**电 子 科 技 大 学**

**实 验 报 告**

1. **实验名称：**

频繁模式与关联规则

1. **实验内容简介：**

利用C++语言构建FP-tree算法，利用条件模式树生成频繁模式，再通过给定的推理规则得出关联规则。

1. **实验器材（设备、元器件）**

设备信息：家用笔记本电脑

操作系统：Windows 10 Educational 21H1

处理器：AMD Ryzen 5 5600U with Radeon Graphics 2.30 GHz

软件平台：CLion 2022.2.3

1. **算法说明：**

1. 相关概念说明

**1.1. 项与项集:** 项可看作最小单位的研究对象，而项集则是项的集合，含有k个项的项集称为k-项集，包含所研究的所有项的项集为ItemSet。

**1.2. 事务与事务集：**一个事务是一个项集，且与唯一的标识符Tid相联系，不同的事务构成事务集D，在这里，事务集作为关联规则挖掘的原始数据库。

**1.3. 支持度与置信度：**支持度support（A->B）定义为P（A\cdotB），即项集（A,B）在总项集中出现的概率；而置信度confidence（A->B）=P(B|A)

**1.4. 频繁项集：**在项集中频繁出现，其支持度大于最小支持度minSup的项集。

**1.5. 关联规则：**关联规则是形如A->B的蕴含式，其中，A∩B为空集，且A和B都不为空集。定义强关联规则为其支持度大于最小支持度minCon的关联规则。

2. FP-tree 算法

在之前的项目中已经接触过挖掘频繁模式的另一种算法：Apriori算法，但其较高的扫描数据集频次使其无法作为高效的挖掘频繁模式的算法。FP-Tree基于频繁模式树（Frequent Pattern Tree）来挖掘频繁模式的算法FP-Growth，通过构建FP-tree，将原始数据经处理后置于树上，在通过构建条件模式树来挖掘频繁模式。相比Apriori算法，其只需要扫描两遍原始事务集，其性能较Apriori算法更好。FP-tree算法的具体细节见下节。

**五、算法分析与设计**

1. 建立FP-tree树

FP-tree 树的建立需要两种数据结构：项头表headTable，和具体的FP-tree树。在建立FP-tree树前，需要对数据进行预处理。

预处理时便可以建立项头表，扫描原始事务集对所有出现的项进行计数，并通过计数结果舍去比例小于最小支持度的项，按计数结果从大到小重新对事务集中事物的项进行排序，并同样舍去比例小于最小支持度的项。这样就可以得到已经完成的项头表头部，和预处理后的新事务集。

预处理后FP-tree树的建立过程类似于字典树的建立过程，每个事务映射进树中的一个路径，一个项对应一个节点，并对每个节点进行计数，新增加的事务最大限度地复用原来的路径，并更新每个节点的计数值。同时，对于每个新增的节点，项头表的对应项的节点就会在其对应的链表末尾新增到该节点的链接。按此操作对每一个事务进行映射，最后建立起完整的FP-tree树以及项头表。

从上述过程中可以看到，对原数据集的扫描只要预处理和建立FP-tree这两次过程，相比Apriori算法对原始事务集的频繁扫描而言更加高效。

2. 频繁项集挖掘

FP-Tree算法的第二步便是从项头表的底部依次向上挖掘，而对于每个项头表对应的FP-tree中的每一个节点，需要找到其条件模式基。

条件模式基是以要挖掘的节点为对应的FP子树，递归构建出FP子树后，将子树中每个节点的技术设置为叶子节点的计数，并且删除计数低于支持度的节点。通过不断构建条件模式基，便可以递归挖掘得到频繁项集，并将结果输出至frequentPatterns.txt中。

**3. 关联规则挖掘**

在频繁项集已由FP-Tree算法全部列出的情况下，关联规则的挖掘就较为容易了。设计GenerateRules函数，扫描频繁模式集合，若有频繁模式含有两个及以上的项，即可对其进行关联模式识别。

在关联规则识别中，可以用递归算法进行挖掘。对传入的频繁模式freItem，可以生成子频繁模式，若子频繁模式确实出现在频繁模式集合中，便可计算子频繁模式到两者差集的支持度，生成关联模式；若子频繁模式仍然含有两个及以上的项，则继续递归识别关联规则，并将最后的结果输出至confidenceRules.txt中。

**六、实验数据及结果：**

1. 数据集简介

本项目所用数据集为于某超市某一时间段的购物记录数据集，数据来源于：

<https://download.csdn.net/download/qq_41935823/22001900>

其中共有746行数据，197种项，每一事务最大项数为23.

