Apirori和FP-growth算法性能比较

曾佳

广西大学计算机与电子信息学院

**摘 要**

关联规则最早由 Agrawal 等[1]提出,是用于从大量原始数据中挖掘令人感兴趣的规则的方法,是近年数据挖掘研究方向之一.Chen 等[2]给出一种改进的关联规则 Apriori算法,Apriori算法是关联规则挖掘算法和最经典的算法；FP-Growth是一种基于频繁模式树（FP-Tree）的频繁模式查找算法,本文主要是对这两种算法基于不同数据,不同支持度来进行比较,从而对这两种算法性能进行全面的评估.

**关键词**:数据挖掘；关联分析；Apriori；FP-Growth；性能比较

Comparison of performance between Apirori and fp-growth algorithms

Jia zeng

School of Computer and Electronic Information, Guangxi University

**Abstract**

Association rules, first proposed by Agrawal et al,[1] are a method used to mine interesting rules from a large number of original data, and are one of the research directions of data mining in recent years.Chen et al. [2] proposed an improved Apriori algorithm for association rules, which is the most classic algorithm for association rule mining.FP-Growth is a frequent pattern search algorithm based on FP-Tree. This paper mainly compares the two algorithms based on different data and different support degrees, so as to comprehensively evaluate the performance of the two algorithms.

**Key words:** Data mining ；association analysis ；Apriori； FP-Growth； performance comparison

目录

[1 关联规则 4](#_Toc70194074)

[1.1 支持度 4](#_Toc70194075)

[1.2 置信度 4](#_Toc70194076)

[1.3 频繁项集和强关联规则 4](#_Toc70194077)

[2 算法介绍和流程 5](#_Toc70194078)

[2.1 Apriori 算法介绍 5](#_Toc70194079)

[2.1.1 算法思想 5](#_Toc70194080)

[2.1.2 算法功能模块介绍 5](#_Toc70194081)

[2.1.3 Apriori流程图 6](#_Toc70194082)

[2.2 FP-Growth 算法介绍 7](#_Toc70194083)

[2.2.1 算法思想 8](#_Toc70194084)

[2.2.2 算法模块功能介绍 8](#_Toc70194085)

[2.2.3 FP-growth流程图 9](#_Toc70194086)

[3 Apriori和FP-growth算法性能比较 9](#_Toc70194087)

[3.1 数据集选择 9](#_Toc70194088)

[3.2 比较方法 9](#_Toc70194089)

[3.3 性能比较分析 10](#_Toc70194090)

[3.3.1 自动生成数据T10I4D100K 10](#_Toc70194091)

[3.3.2 实际数据retail 11](#_Toc70194092)

[3.3.3 稠密数据chess 12](#_Toc70194093)

[3.3.4 综合比较分析 13](#_Toc70194094)

[4 总结 14](#_Toc70194095)

[参考文献 16](#_Toc70194096)

1. 关联规则

关联规则挖掘主要用于发现存在于数据库中的项或属性间的关联关系．设I={I1, I2,…Im}是项的集合,其中ｍ表示项的数目．对于项集Ａ,若Ａ中含有的项数为(k≤m),则称A为k项集.Ｄ为数据库事务的集合,用Ｄ表示事务集中事务的个数．对于每个事务Ｔ有Ｔ={t1, t2,…tm},ti∈I,Ｔ≠Ø.关联规则是形如ＡＢ 的蕴涵式,其中Ａ∈Ｉ,Ｂ∈Ｉ,Ａ≠Ø,Ｂ≠Ø,并且Ａ∩Ｂ＝Ø．表示事务Ｔ在含有项集Ａ的条件下,同时含有Ｂ的概率[3].用户关心的关联规则,可以用两个标准来衡量:支持度和可信度

* 1. 支持度

支持度的意义在于度量项集在整个事务集中的重要性.我们在发现规则时,总希望找到高概率出现的项集．单一项集的支持度表示该项集在事务集中出现的概率．即,

规则AB的支持度,表示项集｛A,B｝在事务集中同时出现的概率．即,

最小支持度是项集的最小支持阈值,记为min\_sup,代表了用户关心的关联规则的最低重要性[4] ．

* 1. 置信度

置信度通常用来衡量规则的可信程度,表示在关联规则的先决条件Ａ发生的条件下,关联结果Ｂ发生的概率.计算公式如下:

最小置信度是置信度的最低阈值,记为min\_conf,代表关联规则的最低可信度[4]

* 1. 频繁项集和强关联规则

支持度不小于min\_sup的项集称为频繁项集．如果规则 满足)≥且,则称为强关联规则.min\_sup和min\_conf通常由用户根据先验知识来设定.

