

华中科技大学

课程实验报告

课程名称： 大数据分析

专业班级： 计科 2003 班
学 号： U202015374
姓 名： 张隽翊
指导教师： 崔金华
报告日期： 2022 年 12 月 28 日

计算机科学与技术学院

目录

实验三 关系挖掘实验.....	1
3.1 实验内容	1
3.2 实验过程	1
3.2.1 编程思路.....	1
3.2.2 遇到的问题及解决方式.....	2
3.2.3 实验测试与结果分析.....	3
3.3 实验总结	5

实验三 关系挖掘实验

3.1 实验内容

1. 实验内容

编程实现 Apriori 算法，要求使用给定的数据文件进行实验，获得频繁项集以及关联规则。

2. 实验要求

以 Groceries.csv 作为输入文件，输出 1~3 阶频繁项集与关联规则，各个频繁项的支持度，各个规则的置信度，各阶频繁项集的数量以及关联规则的总数。

固定参数以方便检查，频繁项集的最小支持度为 0.005，关联规则的最小置信度为 0.5。

3.2 实验过程

3.2.1 编程思路

本实验采用 Apriori 算法实现对给定数据关联规则的分析。大致过程是首先从数据中生成候选一阶项集，统计后得到一阶频繁项集，再生成候选二阶频繁项集，依次类推直到生成三阶频繁项集，最后由一、二、三阶段频繁项集得到关联规则。Apriori 算法的思路如图 1.1 所示。

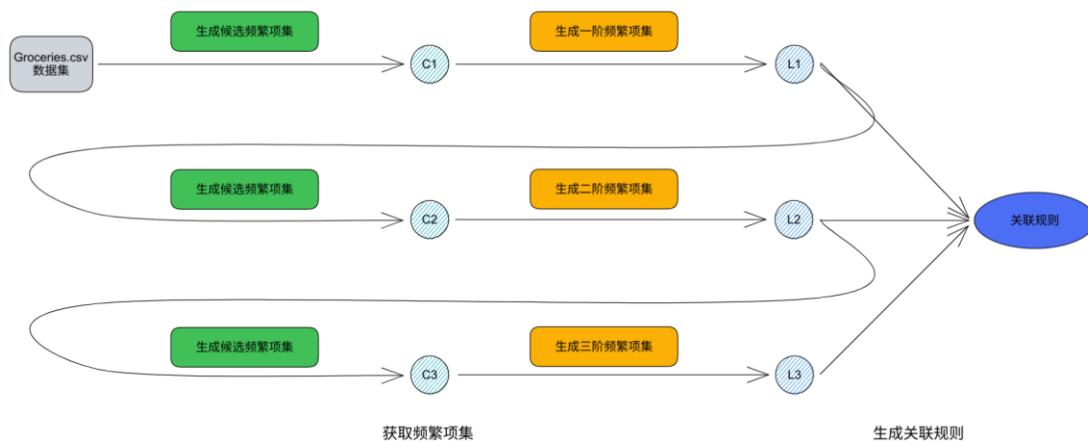


图 1.1 Apriori 算法示意图

(1) 数据集读取

将 Groceries.csv 中的项集信息表示为二级列表的结构，记作 dataset。

(2) 生成频繁项集

用 Python 的 set 集合结构存储频繁项集，用 dict 字典结构存储项集和支持度，具体结构为<frozenset{项集}, 支持度>。 k 阶候选频繁项集记作 C_k ， k 阶频繁项集记作 L_k 。

生成 C_1 比较简单，只需要遍历项集列表，将首次出现的项加入 C_1 即可。

模块化候选频繁项集生成频繁项集的过程，即由 C_k 生成 L_k 的过程。遍历 C_k 计算每个项集的支持度，即遍历项集列表，查看 C_k 内的项集是否是其子集，如果是则将计数器加一，最后将计数值除以项集总数计算支持度，将不低于最小支持度的支持度对应的项集加入 L_k ，将<item, support>加入字典。

模块化频繁项集生成下一阶候选频繁项集的过程，即由 L_k 生成 C_{k+1} 的过程。对 L_k 中的项集两两一组取并集后计算元素个数，将满足要求的项集加入 C_k ，这里引入连接策略和剪枝策略进行筛选，减少进入下一阶段的项集数量，从而加速 L_{k+1} 的生成。