2. 项目结果与分析

在代码具体实现中，最小支持度通过整数来指定比例，而最小置信度通过浮点数来指定具体比例，通过选取不同的置信度、支持度对原数据集进行挖掘，以挖掘的频繁项集、关联模式个数为输出，结果如下：

表1 不同最小支持度[1]下的频繁模式挖掘个数

|  |  |
| --- | --- |
| 最小支持度 | 频繁模式(个) |
| 50/764 | 4576 |
| 70/764 | 1641 |
| 100/764 | 509 |
| 150/764 | 111 |

[1]由于置信度与频繁模式挖掘相互独立，因此只列出支持度和频繁项集。

从表中可知，随着最小支持度的降低，挖掘的频繁模式个数增加。并且在选择的区间中，支持度的细微变化也会导致频繁模式个数的急剧变化，从中可以反映出选定数据集的相关性质。

表2 不同最小支持度、最小置信度的关联规则挖掘个数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 支持度/置信度 | 0.01 | 0.1 | 0.3 | 0.5 |
| 50 | 125604 | 125604 | 98226 | 57225 |
| 70 | 28668 | 28668 | 24288 | 15262 |
| 100 | 4904 | 4904 | 4546 | 3024 |
| 150 | 474 | 474 | 451 | 345 |

从表中以及前文所述可知，最小支持度的选择决定了频繁项集的个数，而频繁项集个数决定了能够挖掘的关联模式上限。

在最小支持度较大的情况下，其能够挖掘的关联规则个数较少，且随着置信度的降低，关联规则数量增长较为有限。而到了最小支持度较小的情况，其能从频繁模式中挖掘的关联规则大幅增加，并且随着最小置信度阈值的降低，关联规则数量也随之增长，最小支持度越低，其增长比例越高。

另外，随着置信度的降低，超过一定阈值的时候，关联规则数量不再增长。

现以最小支持度为70/746，最小置信度为1%为例，给出具体分析。

表3 部分关联规则A->B挖掘结果

|  |  |  |
| --- | --- | --- |
| A | B | 置信度 |
| 调味品 | 即食主食 散装休闲食品 饮料 | 0.467391 |
| 即食主食 | 常温熟食类 进口食品 | 0.578182 |
| 进口食品 休闲小食品 | 常温乳制品 饮料 | 0.490798 |
| 冷藏乳制品 饮料 | 调味品 即食主食 进口食品 | 0.382222 |
| 冷藏乳制品 饮料 | 调味品 即食主食 进口食品 散装休闲食品 | 0.342222 |
| 冷藏乳制品 饮料 | 调味品 即食主食 散装休闲食品 | 0.386667 |

表4 部分频繁项集挖掘结果

|  |  |
| --- | --- |
|  | 支持度 |
| 调味品 进口食品 冷藏乳制品 散装休闲食品 糖果巧克力 | 88/764 |
| 白酒 | 126/764 |
| 冰品 常温熟食类 调味品 | 70/764 |
| 饼干 冲饮食品 即食主食 进口食品 饮料 | 70/764 |
| 饼干 即食主食 饮料 | 141/764 |

可以看到，上述关联规则中，有多项->多项的关联规则；其对项集的大小没有限制；另外，可以看到多个关联规则的交集中，推导结果的并集有着更小的置信度。

表5 k项集相关的关联规则挖掘结果

|  |  |
| --- | --- |
|  | 关联模式（个） |
| 1 | 6251 |
| 2 | 9780 |
| 3 | 8044 |
| 4 | 3665 |
| 5 | 858 |
| 6 | 70 |
| 总共 | 28668 |

关联规则的k-项集统计中，随着k的增长，关联规则的个数呈先增长后下降的趋势。

**七、实验分析与探讨：**

在此之前需要指出的是，在本项目中，不对挖掘出的关联规则含义、取舍进行进一步的描述。

可以看到，最小支持度影响FP-Tree算法能够挖掘的频繁模式个数，而频繁模式作为挖掘关联模式的基础，其对于关联模式数量可以说是呈指数型影响，关联规则的个数随着频繁模式的个数增加而呈指数型增长。

