【换成计算机学报模板】

标题、摘要最后定。

1. 引言

推荐系统在近十余年来受到学术界的充分重视和深入研究，并在电子商务网站、社交平台、及视频音乐网站上得到广泛应用.各种各样的推荐模型和方法相继被提出，主要包含基于内容的方法、协同过滤模型、混合式模型、基于关联规则的方法等[Adomacicius-05].关联规则最早应用于购物篮分析[Tan-Book]，可揭示一组经常被一起购买的商品，因此自然而然地成为一种简明且可解释性极佳的推荐模型.例如，由于集合{单反相机,单反镜头,三脚架}中的商品经常一起被购买，当用户购买或频繁浏览“单反相机”和“单反镜头”时，电子商务网站就将“三脚架”推荐给用户.基于关联规则的推荐本质上利用了项目(item)之间的关联性，其机理非常类似于基于项目的协同过滤模型(Item-based Collaborative Filtering) [Linden-03]，因而很多文献[Nakagawa-03+]将基于关联规则的推荐归类于协同过滤模型.

关联规则推荐可能难以获得像基于奇异值分解的协同过滤[Koren-KDD08]这类复杂模型的精准度，但这并没有妨碍关联规则推荐方法的实用性.大量电子商务网站将基于关联规则的方法作为其商用推荐引擎，比如：YouTube使用关联规则推荐视频[Davidson-10]、淘宝和亚马逊网站上的“购买此商品的顾客也同时购买”及“经常一起购买的商品”等推荐方式也依托于关联规则构建. 关联规则推荐能获得大量实际应用的原因在于：(1) 推荐结果可由用户操作动态触发，即用户浏览或购买记录发生变化时，经过与关联规则库的快速匹配，推荐结果可以快速更新；(2) 推荐结果与浏览或购买商品之间具有明显的语义关联，带来优越的可解释性.

由于频繁模式的反单调性[Tan-Book]，一个频繁模式包含很多频繁子模式，而一个频繁模式也能导出多个关联规则，因此关联规则数量巨大、且多个规则蕴含同个目标项目的情况广泛存在.为了提升推荐准确率，大量研究工作围绕如何挑选高质量规则展开.早期的研究大多挑选置信度(confidence)最高的规则用于推荐[3篇]，文献[Wang-04]则挑选最长规则，文献[Li-01]则提出综合利用置信度、支持度(support)和长度对规则进行排序.近年来，一些新指标被认为能进一步提升关联规则推荐的性能，比如校正置信度(adjusted confidence)[Rudin-11]、分离置信度(disjunctive confidence)[Ghoshal-14]、互信息[Ghoshal-15]等.还有一些研究认为需融合目标项目相同的规则集得到综合推荐分值，文献[Lin-02,Wang-14]将多个规则的统计量加和计算推荐分值，文献[Wick-09]引入D-S证据理论合成多规则的分段支持度值(partitioned support)作为推荐分值.

如

含相关工作，最后写。

动机：很多规则推荐计算方法，但是跟本文正交，本文作为已有研究的重要扩充，解决效率问题。

1. 问题描述和总体框架
   1. 关联规则推荐的可扩展性问题

关联规则推荐是怎么做的。数学化的描述。

大量在线用户，实时推荐时，为什么有效率问题。

* 1. 总体分布式框架



图1. 面向关联规则推荐的可扩展分布式总体框架

提出总体框架，并清晰描述；突出本文在框架中的贡献，衔接第3部分。

1. 技术细节（找找计算机学报论文，别人怎么定类似标题的）

注：设计的例子贯穿本节，图表清晰；符号的规范：行中不出现公式编辑器，如不可避免尽量公式；其他规范：标点、字体参考CJC模板，避免错别字。

* 1. 组投影与分布式频繁模式挖掘

这块不算贡献，简要介绍清楚FP-growth分布式机理。

* 1. 有序模式森林的定义

衔接，受组投影启发，定义有序模式森林。引出定义

* 1. 基于路径搜索的推荐计算

…有。

* 1. 负载均衡分割方法

这块根据实验结果写。

1. 实验结果和分析

一周内把2-3小节写差不多了，开始写实验部分。

1. 总结

参考文献 [边写边整理]

题目：大数据背景下模式挖掘和匹配负载均衡策略

摘要

基于关联规则推荐的方法是一种用来构建推荐系统最常见的技术，它被广泛应用于商业活动。基于关联规则的推荐包括两个环节，关联规则的挖掘和基于关联规则的推荐过程，在关联规则的挖掘阶段，PFP-Growth算法是一种广泛被采用的并行模式挖掘算法，本文在实现Spark平台下PFP-Growth算法的前提下，提出一个合理的分组策略，提高挖掘的负载均衡性。在基于关联规则的推荐阶段，不同的技术（包括合适规则的选择和多种规则的组合）着重于提高关联规则推荐的有效性，而忽略了关联规则推荐方法的伸缩性的问题。基于关联规则方法的计算复杂性很有可能随着在线用户和规则的增长而急剧增加，在传统的电子商务平台一般都会有七百多万。然而，用户行动的动态性要求推荐系统做出接近实时的推荐，这进一步强调了基于关联规则推荐系统的伸缩性问题。