1. 算法介绍和流程
   1. Apriori 算法介绍

Apriori 算法是关联规则挖掘算法和最经典的算法.它使用逐层搜索的迭代方法来查找数据库中各项之间的关系以形成规则.该过程包括连接（类似于矩阵的操作）和修剪（删除不必要的中间结果）.该算法中项目集的概念是项目的集合 .包含 K 个项的集合为k项 集.项目集出现的频率是包含该项目集的事务数,称为项目集的频率.如果某个项目集满足最小支持度,则称为频繁项目集.Apriori 算法是用于挖掘布尔关联规则的最有影响力的频繁项集算法之一,由Rakesh Agrawal 和 Ramakrishnan Skrikant 提出.它使用称为逐层搜索的迭代方法,其中 k 个项目集用于探索（k+1）个项目集.首先,找到频繁的1个项目集.该集合表示为 L1.L1 用于查找频繁 2 项集 L2,L2用于查找 L3,依此类推,直到找不到 k 个项集.每次找到 Lk 时,都需要进行数据库扫描. 为了提高逐层生成频繁项集的效率,使用了称为 Apriori 的属性来压缩搜索空间.约束:一个是频繁项目集的 所有非空子集也必须是频繁的,另一个是不频繁项目集的所有父集都不频繁[5] .Apriori 算法已广泛用于商业和网络安全等各个领域.

* + 1. 算法思想

Apriori算法的目标是找到最大的K项频繁集，首先搜索出候选1项集及对应的支持度，剪枝去掉低于支持度的1项集，得到频繁1项集。然后对剩下的频繁1项集进行连接，得到候选的频繁2项集，筛选去掉低于支持度的候选频繁2项集，得到真正的频繁二项集，以此类推，迭代下去，直到无法找到频繁k+1项集为止，对应的频繁k项集的集合即为算法的输出结果。

* + 1. 算法功能模块介绍

本文的Apriori和FP-growth算法均源于github, 采用python编写，地址为：

<https://github.com/blackAndrechen/data_mine>

该Apriori算法一共有8个模块。

def load\_data(path)：

功能：用来读取数据，将文件数据转化为列表

参数：path:数据路径

说明：主要读取csv文件，由于老师给的文件都是txt和dat文件，在导入数据之前均已 转化为csv文件

def save\_rule(rule,path)：

功能：用来保存得到的所有的强关联规则。

参数：rule：保存所有强关联规则的列表

class Apriori():

def create\_c1(self,dataset)

功能：遍历整个数据集生成c1候选集（所有的1项集）

参数：dataset：保存所有事务的列表

def create\_ck(self,Lk\_1,size)

功能：通过频繁项集Lk-1创建ck候选项集

参数：LK\_1:候选k项集的前一个频繁项集，size：要生成频繁项集的长度

说明：相当于自连接操作

def has\_infrequent\_subset(self,Ck\_item, Lk\_1)

功能：检查候选项Ck\_item的子集是否都在Lk-1中

参数：CK\_item:候选项，LK\_1: 候选k项集的前一个频繁项集

说明：根据Apriori算法原理，一个频繁项集的子集必须全部都是频繁项集

def generate\_lk\_by\_ck(self,data\_set,ck,min\_support,support\_data)

功能：通过候选项ck生成lk，并将各频繁项的支持度保存到support\_data字典中

参数：data\_set: 保存所有事务的列表,ck:第k项候选集，min\_support:最小支持度，suppor\_data:保存频繁项集的字典

说明：相当于剪枝操作

def generate\_L(self,data\_set, min\_support)

功能：用于生成所有频繁项集的主函数

参数：data\_set: 保存所有事务的列表，min\_support:最小支持度

def generate\_R(self,dataset, min\_support, min\_conf)

功能：用于生成所有的强关联规则

参数：data\_set: 保存所有事务的列表，min\_support:最小支持度，min\_conf:最小置信度

* + 1. Apriori流程图

图示

描述已自动生成

图1 Aprioti算法流程图

* 1. FP-Growth 算法介绍

FP-Growth 基于频繁模式树（FP-Tree）FP-Growth 的频繁模式查找算法.这个想法是构造一个FP-Tree,将 数据集中的数据映射到树,然后根据该 FP-Tree 找到所有频繁项集.FP-Growth 算法意味着通过扫描两次交易数据集,每个交易中包含的频繁项目将按其支持的降序压缩并存储在FP-Tree 中.在将来发现频繁模式的过程中,无 需扫描交易数据集,而只需在 FP-Tree 中进行搜索.递归调用 FP-Growth 的方法可以直接生成频繁模式,因此不需要在整个发现过程中生成候选模式.由于数据集仅被扫描两次,FP-Growth 算法克服了 Apriori 算法中的问题,其执行效率也明显优于 Apriori算法[5].