(3) 生成关联规则

得到 L_1 、 L_2 和 L_3 后，利用置信度公式计算关联规则的置信度，将不低于最小置信度的置信度对应的关联规则加入关联规则列表，作为最终结果。

3.2.2 遇到的问题及解决方式

(1) 选择合适的数据结构存放 C_k 和 L_k

若使用列表结构存放，在判断子集的过程会比较麻烦，最终选择了集合存放。在作为字典键时，使用 frozenset 转换为不可变集合，这样可以提高程序的运行速度。

(2) 由 L_k 生成 C_{k+1} 的速度较慢

在 L_k 生成 C_{k+1} 的过程中，仅采用连接策略的算法运行速度较慢，如图 1.2 所示。

```
/home/Asuna/anaconda3/bin/python /home/Asuna/Documents/VSCode Files/BDA2022/LAB3-Apriori/src/apriori.py
generate_c1 cost time: 0.010 s
generate_lk cost time: 0.269 s
generate_next_ck cost time: 0.006 s
generate_lk cost time: 11.959 s
generate_next_ck cost time: 0.081 s
generate_lk cost time: 20.973 s
generate_rules cost time: 0.001 s
L1.csv done.
L2.csv done.
L3.csv done.
1阶频繁项集个数为: 120
2阶频繁项集个数为: 605
3阶频繁项集个数为: 264
关联规则个数为: 98
用时: 33.57760572433472 s
```

图 1.2 改进前程序运行速度截图

为此引入剪枝策略。由于任何非频繁的 k 项集都不是频繁 $k+1$ 项集的子集，取其逆否命题就是频繁项集的子集也是频繁项集。根据这一点，在生成

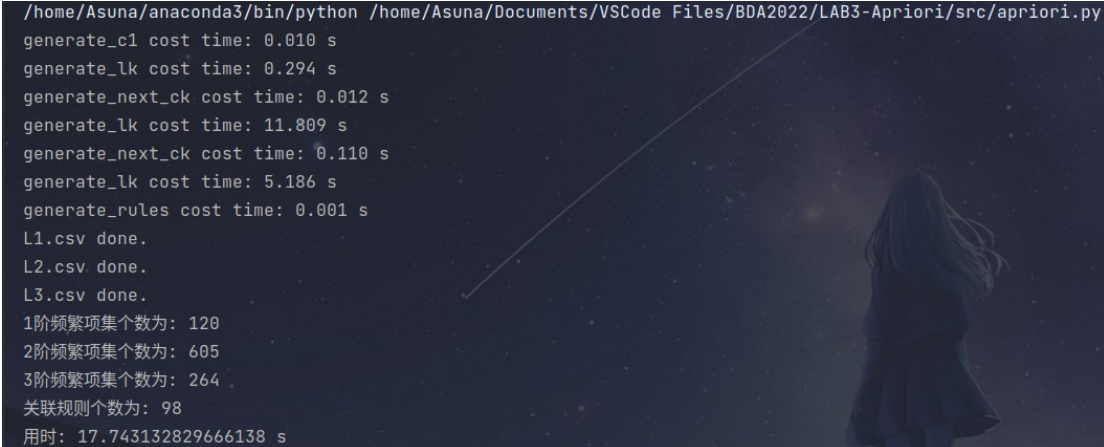
C_{k+1} 时, 如果存在 k 项集不属于 L_k , 可以无需将这一项加入下一阶的候选。判断子函数代码如图 1.3 所示。



```
1 def is_k_sub(k_item_set: frozenset, lk: set) -> bool:
2     """判断 k 项集是否是 k + 1 项子集
3
4     Args:
5         k_item_set (frozenset): k 项集
6         lk (set): k 阶频繁项集
7
8     Returns:
9         bool: k 项集是否是 k + 1 项子集
10    """
11    for item in k_item_set:
12        sub_item = k_item_set - frozenset([item])
13        if sub_item not in lk:
14            return False
15    return True
```

图 1.3 判断函数 is_k_sub 代码

引入剪枝策略后的算法运行速度和结果如图 1.4 所示。



```
/home/Asuna/anaconda3/bin/python /home/Asuna/Documents/VSCode Files/BDA2022/LAB3-Apriori/src/apriori.py
generate_c1 cost time: 0.010 s
generate_lk cost time: 0.294 s
generate_next_ck cost time: 0.012 s
generate_lk cost time: 11.809 s
generate_next_ck cost time: 0.110 s
generate_lk cost time: 5.186 s
generate_rules cost time: 0.001 s
L1.csv done.
L2.csv done.
L3.csv done.
1阶频繁项集个数为: 120
2阶频繁项集个数为: 605
3阶频繁项集个数为: 264
关联规则个数为: 98
用时: 17.743132829666138 s
```

