

**《数据挖掘》实验报告**

|  |  |
| --- | --- |
| **专 业：** | **人工智能** |
| **学 号：** |  |
| **姓 名：** |  |

**2024-2025-1学期**

**南京邮电大学**

**实验2 基于****Apriori 算法的商品销售关联分析**

一、实验目的：

1. 熟悉Apriori算法等常用的数据挖掘关联规则算法；

2. 掌握matlab编程，实现Apriori算法对商品销售的分析。

二、实验设备

1. 硬件：微型计算机。

2. 软件：操作系统Windows XP以上版本、Matlab2012及以上版本。

三、实验内容与步骤

1. 配置matlab编程环境。

2. 给定事务数据库menu\_orders，根据给定的商品a、b、c、d、e之间的关系，找出存在于事务数据集中最大的频繁项集，利用得到的最大频繁项集与预先设定的最小置信度阈值生成强关联规则（**最小支持度计数为2，最小置信度为50%**）。

3. 由频繁项集产生强关联规则，并将结果输出进行分析。

4. 运行完程序后，生成rules.txt，里面的内容包括关联规则、支持度和置信度。

四、实验要求

完成基本的Apriori算法编程实验。掌握基本的数据挖掘算法原理，理解数据挖掘算法的实现过程。

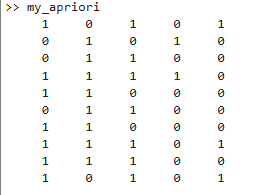
五、实验步骤、原理以及实验结果分析**（共70’；其中实验步骤30’，需给出阶段性运行结果或者系统部署界面截图；原理介绍20’；实验结果截图与分析20’）**

实验步骤：

1.数据读取与处理

从一个包含事务数据的文本文件中读取内容，并将其转换为事务矩阵。

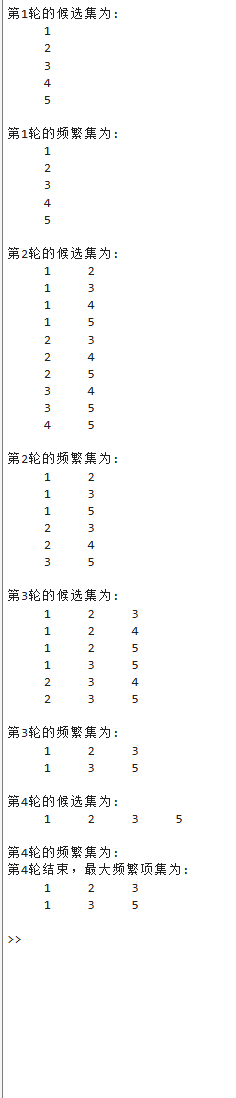
结果如下：



2.生成频繁项集：

实现了Apriori算法，用于挖掘频繁项集。通过逐轮生成候选项集并计算支持度，筛选出满足最小支持度要求的频繁项集，直到无法生成新的频繁项集为止。每轮的候选集和频繁项集都会输出，最终展示最大频繁项集。

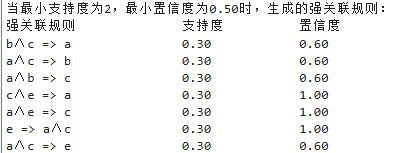
结果如下：



3.关联规则生成：

遍历每个频繁项集，生成其所有可能的子集，并计算每个规则的支持度和置信度。如果置信度满足最小置信度要求（min\_con），则将规则记录下来。最后，输出符合条件的强关联规则，并显示其支持度和置信度。

结果如下：



4.保存结果

将关联规则、支持度和置信度保存到rules.txt。

结果如下：



实验原理：

Apriori算法是一种经典的用于**频繁项集挖掘**和**关联规则学习**的算法，广泛应用于市场篮子分析等领域。其基本原理如下：

**候选集生成与筛选**：

Apriori算法基于**逐层搜索**的思想，首先从单个商品开始，逐步扩展到多商品的组合。

在每一轮中，算法通过**组合前一轮的频繁项集**生成新的候选项集，并通过计算支持度筛选出频繁项集。

**支持度计算**：

支持度是项集在所有事务中出现的频率，用于衡量项集的重要性。只有支持度大于或等于最小支持度（min\_sup）的项集才被认为是频繁项集。

**剪枝策略（Apriori原则）**：

Apriori算法的核心是**剪枝策略**，即：如果一个项集不是频繁项集，那么它的任何超集也不可能是频繁项集。

这一原则帮助减少了候选项集的数量，提高了算法效率。

**生成关联规则**：

在频繁项集挖掘完成后，Apriori算法进一步通过计算**置信度**（即规则的可靠性）生成关联规则。只有置信度大于或等于最小置信度（min\_con）的规则才会被保留。