* + 1. 算法思想

FP-Growth算法的主要思想是:将表示频繁项目集的数据库压缩并存储在频繁模式树中,并将每个交易数据中项目之间的关系保留在频繁模式树中.然后,根据条件模式基础将频繁模式树分为一组条件 FP 树,并分别挖掘这些条件 FP 树.

基本步骤：

（1）遍历数据集，计算每一项的支持度，并降序排序，与min\_sup相比过滤掉不频繁的项，得到频繁1项集L1;

（2）创建FP-tree，根节点记为T，创建频繁项表，再次遍历数据集，按照L1将事务中的频繁项筛选出来，并按照L1中的顺序进行排序，记为P

（3）将P插入到T中，如果T中存在P的前缀则前缀节点的计数值加1 ，仅为跟在前缀之后的项创建新节点并计数加1。

（4）更新频繁项表中对应的链接

* + 1. 算法模块功能介绍

def load\_data(path)：

功能：用来读取数据，将文件数据转化为列表

参数：path:数据路径

def save\_rule(rule,path)：

功能：用来保存得到的所有的强关联规则。

参数：rule：保存所有强关联规则的列表

class Fp\_growth():

def update\_header(self,node, targetNode)

功能：更新headertable中的node节点形成的链表

参数：node:用于存放频繁项元素的节点，targetNode:目标节点

def update\_fptree(self,items, node, headerTable)

功能：用于更新fp-tree

参数：node: 用于存放频繁项元素的节点,headerTable:项头表，存放1频繁项集的所有节点并降序排列

def create\_fptree(self,data\_set, min\_support,flag=False)

功能：创建fptree

参数：data\_set:存放数据集的列表，min\_support:最小支持度

def find\_path(self,node, nodepath):

功能：递归将node的父节点添加到路径

def find\_cond\_pattern\_base(self,node\_name, headerTable)

功能：根据节点名字，找出所有条件模式基

参数：node\_name:节点名字，headerTable:项头表

说明：所谓条件模式基是以我们要挖掘的节点作为叶子节点所对应的FP子树。得到这个FP子树，我们将子树中每个节点的的计数设置为叶子节点的计数，并删除计数低于支持度的节点

def create\_cond\_fptree(self,headerTable, min\_support, temp, freq\_items,support\_data)

功能：创建条件模式基树

参数：headerTable:项头表，min\_support:最小支持度，freq\_items:频繁项,support\_data: 保存频繁项集的字典

def generate\_L(self,data\_set, min\_support)

功能：用于生成所有频繁项集的主函数

参数：data\_set: 保存所有事务的列表，min\_support:最小支持度

def generate\_R(self,dataset, min\_support, min\_conf)

功能：用于生成所有的强关联规则

参数：data\_set: 保存所有事务的列表，min\_support:最小支持度，min\_conf:最小置信度

* + 1. FP-growth流程图

图示

描述已自动生成

图2 FP-growth流程图

1. Apriori和FP-growth算法性能比较
   1. 数据集选择

本次性能比较选择了三种数据集的三个数据，分别是实际数据retail，一共有88162个事务项；自动生成数据T10I4D100K，一共有100000条数据；稠密数据chess，一共有3196条数据。

* 1. 比较方法

对于实际数据retail和自动生成数据T10I4D100K，分别用了12个不同的支持度，分别是0.01,0.03,0.05,0.07,0.1,0.2,0.3,0.4,0.5,0.7,0.9,1，本来只准备选择0.1到1的阈值，但是运行这些数据后发现0.1的支持度只能发现很少的频繁项集和强关联规则，因此加了0.01到0.07几个支持度。置信度全部设置为0.7

对于稠密数据chess，由于当支持度为0.1-0.5的时候计算时间太长，所以只选择了0.6，0.7，0.8，0.9,1这5个支持度作为阈值。并且由于最后生成强关联规则的时间太长，只能在生成频繁项集后就停止计算，因此没有设置置信度。