图 1.4 改进后程序运行速度截图

可以看出, 这与剪枝前有了较大的提升, 尤其是在生成 L_3 这一阶段, 由原来的 20 秒左右降到了 5 秒左右。

3.2.3 实验测试与结果分析

运行 apriori.py 文件, 命令行输出如图 1.4 所示。

L_1 中部分频繁项集如图 1.5 所示, 共 120 项。

	L1.csv
1	frequent-itemSets, support
2	{'semi-finished bread'}, 0.017691916624300967
3	{'citrus fruit'}, 0.08276563294356888
4	{'margarine'}, 0.05856634468734113
5	{'tropical fruit'}, 0.10493136756481952
6	{'yogurt'}, 0.13950177935943062
7	{'coffee'}, 0.05805795627859685
8	{'whole milk'}, 0.25551601423487547
9	{'cream cheese '}, 0.03965429588205389
10	{'pip fruit'}, 0.07564819522114896

图 1.5 L₁中部分频繁项集

L₂中部分频繁项集如图 1.6 所示，共 605 项。

	L2.csv
1	frequent-itemSets, support
2	{'citrus fruit', 'margarine'}, 0.007930859176410779
3	{'yogurt', 'tropical fruit'}, 0.029283172343670564
4	{'yogurt', 'coffee'}, 0.009761057447890189
5	{'coffee', 'tropical fruit'}, 0.0071174377224199285
6	{'pip fruit', 'yogurt'}, 0.017996949669547534
7	{'pip fruit', 'cream cheese '}, 0.006100660904931368
8	{'yogurt', 'cream cheese '}, 0.012404677173360447
9	{'long life bakery product', 'other vegetables'}, 0.010676156583629894
10	{'long life bakery product', 'whole milk'}, 0.013523131672597865

图 1.6 L₂中部分频繁项集

L₃中部分频繁项集如图 1.7 所示，共 264 项。

	L3.csv
1	frequent-itemSets, support
2	{'long life bakery product', 'whole milk', 'other vegetables'}, 0.0056939501779359435
3	{'whole milk', 'yogurt', 'butter'}, 0.009354346720894764
4	{'bottled water', 'other vegetables', 'tropical fruit'}, 0.0062023385866802234
5	{'curd', 'yogurt', 'tropical fruit'}, 0.0052872394509405184
6	{'curd', 'whole milk', 'yogurt'}, 0.010066090493136757
7	{'citrus fruit', 'whole milk', 'yogurt'}, 0.010269445856634469
8	{'whole milk', 'butter', 'bottled water'}, 0.005388917132689374
9	{'citrus fruit', 'whole milk', 'bottled water'}, 0.005897305541433655
10	{'whole milk', 'butter', 'tropical fruit'}, 0.0062023385866802234

图 1.7 L₃中部分频繁项集

rule 中的关联规则如图 1.8 所示，总数为 98 项。

	rule
1	{'whipped/sour cream', 'domestic eggs'} => {'whole milk'}, 0.5714285714285715
2	{'root vegetables', 'fruit/vegetable juice'} => {'other vegetables'}, 0.5508474576271186
3	{'rolls/buns', 'root vegetables'} => {'whole milk'}, 0.5230125523012552
4	{'curd', 'tropical fruit'} => {'yogurt'}, 0.5148514851485149
5	{'whipped/sour cream', 'tropical fruit'} => {'other vegetables'}, 0.5661764705882353
6	{'onions', 'root vegetables'} => {'other vegetables'}, 0.6021505376344086
7	{'root vegetables', 'margarine'} => {'other vegetables'}, 0.5321100917431192
8	{'domestic eggs', 'root vegetables'} => {'whole milk'}, 0.5957446808510639
9	{'domestic eggs', 'tropical fruit'} => {'whole milk'}, 0.6071428571428571
10	{'beef', 'yogurt'} => {'whole milk'}, 0.5217391304347826

图 1.8 rule 中部分关联规则

3.3 实验总结

本次实验实现了关系规则挖掘中的 Apriori 算法，并利用该算法生成了给定数据集的关联规则。关联规则学习是一种在大型数据库中发现变量之间有趣性关系的方法，它的目的是利用一些量度来识别数据库中发现的强规则。例如，从超市销售数据中发现的规则可以作为做出促销定价或产品植入等营销活动决定的根据。关联规则如今还被用在许多应用领域中，包括网络用法挖掘、入侵检测、连续生产及生物信息学中。与序列挖掘相比，关联规则学习通常不考虑在事务中、或事务间的项目的顺序。除了本实验中使用的 Apriori 算法外，常用的算法还有 Eclat、FP-growth、ASSOC、OPUS search 等。

在本实验中，充分利用剪枝策略对运算过程进行优化也让我颇有感触。