在这篇文章中，我们提出了一个关联规则挖掘的负载均衡策略，提出了一个分布式框架，这个框架可以用一个统一的方法扩展各种基于关联规则推荐方法。特别的是，在总结现存的基于规则的方法的基础上，一个通用的树形结构用来分开存储不同种类的模式，接着一个高效的算法被设计出来用来挖掘合适的模式并计算推荐分值。为了处理持续增长的在线用户，一个分布式的框架被提出来，两个划分树的负载均衡策略被提出用来分别针对稀疏和稠密的数据集。通过框架，基于关联规则推荐系统的效率可以明显地提高。

关键词：

英文摘：

引言

研究动机：

问题定义和分析

图

相关工作

模式挖掘

模式匹配

无论是模式挖掘阶段还是基于关联规则的推荐阶段，都涉及到多机调度问题，在给定交易数据和模式的前提下，如何划分用户交易数据的子交易数据和模式到各个计算节点，从而获得负载均衡和高效率表现是我们所关心的问题。

假定最优的负载是L\*，我们有L\*≥1/k，因为K个计算节点至少要做1/k部分的任务。为了最小化全局的负载L等同于发现S的k个子集，这些子集的负载Lk (1 ≤ k ≤ K)尽可能等同。子树和问题是NP完全问题，也就是说，给定一个有限的整整数集合和一个正整数t，问是否存在一个子集他们的和为t。如果我们令，在两个计算节点上的负载均衡分区问题等同于子集和的问题。因此，负载均衡分割问题毫无疑问是一个NP完全问题。在文献中，处理负载均衡分区问题最好的近似算法是最长处理时间策略，他被证明是4/3近似，也就是。详细的，LPT策略是将q个子树按照负载指数降序排列，接着将子树一个一个放进当前负载和最小的计算节点上。

负载指数上的考虑 NP完全的LBP问题和4/3近似算法都是被广泛认可的。

1. 挖掘频繁模式（一个贯穿首尾的例子）

A E G

A B C D E

A B C D E G

B C D F

C E F G

B D E F G

B D E F

F G

C F

A F

支持度 A:4 B:5 C:5 D:5 E:6 F:7 G:5

MinSupport = 20%

FList排序：F E B C D G A

负载分组（2组）： 组1：F C G（7+5+5 = 17）组2：E B D A（6+5+5+5 = 21）

Trans排序：

E G A

E B C D A

E B C D G A

F B C D

F E C G

F E B D G

F E B D

F G

F C

F A

子trans的分区结果：

组一：E G，E B C，E B C D G，F B C，F E C G，F E B D G，F，F G，F C，F

组二：E G A，E B C D A，E B C D G A，F B C D，F E，F E B D，F E B D，F A

两组分别运行Fp-Growth算法，将得到的结果聚合得到最终结果。（后面有大数据实验结果，这里只是例子）

现在，将被探索的问题就是在子trans分区的背景下如何设计合理的负载指数。给定用户交易数据集，首先能想到的是项目的支持度，项目的支持度显示此项目在用户交易数据中出现的可能性的大小，支持度大的则此项目在多个用户交易数据中出现，支持度小的则此项目在少量用户交易数据中出现。在模式挖掘的情况下，支持度大的项目将会出现在多条树的路径下，则挖掘的工作量也会变大，由此可见，支持度则是一个好的负载指数。

方案一（官方）：在Spark MLlib库中的fpm包采用的是Hash方法求得子trans末尾item的hash值，将子trans分到相应的节点。

方案二（顺序）：将FList均匀分割，形成多个分区，根据子trans末尾item所在的分区号，分到相应的分区。

方案三（自己）：将FList中的项目按顺序放入当前支持度和最小的分区里，并获得item的分区号，在划分子trans时，根据子trans末尾item所在的分区号，划分到相应的分区。

1. 基于关联规则推荐（一个贯穿首尾的例子）

例子：根据上面算法 得到的47个pattern（这里还要对pattern按support升序排序）

G F 30.0，G E F 20.0，G E 40.0，E F 30.0，D G E 20.0，D G 20.0，D F 30.0，D E F 20.0，D E 40.0，C G E 20.0，C G 20.0，C F 30.0，C E 30.0，C D E 20.0，C D 30.0，B G E 20.0，B G 20.0，B F 30.0，B E F 20.0，B E 40.0，B D G E 20.0，B D G 20.0，B D F 30.0，B D E F 20.0，B D E 40.0，B D 50.0，B C E 20.0，B C D E 20.0，B C D 30.0，B C 30.0，A G E 20.0，A G 20.0，A E 30.0，A D E 20.0，A D 20.0，A C E 20.0，A C D E 20.0，A C D 20.0，A C 20.0，A B E 20.0，A B D E 20.0，A B D 20.0，A B C E 20.0，A B C D E 20.0，A B C D 20.0，A B C 20.0，A B 20.0