* 1. 性能比较分析
     1. 自动生成数据T10I4D100K

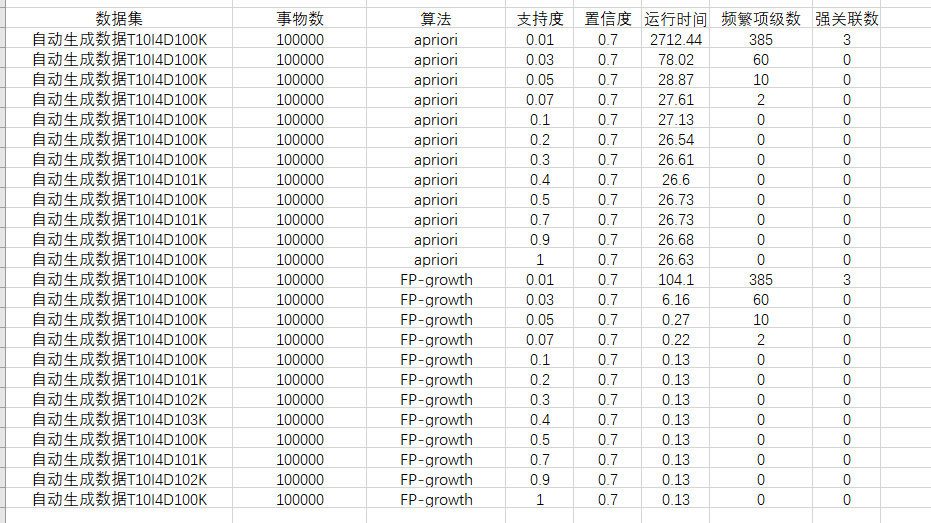


表1自动生成数据T10I4D100K不同支持度数据表

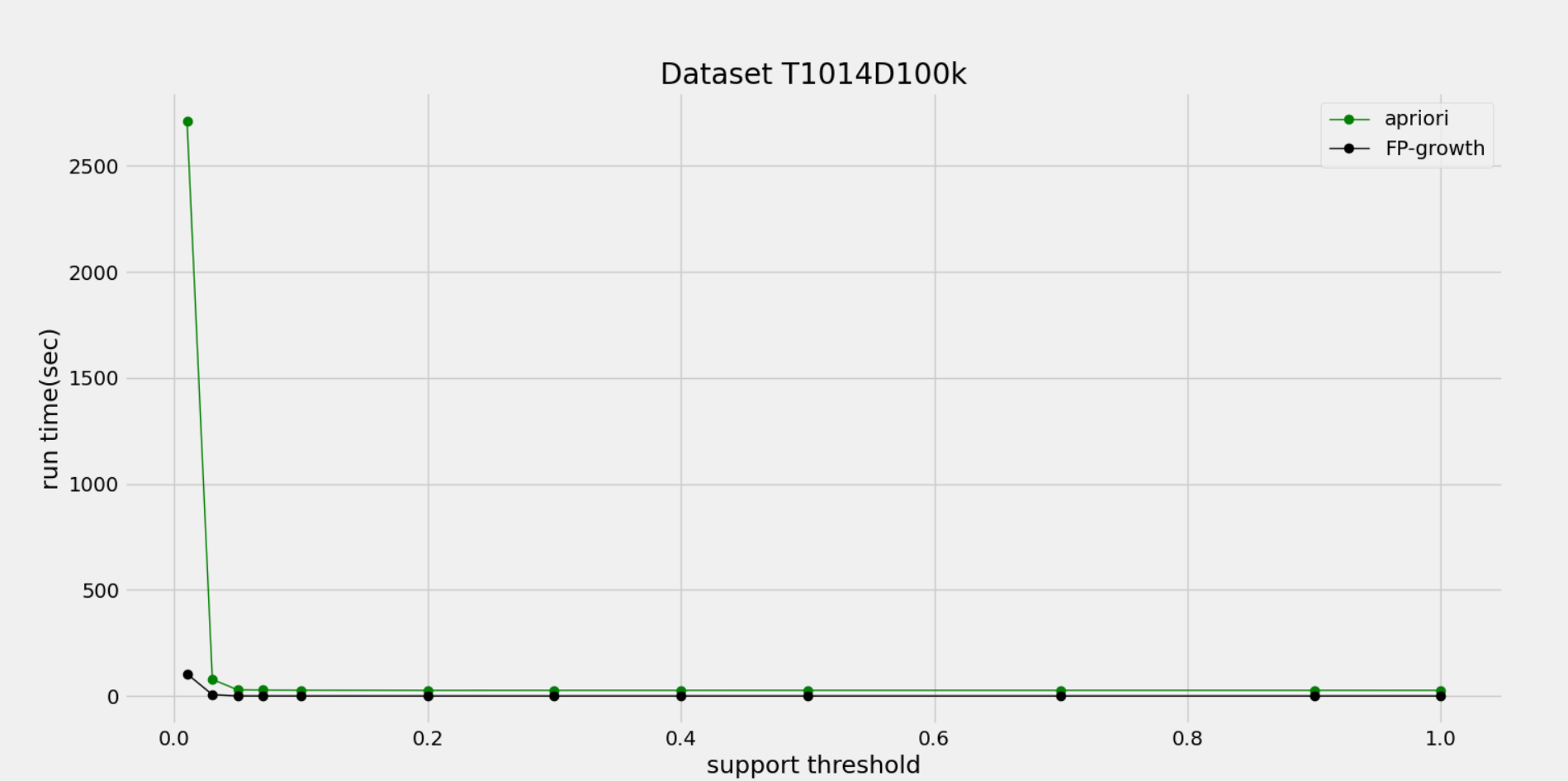


图3 T10I4D100K Apriori和FP-growth算法性能比较

根据表1和图3可以看出：

1. FP-growth算法性能明显优于Apriori，特别是当支持度阈值比较小的时候，FP-growth的优势愈发明显。
2. 当支持度设到0.1以上时，此时已经没有频繁项了，两个算法的运行时间不再随着支持度增高而下降，而是趋于平缓。
   * 1. 实际数据retail