由于挖掘的关联规则的置信度有着上下限，因此在实际的挖掘过程中，不仅需要考虑实际需要来指定最小置信度，还需要考虑能够使最小置信度发挥作用的可行区间。

另外，相比于Apriori算法，FP-Tree在实际运算过程中无疑是高效的。在进行Apriori算法的过程中，算法所花费的时间是对算法分析的重要一步；然而在FP-Tree算法中，FP-Tree的频繁模式挖掘所花费的时间可以忽略不计，其主要的时间花费其实是在递归识别关联规则的过程中，例如选定最小支持度50/764,最小置信度0.1时，识别关联模式的过程花费了一整个下午。因此，在上述的分析过程中未对FP-Tree算法的时间进行分析讨论。

**八、改进建议**

在实际的算法实现中，可以对存储数据的数据结构进行进一步优化。由于采用的数据结构没有很好的组织，代码的编写过程中受到了较大的限制，关联模式的识别过程较为低效也是因为这个原因。

另外，可以采用多种数据集对项目的FP-Tree算法进行测试，降低单数据集下的特征随机性。

**九、附录**

1. 源代码

1.1 FP-Tree头文件

#ifndef UNTITLED\_FPTREE\_H

#define UNTITLED\_FPTREE\_H

#include "map"

#include "string"

#include "vector"

#include "iostream"

#include "set"

#include "utility"

#include "memory"

struct FPNode {

const std::string item;

unsigned long long cnt;

std::shared\_ptr<FPNode> link;

std::weak\_ptr<FPNode> parent;

std::vector<std::shared\_ptr<FPNode>> children;

FPNode(std::string item, const std::shared\_ptr<FPNode>& parent);

};

class FPTree {

private:

void ReadData(const std::string& path);

std::map<std::string,unsigned long long> itemCnt;

std::vector<std::vector<std::string>> originData;

unsigned long long minSup;

double minCon;

std::shared\_ptr<FPNode> root;

std::map<std::string,std::shared\_ptr<FPNode>> headTable;

bool isSinglePath(const std::shared\_ptr<FPNode>& nowNode);

bool isSinglePath(const FPTree& nowTree);

std::set<std::pair<std::set<std::string>, unsigned long long int>> FP\_Growth(const FPTree& nowTree);

void getRules(const std::pair<std::set<std::string>,unsigned long long>& freItem,

const std::pair<std::set<std::string>,unsigned long long>& nowFreItem);

public:

std::set<std::pair<std::set<std::string>,unsigned long long>> frequentSet;

std::map<std::pair<std::set<std::string>,std::set<std::string>>,double>

rulesWithConfidence;

FPTree(const std::vector<std::vector<std::string>>& nowTransactions,

const unsigned long long minSup,double minCon);

bool empty() const;

void GenerateFrequentRules();

void GenerateRules();

};

#endif //UNTITLED\_FPTREE\_H

2.2 FP\_Tree源文件

#include "FPtree.h"

#include <utility>

#include "sstream"

#include "algorithm"

using namespace std;

FPNode::FPNode::FPNode(std::string item, const std::shared\_ptr<FPNode> &parent):

item(std::move(item)),cnt(1),link(nullptr),parent(parent),children(){

}

bool FPTree::empty() const {

if(root== nullptr){

cout<<"Error:root is null!";

exit(0);

}

return root->children.empty();

}

FPTree::FPTree(const std::vector<std::vector<std::string>>& nowTransactions,const unsigned long long min\_Sup,

double min\_Con):

root(std::make\_shared<FPNode>(std::string(), nullptr)),headTable(),minSup(min\_Sup),minCon(min\_Con),

originData(nowTransactions){

for(auto &i :originData){

for(auto &j:i){

++itemCnt[j];

}

}

for(auto i=itemCnt.begin(); i!=itemCnt.end();){

const unsigned long long temCnt=i->second;

if(temCnt<minSup){

itemCnt.erase(i++);

} else {

++i;