六、思考与总结**（20’，每题10’）**

1. Apriori算法的原理是什么，有什么优缺点？

Apriori算法是一种用于挖掘频繁项集的经典算法，主要用于关联规则学习。其核心思想是通过迭代找出数据集中频繁出现的项集，并基于这些项集生成关联规则。

原理

1）. 频繁项集：项集的出现频率不低于设定的最小支持度阈值。

2）. Apriori性质：如果一个项集是频繁的，那么它的所有子集也必须是频繁的。

3）. 算法步骤：

- 生成候选项集：通过连接和剪枝操作生成候选项集。

- 计算支持度：扫描数据集，计算每个候选项集的支持度。

- 筛选频繁项集：保留支持度不低于阈值的项集。

- 迭代：重复上述步骤，直到无法生成新的频繁项集。

优点

1）. 简单易懂：算法逻辑清晰，易于实现。

2）. 广泛应用：适用于多种关联规则挖掘场景。

缺点

1）. 计算量大：需要多次扫描数据集，候选项集数量大时效率低。

2）. 内存消耗高：存储大量候选项集和频繁项集，内存需求大。

3）. 对阈值敏感：支持度和置信度阈值的选择对结果影响显著。

2. 使用剪枝目的是什么？

在Apriori算法中，剪枝的目的是减少候选项集的数量，从而提升算法的效率。具体来说，剪枝通过提前排除不可能成为频繁项集的候选项集，避免不必要的计算和存储开销。

剪枝的作用

减少计算量：

Apriori算法需要多次扫描数据集以计算候选项集的支持度，剪枝能有效减少候选项集的数量，从而降低计算复杂度。

节省内存：

剪枝减少了需要存储的候选项集数量，降低了内存占用。

提升效率：

通过剪枝，算法能更快地找到频繁项集，尤其在处理大规模数据集时效果显著。

七、实验过程遇到的问题以及心得体会**（10’）**

问题：

支持度和置信度阈值选择问题：

选择合适的最小支持度和最小置信度阈值是实验成功的关键。如果阈值过高，可能会导致没有足够的规则产生；如果阈值过低，可能会产生过多无用的规则。

解决办法：进行多次实验，逐步调整阈值，找到适合数据集的最佳支持度和置信度。

计算性能问题：

Apriori算法的时间复杂度较高，尤其在处理大型数据集时，频繁项集生成和候选集筛选的过程可能会变得非常慢。随着数据集增大，算法的计算时间会迅速增加。

解决办法：对于大规模数据集，可以使用改进的算法或并行计算技术来加速处理。比如，使用分布式计算框架（如Spark）进行数据处理，或者利用内存优化技术减少频繁项集生成的计算量。

心得体会：

通过这个实验，我深入理解了Apriori算法的原理和应用，体会到数据预处理和算法优化对结果的重要性。在实验过程中，调整最小支持度和置信度阈值，生成有意义的关联规则，并结合业务背景进行分析，能够提取有价值的销售洞察。尽管Apriori算法计算复杂，但通过合理的数据处理和优化策略，能够提高效率并取得较好的实验结果。总之，这次实验加深了我对数据挖掘技术的理解，并为实际应用打下了基础。

代码：

function apriori(inputfile, splitter, min\_sup, min\_con)

% 读取和预处理数据

[transactions, code] = trans2matrix(inputfile, splitter);

disp(transactions);

% 初始化变量

[n, m] = size(transactions);

for i = 1:n

x{i} = find(transactions(i, :) == 1); % 获取每一行购买商品的编号

end

k = 0;

L = {}; % 用于存储每轮的频繁项集

% 迭代生成频繁项集

while true

k = k + 1;

L{k} = {};

% 生成候选集C{k}

if k == 1

C{k} = (1:m)'; % 第一次候选集是所有商品

else

[nL, ~] = size(L{k-1});

cnt = 0;

for i = 1:nL

for j = i+1:nL

tmp = union(L{k-1}(i,:), L{k-1}(j,:)); % 两集合并集

if length(tmp) == k

cnt = cnt + 1;

C{k}(cnt, 1:k) = tmp;

end

end

end

C{k} = unique(C{k}, 'rows'); % 去掉重复的行

end

% 计算候选集的支持度

[nC, ~] = size(C{k});

