定程度，我们才有研究该项集的必要。

项集{香蕉，葡萄}的支持度是（ ）。

A 1 / 6

B 1 / 3

C 1 / 2

D 2 / 3

置信度

关联规则{A -> B}中，**置信度**为A与B同时出现的次数，除以A出现的次数。

$$confidence(A->B) = \frac{count(AB)}{count(A)}$$

$$= \frac{count(AB) \ / \ count(dataset)}{count(A) \ / \ count(dataset)}$$

$$= \frac{P(AB)}{P(A)}$$

$$= P(B \mid A)$$

置信度体现的是关联规则的可靠程度，如果关联规则{A -> B}的置信度较高，则说明当A发生时，B有很大概率也会发生，这样就可能会带来研究价值。

提升度

关联规则{A -> B}中，**提升度**为{A -> B}的置信度，除以B的支持度。

$$lift(A->B) = \frac{confidence(A->B)}{support(B)}$$

$$= \frac{P(B|A)}{P(B)}$$

$$= \frac{P(AB)}{P(A)P(B)}$$

提升度体现的是组合（应用关联规则）相对不组合（不应用关联规则）的比值，如果提升度大于1，则说明应用该关联规则是有价值的。如果提升度小于1，说明应用该关联规则起到了负面影响。因此，我们应该尽可能让关联规则的提升度大于1，提升度越大，则应用该关联规则的效果越好。

如果你是水果店经理，为了提高利润，促进葡萄的销售，你会使用哪项关联规则?

A {香蕉 -> 葡萄}

B {苹果 -> 葡萄}

C {火龙果 -> 葡萄}

D 以上都可以。

频繁项集

通常情况下，我们只会对频繁出现的项集进行研究。因此，我们会设置一个支持度阈值，如果一个项集的支持度达到（大于等于）该阈值，则该项集就称为**频繁项集**。特别的，如果频繁项集中含有k个元素，我们称之为频繁k项集。

关联分析过程

关联分析可以分为如下两个过程：

1. 从数据集中寻找频繁项集。
2. 从频繁项集中生成关联规则。

寻找频繁项集

首先，我们需要能够找到所有的频繁项集，即经常出现在一起的对象集合。实际上，找到频繁项集并不复杂，我们只需要按照如下的步骤来进行操作即可：

1. 遍历对象之间所有可能的组合（包括单个对象的组合），每种组合构成一个项集。
2. 针对每一个项集A，计算A的支持度（A出现的次数除以记录总数）。
3. 返回所有支持度大于指定阈值的项集。

上面发现频繁项集的方式正确吗?
A 理论正确，实际应用也可行。
B 理论正确，实际应用不可行。
C 理论不正确，实际应用可行。
D 理论不正确，实际应用也不可行。

Apriori算法原理

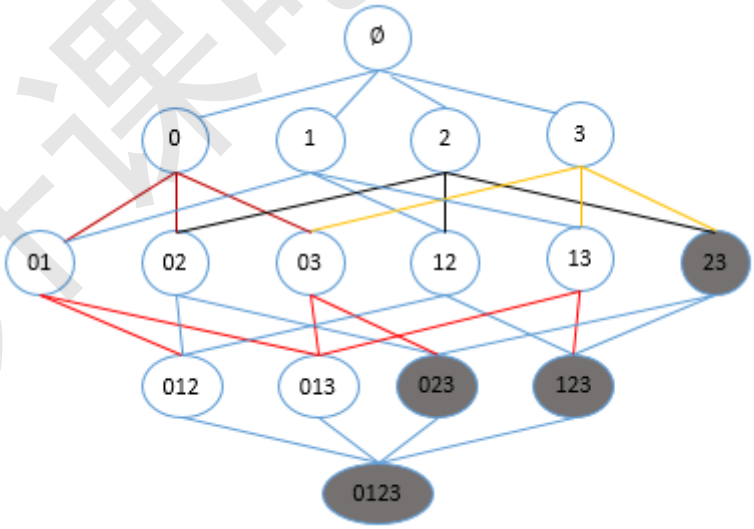
以上的理论是没有问题的，但是，却很难在实际应用中使用。因为，对象之间任意组合构成的项集，数量可能非常大。例如，在上图中，4个不同的对象（项），就可以构成15种组合。而对于含有N个对象的数据集，总共可以构成$2^N - 1$种组合，这是一个非常大的数字。

因此，为了降低计算量，我们使用Apriori算法原理进行优化。Apriori算法原理可以解释如下：

1. 如果一个项集是频繁项集，则其所有子集（非空）也是频繁项集。
2. 如果一个项集（非空）是非频繁项集，则其所有父集也是非频繁项集。

Apriori算法会从$k=1$开始，使用两个k项集进行组合，从而产生$k+1$项集。结合之前介绍的算法原理，我们可知，频繁$k+1$项集是由两个k项集组合而成，而对于频繁$k+1$项集来说，其所有的k项集子集必然都是频繁项集，这就意味着，频繁$k+1$项集只可能从两个频繁k项集组合产生，因此，当我们在组合的过程中，一旦发现某个k项集不是频繁项集（支持度小于指定的阈值），就可以将其移除，而无需再参与后续生成$k+1$项集的组合。这样一来，就可以大大减少计算量。