}

}

struct self\_compare{

bool operator()(const pair<string,unsigned long long>&l,const pair<string,unsigned long long>& r) const {

return tie(l.second,l.first)>tie(r.second,r.first);

}

};

set<pair<string,unsigned long long>,self\_compare> sortedItem(itemCnt.cbegin(),itemCnt.cend());

for(auto &event:originData){

auto nowRoot=root;

for(auto & nowPair:sortedItem){

auto & nowItem=nowPair.first;

if(find(event.begin(),event.end(),nowItem)!=event.end()){

auto nxtChild\_it=find\_if(

nowRoot->children.begin(),nowRoot->children.end(),[nowItem](const shared\_ptr<FPNode>& nowChild) {

return nowItem == nowChild->item;

});

//若未能在现节点下的root的child找到，新建节点。

if(nxtChild\_it == nowRoot->children.end()){

//维护新的信息。

auto newChild= make\_shared<FPNode>(nowItem,nowRoot);

nowRoot->children.push\_back(newChild);

//更新项头表信息

if(headTable.count(newChild->item)){

auto preNode=headTable[newChild->item];

while(preNode->link){

preNode=preNode->link;

}

preNode->link=newChild;

} else {

headTable[newChild->item]=newChild;

}

nowRoot=newChild;

} else {

auto nxtNode=\*nxtChild\_it;

++nxtNode->cnt;

nowRoot=nxtNode;

}

}

}

}

}

bool FPTree::isSinglePath(const FPTree &nowTree) {

return nowTree.empty()|| isSinglePath(nowTree.root);

}

bool FPTree::isSinglePath(const std::shared\_ptr<FPNode> &nowNode) {

if(nowNode->children.empty()){

return true;

}

if(nowNode->children.size()>1){

return false;

}

return isSinglePath(nowNode->children[0]);

}

std::set<std::pair<std::set<std::string>, unsigned long long int>> FPTree::FP\_Growth(const FPTree &nowTree) {

if(nowTree.empty()){

return {};

}

if(isSinglePath(nowTree)){

set<pair<set<string>,unsigned long long>> singlePatterns;

auto nowNode=nowTree.root->children.front();

while(nowNode){

auto &nowItem=nowNode->item;

auto nowCnt=nowNode->cnt;

pair<set<string>,unsigned long long> newPattern{{nowItem},nowCnt};

singlePatterns.insert(newPattern);

for(auto & nowPattern:singlePatterns){

auto newPattern\_2{nowPattern};

newPattern\_2.first.insert(nowItem);

newPattern\_2.second=nowCnt;

singlePatterns.insert(newPattern\_2);

}

if (nowNode->children.size() == 1)

nowNode = nowNode->children[0];

else

nowNode = nullptr;

}

return singlePatterns;

} else {

set<pair<set<string>,unsigned long long>> multiPatterns;

for(auto & p:nowTree.headTable){

auto & nowItem=p.first;

vector<pair<vector<string>,unsigned long long>> conditionalPattern\_Prefix;

auto baseNode=p.second;

while(baseNode){

auto nowCnt=baseNode->cnt;

auto nowNode=baseNode->parent.lock();

if(nowNode->parent.lock()){

pair<vector<string>,unsigned long long> nowPrefix{{},nowCnt};

while(nowNode->parent.lock()){

nowPrefix.first.push\_back(nowNode->item);

nowNode=nowNode->parent.lock();

}

conditionalPattern\_Prefix.push\_back(nowPrefix);

}

baseNode=baseNode->link;

}

vector<vector<string>> conditionalTransactions;

for(auto& nowPath:conditionalPattern\_Prefix){

auto& nowPathItems=nowPath.first;

auto nowPathCnt=nowPath.second;

for(unsigned long long i=0;i<nowPathCnt;++i){

conditionalTransactions.push\_back(nowPathItems);

}

}

const FPTree conditionalFPTree(conditionalTransactions,nowTree.minSup,minCon);

auto conditionalPatterns= FP\_Growth(conditionalFPTree);

set<pair<set<string>,unsigned long long>> nowPatterns;

unsigned long long nowCnt=0;

auto nowNode=p.second;

while(nowNode){

nowCnt+=nowNode->cnt;

nowNode=nowNode->link;