表2 retail数据集不同支持度数据表

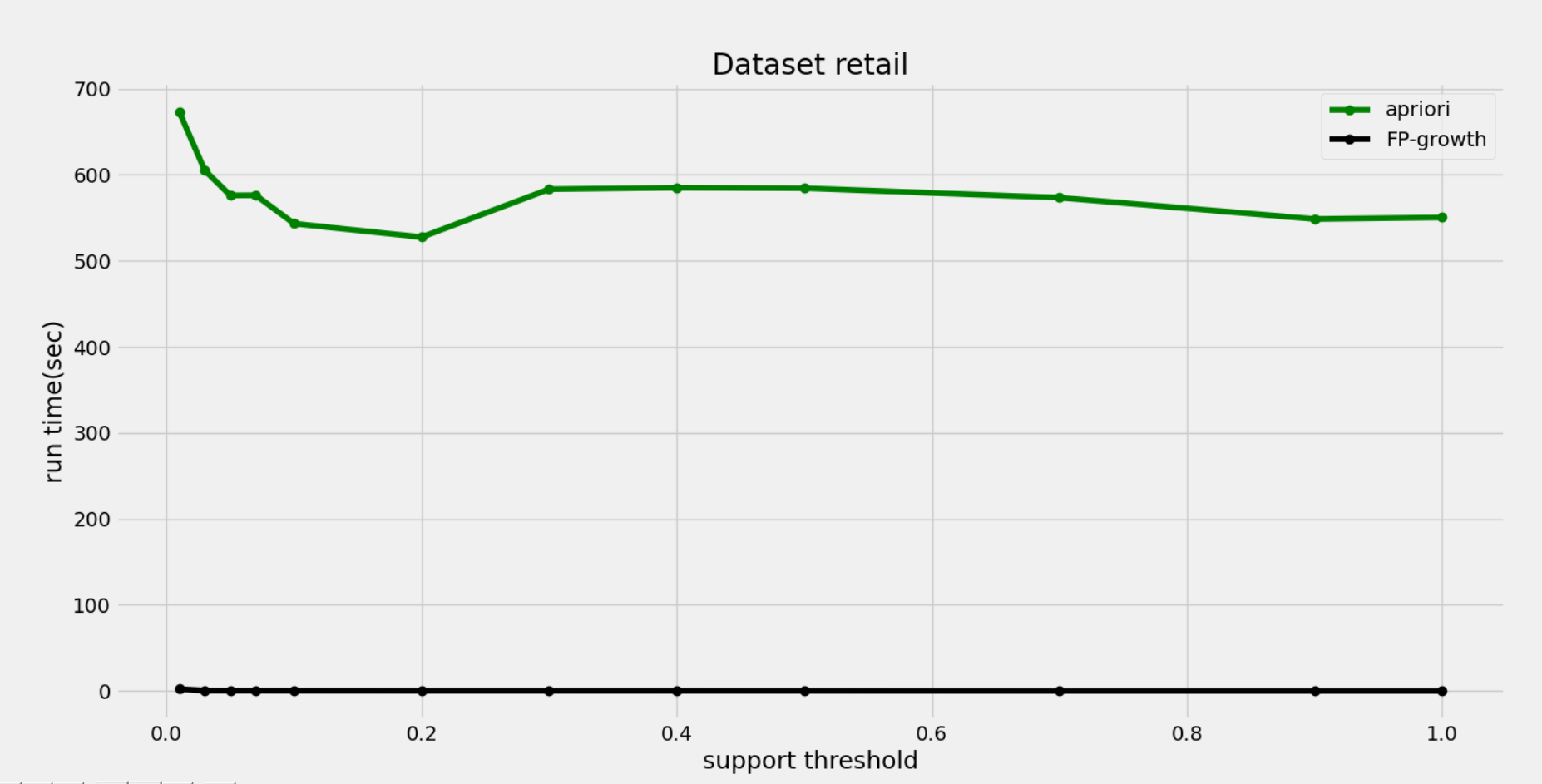


图4 retail Apriori和FP-growth算法性能比较

从表2和图4可以看出：

1. 相比于T10I4D100K，FP-growth的性能优势更加明显，
2. 相比于T10I4D100K，Aprioti在retail数据集上运行时间变长了很多，而Fp-growth却没有什么影响

我觉得这可能是数据集不同造成的影响，T10I4D100K和retail的事务数相差不多，但是元素种类差了很多，相比于T10I4D100K，retail的元素数要大的多，因此Apriori在每次扫描的时候需要耗费大量时间，而FP-growth却只需要两次。

* + 1. 稠密数据chess



表3 chess据集不同支持度数据表



图5 chess Apriori和FP-growth算法性能比较

从表3和图5可以看出：

1. 当支持度较低的时候，Apriori的运行时间成指数增长（当把支持度设为0.5的时候，程序运行了一天还没有算完），相比之下，FP-growth的运行时间比较容易让人接受。
2. 虽然chess的事务数比较少，元素数目也是最少的，但当支持度低于0.8的时候，不管是Apriori还是FP-growth的运行时间都很长（相比于前两个数据集）。

我觉得这是由于chess这个数据集的数据分布造成的，虽然事务数较少，但是其数据非常稠密，会造成每一次扫描的频繁项也非常多，由于Apriori需要多次扫描并且进行自连接操作，其运行时间就会随着支持度的降低呈指数增长。而FP-growth只需要扫描数据集两次，故运行时间少的多，但是由于数据稠密，其构建条件模式基的时间也会变长，所以也会花比其他数据集多的时间。

* + 1. 综合比较分析

1. 从数据集大小来看，数据集越大，两个算法的运行效率越低，但是FP-growth的运行时间会更快