例如在图中，假设{2，3}是非频繁项集，则根据Apriori算法原理，其所有父集也是非频繁项集，故{0,2,3}，{1,2,3}与{0,1,2,3}也是非频繁项集。因此，我们就无需使用{2，3}与其他2项集进行组合，去生成3项集了（因为生成的所有3项集都是非频繁项集）。



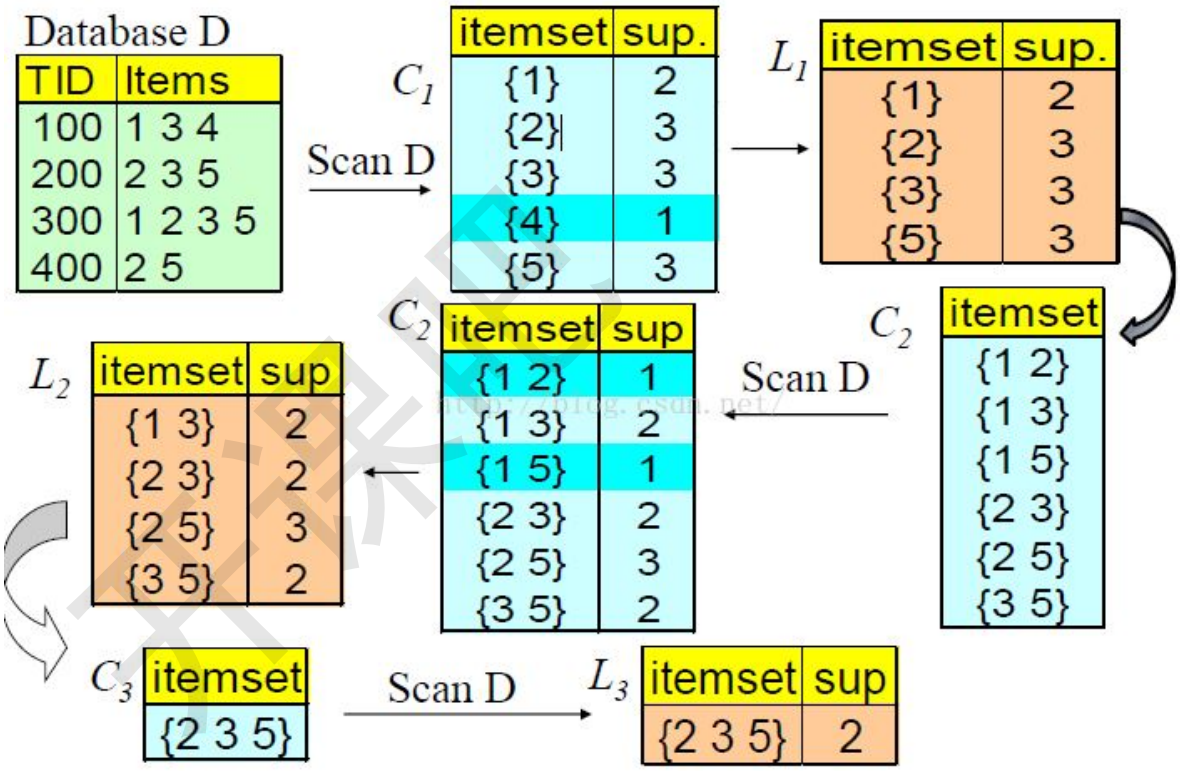
Apriori算法流程

Apriori算法流程如下：

1. 扫描数据集，从数据集中生成候选k项集C_k（k从1开始）。
2. 计算C_k中，每个项集的支持度，删除低于阈值的项集，构成频繁项集L_k。
3. 将频繁项集L_k中的元素进行组合，生成候选$k+1$项集C_{k+1}
4. 重复步骤2,3，直到满足以下两个条件之一时，算法结束。
    - 频繁k项集无法组合生成候选$k+1$项集。
    - 所有候选k项集支持度都低于指定的阈值（最小支持度），无法生成频繁k项集。

说明：

- C_k：所有的候选k项集
- L_k：所有的频繁k项集。



生成关联规则

当产生频繁项集后，生成关联规则会相对简单。我们只需要将每个频繁项集拆分成两个非空子集，然后使用这两个子集，就可以构成关联规则。当然，一个频繁项集拆分两个非空子集可能有很多种方式，我们要考虑每一种不同的可能。例如，频繁项集{1, 2, 3}可以拆分为：