}

pair<set<string>,unsigned long long> temPattern{{nowItem},nowCnt};

nowPatterns.insert(temPattern);

for(auto& conditionalPattern:conditionalPatterns){

auto newConditionalPattern{conditionalPattern};

newConditionalPattern.first.insert(nowItem);

newConditionalPattern.second=conditionalPattern.second;

nowPatterns.insert({newConditionalPattern});

}

multiPatterns.insert(nowPatterns.cbegin(),nowPatterns.cend());

}

return multiPatterns;

}

}

void FPTree::GenerateFrequentRules() {

frequentSet=FP\_Growth(\*this);

}

std::set<std::string> subtractSet(const set<string>& a,const set<string>& b){

auto ans=a;

for(const auto& i:b){

if(ans.count(i)){

ans.erase(i);

}

}

return ans;

}

void FPTree::getRules(

const std::pair<std::set<std::string>,unsigned long long>& freItem,

const std::pair<std::set<std::string>, unsigned long long> &nowFreItem) {

for(auto& item:nowFreItem.first){

auto subSet=nowFreItem.first;

subSet.erase(item);

unsigned long long value= 0;

for(auto &j:frequentSet){

if(j.first==subSet){

value=j.second;

break;

}

}

double confidence=0;

if(value){

confidence= (double)freItem.second / (double )value;

}

if(confidence>=minCon){

bool flag=false;

auto temSet= subtractSet(freItem.first,subSet);

for(auto& rule:rulesWithConfidence){

if(rule.first.first==subSet&&rule.first.second== temSet){

flag=true;

break;

}

}

if(!flag){

rulesWithConfidence[{subSet,temSet}]=confidence;

}

if(subSet.size()>1){

getRules(freItem,{subSet,value});

}

}

}

}

void FPTree::GenerateRules() {

int cnt=0;

for(auto& i:frequentSet){

cout<<++cnt<<' '<<rulesWithConfidence.size()<<'\n';

if(i.first.size()>1){

getRules(i,i);

}

}

}

2.3 主程序（main.cpp）

#include <iostream>

#include <fstream>

#include <sstream>

#include "FPtree.h"

using namespace std;

const string path("dataset.txt");

std::vector<std::vector<std::string>> ReadData(const std::string &path) {

vector<vector<string>> originData;

ifstream fInput(path);

if(!fInput.is\_open()){

cout<<"Error:can't open the file!";

exit(0);

}

string nowLine;

while(getline(fInput,nowLine)){

stringstream sInput(nowLine);

string nowWord;

vector<string> nowEvent;

while(sInput>>nowWord){

nowEvent.push\_back(nowWord);

}

originData.push\_back(nowEvent);

}

return originData;

}

void Print\_Patterns(const set<pair<set<string>,unsigned long long>>& frequencyPatterns){

ofstream fOutput("frequencePatterns.txt");

for(auto&i:frequencyPatterns){

fOutput<<"[";

for(auto&j:i.first){

fOutput<<j<<" ";

}

fOutput<<i.second<<"]\n";

}

cout<<"Generate "<<frequencyPatterns.size()<<"patterns.\n";

fOutput<<"Generate "<<frequencyPatterns.size()<<"patterns.";

}

void Print\_Rules(const map<pair<set<string>,set<string>>,double>& rules){

ofstream fOutput("confidenceRules.txt");

for(auto& rule:rules){

fOutput<<"[";

for(auto &item:rule.first.first){

fOutput<<item<<" ";

}

fOutput<<"]-->[";

for(auto &item:rule.first.second){

fOutput<<item<<" ";

}

fOutput<<"]: "<<rule.second<<"\n";

}

cout<<"Generate "<<rules.size()<<" rules.\n";

fOutput<<"Generate "<<rules.size()<<" rules";

}

int main(){

unsigned long long minSup;

double minCon;

cin>>minSup>>minCon;

FPTree testTree(ReadData(

path)

,minSup,minCon);

cout<<"The construction of the FP-tree complete!\n";

testTree.GenerateFrequentRules();

Print\_Patterns(testTree.frequentSet);

testTree.GenerateRules();

Print\_Rules(testTree.rulesWithConfidence);

// cout<<"ALL OVER! THANKS FOR THE CLASS!";

}