C\_sup{k} = zeros(nC, 1);

for i = 1:nC

cnt = 0;

for j = 1:n

if all(ismember(C{k}(i,:), x{j}), 2)

cnt = cnt + 1;

end

end

C\_sup{k}(i) = cnt; % 存储每个候选集的支持度

end

% 求频繁项集L{k}

L{k} = C{k}(C\_sup{k} >= min\_sup, :);

if isempty(L{k})

break;

end

end

% 输出频繁项集

for i = 1:k

fprintf("第%d轮的候选集为:\n", i); disp(C{i});

fprintf("第%d轮的频繁集为:\n", i); disp(L{i});

end

fprintf("第%d轮结束，最大频繁项集为:\n", k); disp(L{k-1});

% 生成关联规则

rule\_count = 0;

[nL, mL] = size(L{k-1});

rule = [];

for p = 1:nL

L\_last = L{k-1}(p, :); % 当前频繁集

cnt\_ab = sum(cellfun(@(transaction) all(ismember(L\_last, transaction)), x));

len = floor(length(L\_last) / 2);

% 生成关联规则

for i = 1:len

s = nchoosek(L\_last, i); % 选i个数的所有组合

[ns, ~] = size(s);

for j = 1:ns

a = s(j, :);

b = setdiff(L\_last, a);

[na, ~] = size(a);

[nb, ~] = size(b);

% 计算关联规则a->b

cnt\_a = sum(cellfun(@(transaction) all(ismember(a, transaction)), x));

pab = cnt\_ab / cnt\_a;

if pab >= min\_con

rule\_count = rule\_count + 1;

rule(rule\_count, 1:na) = a;

rule(rule\_count, na+1:na+nb) = b;

rule(rule\_count, end+1) = pab; % 置信度

rule(rule\_count, end+1) = cnt\_ab / n; % 支持度

end

% 计算关联规则b->a

cnt\_b = sum(cellfun(@(transaction) all(ismember(b, transaction)), x));

pba = cnt\_ab / cnt\_b;

if pba >= min\_con

rule\_count = rule\_count + 1;

rule(rule\_count, 1:nb) = b;

rule(rule\_count, nb+1:nb+na) = a;

rule(rule\_count, end+1) = pba; % 置信度

rule(rule\_count, end+1) = cnt\_ab / n; % 支持度

end

end

end

end

% 输出关联规则

fprintf("当最小支持度为%d，最小置信度为%.2f时，生成的强关联规则：\n", min\_sup, min\_con);

fprintf("强关联规则\t\t支持度\t\t置信度\n");

for i = 1:rule\_count

pos = find(rule(i, :) == 0, 1) - 1; % 断开位置

for j = 1:pos

if j == pos

fprintf("%s", code{rule(i, j)});

else

fprintf("%s∧", code{rule(i, j)});

end

end

fprintf(" => ");

for j = pos+1:size(rule, 2)-2

if j == size(rule, 2)-2

fprintf("%s", code{rule(i, j)});

else

fprintf("%s∧", code{rule(i, j)});

end

end

fprintf("\t\t%.2f\t\t%.2f\n", rule(i, end-1), rule(i, end));

end

% 保存到文件

fileID = fopen('rules.txt', 'w');

fprintf(fileID, "当最小支持度为%d，最小置信度为%.2f时，生成的强关联规则：\n", min\_sup, min\_con);

fprintf(fileID, "强关联规则\t\t支持度\t\t置信度\n");

for i = 1:rule\_count

pos = find(rule(i, :) == 0, 1) - 1;

for j = 1:pos

if j == pos

fprintf(fileID, "%s", code{rule(i, j)});

else

fprintf(fileID, "%s∧", code{rule(i, j)});

end

end

fprintf(fileID, " => ");

for j = pos+1:size(rule, 2)-2

if j == size(rule, 2)-2

fprintf(fileID, "%s", code{rule(i, j)});

else

fprintf(fileID, "%s∧", code{rule(i, j)});

end

end

fprintf(fileID, "\t\t%.2f\t\t%.2f\n", rule(i, end-1), rule(i, end));

end

fclose(fileID);

fprintf('关联规则已保存到 rules.txt 文件中。\n');

end

function [transactions, code] = trans2matrix(inputfile, splitter)

% 读取输入数据并转换为事务矩阵

transactions = [];

code = {};

fid = fopen(inputfile);

if fid == -1

error('Cannot open input file.');

end

lines = {};

tline = fgetl(fid);

while ischar(tline)

lines{end+1} = tline;

tline = fgetl(fid);

end

fclose(fid);

all\_items = {};

for i = 1:length(lines)

items = regexp(lines{i}, splitter, 'split');

all\_items = [all\_items, items];

end

code = unique(all\_items);

num\_transactions = length(lines);

transactions = zeros(num\_transactions, length(code));

for i = 1:num\_transactions

items = regexp(lines{i}, splitter, 'split');

idx = ismember(code, items);

transactions(i, idx) = 1;

end

end