{1 -> 2, 3}
{2 -> 1, 3}
{3 -> 1, 2}
{1, 2 -> 3}
{1, 3 -> 2}
{2, 3 -> 1}

然后，我们针对每一个关联规则，分别计算其置信度，仅保留符合最小置信度的关联规则。

程序实现

加载数据集

首先，我们需要从文件中读取数据。

```
def load_data(path):
    """读取指定路径的文件。

    Parameters
    ----------
    path : str
        文件所在的路径。

    Returns
    -------
    content : list
        文件的内容。
```

```
    """

    content = []
    with open(path, encoding="UTF-8") as f:
        for line in f:
            line = line.strip("\n")
            content.append(line.split(","))
    return content


# dataset是一个二维列表，每个元素（一维列表）保存每个购物清单的商品记录。
dataset = load_data("data.txt")
print(len(dataset))
```

```
7501
```

然后，我们来浏览数据的大致形式，由于记录较多，我们这里仅查看前10条记录（订单）。

```
# print(dataset[:10])
for i in range(10):
    print(i + 1, dataset[i], sep="->")
```

```
1->['虾', '杏仁', '鳄梨', '混合蔬菜', '绿葡萄', '全麦面粉', '山药', '农家干酪', '功能饮料', '番茄汁', '低脂酸奶', '绿茶', '蜂蜜', '沙拉', '矿泉水', '三文鱼', '抗氧化剂果汁', '冷冻果汁', '菠菜', '橄榄油']
2->['汉堡', '肉丸', '鸡蛋']
3->['酸辣酱']
4->['火鸡', '鳄梨']
5->['矿泉水', '牛奶', '能量条', '全麦大米', '绿茶']
6->['低脂酸奶']
7->['全麦意大利面', '炸薯条']
8->['汤', '淡奶油', '青葱']
9->['冷冻蔬菜', '意大利细面条', '绿茶']
10->['炸薯条']
```

挖掘关联规则

我们可以通过Apriori算法来发现关联规则，efficient-apriori模块为我们提供了该算法的实现。不过，该模块不是Anoconda内建模块，需要我们自行安装，如下：

`pip install efficient-apriori`

```
from efficient_apriori import apriori

# transactions：交易数据，要求为二维数据结构，每个低维用来存放每个购物清单的商品。
# min_support：最小支持度。
# min_confidence：最小置信度。
# 该函数会返回满足条件的频繁项集与关联规则。
itemsets, rules = apriori(transactions=dataset, min_support=0.05, min_confidence=0.3)
# 输出所有满足条件频繁k项集。
print(itemsets)
# 输出关联规则。
print(rules)
```

```
{1: {('虾',): 536, ('低脂酸奶',): 574, ('矿泉水',): 1788, ('绿茶',): 991, ('橄榄油',): 494, ('冷冻果汁',): 475, ('汉堡',): 654, ('鸡蛋',): 1348, ('火鸡',): 469, ('牛奶',): 972, ('全麦大米',): 439, ('炸薯条',): 1282, ('汤',): 379, ('冷冻蔬菜',): 715, ('意大利细面条',): 1306, ('饼干',): 603, ('食用油',): 383, ('鸡肉',): 450, ('巧克力',): 1229, ('西红柿',): 513, ('胡椒粉',): 557, ('煎饼',): 713, ('奶酪粉',): 393, ('碎牛肉',): 737, ('薄肉片',): 595, ('蛋糕',): 608}, 2: {('矿泉水', '鸡蛋'): 382, ('意大利细面条', '矿泉水'): 448, ('巧克力', '矿泉水'): 395}}
[{意大利细面条} -> {矿泉水}, {巧克力} -> {矿泉水}]
```

apriori函数返回的关联规则列表，每个元素为efficient_apriori.rules.Rule类型。我们可以输出这些对象，能看到更加详细的信息：

- 支持度
- 置信度
- 提升度
- 确信度

其中，给定关联规则{A - B}，**确信度**的计算方式为：

$$conviction(A->B) = \frac{1-support(B)}{1-confidence(A->B)}$$

$$= \frac{P(\bar{B})}{P(\bar{B}|A)}$$

如果一个关联规则的确信度大，则表示该规则的置信度大，并且B经常出现。

```
for r in rules:
    print(r)
```

```
{意大利细面条} -> {矿泉水} (conf: 0.343, supp: 0.060, lift: 1.439, conv: 1.159)
{巧克力} -> {矿泉水} (conf: 0.321, supp: 0.053, lift: 1.348, conv: 1.122)
```

此外，我们还可以通过Rule类提供的属性，来获取对应的信息值。

```
r = rules[0]
print("支持度：", r.support)
print("置信度：", r.confidence)
print("提升度：", r.lift)
print("确信度：", r.conviction)
```

```
支持度： 0.05972536995067324
置信度： 0.3430321592649311
提升度： 1.4390851379453289
确信度： 1.1593136437508424
```

手写算法实现

以上程序为efficient_apriori提供的Apriori算法实现，不过，我们也可以自行编写程序，来实现Apriori算法。关于该算法的实现细节以及讲解，老梁提供辅助视频，供大家扩展学习。

拓展点

- 对本程序，可以改用不同的支持度与置信度，观看结果。
- 自行手写实现Apriori算法。