1.模式在树上的压缩存储

构建树

。

。

。

树上基于规则的推荐

。

。

。

集成推荐结果计算和挖掘基本模式

。

。

。

2. 分布式框架

基于关联规则的推荐方法可能适合大数据有两个方面。第一个就是大量的频繁模式导致GOPC-Tree规模变大。另一方面就是大量的在线用户正在等待实时的推荐。在这一节，我们提出一个基于GOPC-Tree的分布式框架缓解大数据的问题。

综述

频繁模式挖掘并不严格受制于相应时间，它通常以离线模式完成。如果交易数据非常大，并行的FP算法，或者甚至spark上的FP算法可以被用来有效的挖掘频繁模式。与此相反的是基于规则的推荐应该在线处理，因为需要立刻给活动用户进行推荐。访问一个流行商业网站的用户数量也是特别大的。所以，一个切实可行的办法往往是把推荐服务分不到多个服务器上。

图显示了使用树的分布式基于规则推荐的整体框架。特别的，这个分布式框架灵活的整合了两种分区机制。第一，把大量的在线用户分成多个组，每一个分组分部到一个或多个计算节点。这些节点存储了完整的树，并且负责生成推荐结果给部分用户。事实上，用户分组用户分区非常相似广泛使用的代理缓存服务器，复制静态内容减少网络流量。

第二，如果部分用户仍然很大而且频繁模式也非常大，我们可以进一步分解树来提高效率。详细的，树可以被看做是具有null根节点的森林。例如，图的树，如果root节点被删除，可以被看做是具有六个子树的森林。因此，第二层的每一个子树都可以分布到一个特殊的计算节点上的。结果初始化的树被分别保存到多个计算节点中，（假定有K个节点）。基于规则的推荐分解树后，包含三个操作，1.把用户广播到k个计算节点上，2。在每个节点上为每个用户单独的运行算法2。3.根据用户的id聚合模式或者与用户有关的项目-分数对来获得最后的推荐结果。

负载均衡分割

现在，将被探索的问题就是在树分区的背景下如何设计合理的负载指数。给定一系列用户购物篮数据，基于关联规则推荐在每个子树上所需要的执行时间是后验的，它确实受到一些像子树大小，待服务的人数，用户购物篮的长度，等等不确定因素的影响，因此我们尝试发现一些子树的负载指数，这些指数和每个子树上产生的任务的执行时间是强关联的。与此同时，负载指数应该很容易获得。接下来我们考虑两种可能的负载指数并提供经验性分析。

1.根节点的支持度：直观的，根节点的支持度显示子树有多“忙”，这里一个忙的子树意味着它将服务更多的用户生成合适的模式。如果子树服务一部分用户，支持度就是一个很好的负载指数。因此，我们所感兴趣的是，在给定任何Tu时，子树未能为Tu服务并产生合适模式的可能性。假设训练项目和Tu中所包含项目的支持度服从独立同分布，不失一般性的，使v0子树的根节点，v1~vl是v0的孩子节点。根据算法2，如果Tu不包含v0同样也不包含vj, 1 ≤ j ≤ l,以V0为根节点的子树将不会为Tu产生人格合格的模式，这里的事件被定义为A。我们有。

当->1时，事件A发生，此时需要->0，0 ≤ j ≤ l，因此，如果数据集大而极其稀疏，项目的支持度很有可能非常小，因此趋近于1。在这个例子中，对于负载指数来说，支持度是一个合理的选择。相反的支持度不是个作为密集数据集的负载指数。我们将在实验章节展示证据。

2.子树的大小：因为支持度通常很大，所以事件A几乎不可能发生，换句话说，每个子树倾向于给大量的用户生成合适的模式。在这个例子中，每个子树的负载很大程度上由挖掘基模式的事件决定。因此，子树的大小则成为又一个负载指数，在最差的情况下，通过遍历整个子树来获得合格的模式。正如XX节所提到的，每个模式被存储在树的一个节点上，因此子树的大小被有序模式的数量所估计，根节点是他们的第一个项目。而且，一组项目被挖掘和排序后，起始于每个项目模式的数量都是可得的。

注意：一些组合多种因素的启发式的负载指数可能会被进一步考虑。然而，简单来说，我们使用两种负载均衡策略，分别使用根节点的支持度和子树的大小作为负载指数。在实验章节中，我们将会展示两种简明的负载指标，他们具有互补的优势，分别处理稀疏和稠密的数据。

实验结果和分析：

实验设置

参数 环境

有效性 数据集

实验结果

图表展示

效率提升 加速比

总结

参考文献