2. 从数据分布来看，当数据集的元素数比较多时，会降低Apriori的运行效率，而FP-growth则很正常
3. 从最小支持度阈值设置来看，当阈值设置越低时，两个算法的运行时间一般会越来越长，不过阈值设的越低，FP-growth的性能优势会越来越明显；当支持度越来越高是
4. 从数据稠密性来看，当数据越稠密，两个算法的性能会都越来越差，但是Apriori更为严重，我觉得这是由于其需要多次扫描数据集并进行自连接造成的，这些操作需要多次循环迭代。
5. 总结

本文主要是对Apriori和FP-growth算法在不同最小支持度阈值和不同特点数据集上的性能比较分析，综合上面的实验结果可以看到，Apriori算法在各个角度都是比FP-growth的性能要差的，因此在选择关联分析算法的时候优先选择FP-growth.

参考文献

1. AGRAWAL R,SRIKANT R. Fast algorithms for mining association rules[C]. Proceedings of the 20th International Conference on Very Large Data Bases,1994:21-30.
2. CHEN L J,KEJI MAO K J. Research on mining association rules in university scientific projects management［J］. Network Computing and Information Security,2012,345(1):561-567.
3. HAN J W,KAMBER M,PEI J.数据挖掘:概念与技术[M]．范 明,孟小峰译．第３版．北京:机械工业出版社,2017:158-196.
4. 万 倩,谢 峰,赵 明,等．基于数据挖掘的数字电视套餐包交叉销售模型分析[J].广播电视信息,2016(12):57-59.
5. 梁伟. 关联规则的相关算法研究--基于Apriori和FP-growth算法[D]. 中国地质大学(武汉).