计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2017.20160554
54(9):1945-1957,2017

面向流数据的决策树分类算法并行化

季一木[1,2,3,4] 张永潘[1] 郎贤波[1] 张殿超[1] 王汝传[1,2]
[1](南京邮电大学计算机学院 南京 210023)
[2](江苏省无线传感网高技术研究重点实验室(南京邮电大学) 南京 210023)
[3](南京邮电大学先进技术研究院 南京 210023)
[4](高维信息智能感知与系统教育部重点实验室(南京理工大学) 南京 210094)
(jiym@njupt.edu.cn)

Parallel of Decision Tree Classification Algorithm for Stream Data

Ji Yimu[1,2,3,4], Zhang Yongpan[1], Lang Xianbo[1], Zhang Dianchao[1], and Wang Ruchuan[1,2]
[1](*School of Computer Science and Technology, Nanjing University of Posts and Telecommunications, Nanjing* 210023)
[2](*Jiangsu High Technology Research Key Laboratory for Wireless Sensor Networks (Nanjing University of Posts and Telecommunications), Nanjing* 210023)
[3](*Institute of Advanced Technology, Nanjing University of Posts and Telecommunications, Nanjing* 210023)
[4](*Key Laboratory of Intelligent Perception and Systems for High-Dimensional Information (Nanjing University of Science and Technology), Ministry of Education, Nanjing* 210094)

Abstract With the rise of cloud computing, Internet of things and other technologies, streaming data exists widely in telecommunications, Internet, finance and other fields as a new form of big data. Compared with the traditional static data, stream data in big data has the characters of rapidness, continuity and changing with time. At the same time, the implicit distribution of the data stream will bring about the concept drift problem. In order to satisfy the requirements of stream data classification algorithms in big data, we must improve the traditional static offline data classification algorithms, and propose P-HT parallel algorithm based on distributed computing platform Storm. To meet the requirements of Storm stream processing platform, we improve the flexibility and versatility of the algorithm through sliding window mechanism, alternative tree mechanism and parallel processing mechanism, and the algorithm can adapt to the concept-drift of data stream very well. Finally, we experimentally verify the validity and high efficiency of the algorithm. The results show that the improved P-HT algorithm has better throughput and faster processing speed than the traditional C4.5 algorithm in the case of no reduction in accuracy.

Key words stream data; classification algorithms; Storm platform; sliding windows; C4.5 algorithm; paralleling algorithm

摘 要 随着云计算、物联网等技术的兴起,流数据作为一种新型的大数据形态广泛存在于电信、互联网、金融等领域.与传统静态数据相比,大数据环境下的流数据具有快速、连续和随时间变化等特点.同时数据流的隐含分布变化会带来概念漂移问题.为了适应大数据环境下流数据分类算法的要求,必须对传统的静态离线数据分类算法进行改进,提出基于分布式计算平台 Storm 的 P-HT 并行化算法.算法

收稿日期:2016-08-02;**修回日期**:2016-12-09
基金项目:国家自然科学基金项目(61170065);江苏省自然科学基金优秀青年基金项目(BK20170100);国家重点研发计划(2017YFB0202200);江苏省重点研发计划项目(BE2017166)
This work was supported by the National Natural Science Foundation of China (61170065), Outstanding Youth of Jiangsu Natural Science Foundation (BK20170100), the National Key Research and Development Program of China (2017YFB0202200), and the Jiangsu Key Research and Development Program (BE2017166).

在满足 Storm 流处理平台要求基础上,通过滑动窗口机制、替代子树机制和并行化处理,提高了算法的灵活性和通用性,并且能良好地适应数据流的概念漂移.最后通过实验验证该算法的有效性和高效性,结果表明在与传统 C4.5 算法相比精度没有降低的情况下,改进的 P-HT 算法具有更大的吞吐量和更快的处理速度.

关键词 流数据;分类算法;Storm 平台;滑动窗口;C4.5 算法;并行化算法

中图法分类号 TP391

数据挖掘近年来正逐渐成为经济学、人工智能等领域的研究热点,在当前的大数据环境下,流式数据具有快速性、连续性、变化性和多样性等特点[1],数据流挖掘是数据挖掘领域的一个重要分支.各种流数据挖掘的技术和算法相继被提出并得到运用.如何从连续不断的数据流中挖掘出潜在的关联和有用的信息,成为目前我们面临的一项重要挑战.与传统的数据模型不同,数据流模型具有3个特性:1)数据到达速度快,实时性强;2)数据量巨大,难以将所有的数据都有效地存储下来;3)数据一旦经过处理,除非需要保存供后续使用,否则不能被重复扫描,再次扫描数据消耗过大.然而传统的数据挖掘方法在处理之前就将数据全部存储记录下来,进行分析时访问存储介质进行挖掘.但流数据环境下数据是快速到达的,且数据规模巨大,数据流的隐含分布变化会带来概念漂移问题,传统数据挖掘技术难以满足数据流挖掘的要求[2].

分类是一种非常重要的数据挖掘技术,其目的是根据已有的数据集学习构造一个分类函数或分类模型,该分类模型能够将新到样本映射到一个具体的类别上.传统的分类模型包括决策树(decision tree)、贝叶斯算法(Bayes algorithm)、神经网络(neural network)、K 近邻分类算法(K-nearest neighbour)、支持向量机(support vector machine)等[3].其中,决策树模型[4]是最普遍的一种分类模型,相比与其他的分类模型,决策树模型能简单地生成可以理解的规则,同时计算量相对较小,而且可以处理多种数据类型.它是一种依托于策略抉择而建立起来的树,着眼于从一组无次序、无规则的实例中推理出以决策树表示的分类规则.因而决策树模型可以清晰地显示出模型中的核心字段和潜在信息.

流数据的决策树分类方法比传统的分类在实时性和存储限制等方面面临更多的挑战[5],传统数据分类方法,往往很难满足数据流分类的需求和要求,因此需要将传统分类模型进行重新调整.在文献[6]中,Domingos 等人提出的 VFDT 算法是一种流数据环境下的分类算法,VFDT 基于 Hoeffding 不等式建立决策树,每当新样本流入时,VFDT 都将该新样本沿着决策树从上到下遍历,最终到达树的叶子节点.Gama 等人[7-8]在 VFDT 的基础上提出 VFDTc 算法,使其能够处理连续型数据.但是 VFDTc 算法对每一个可能的属性值均计算信息熵,这在高速数据流环境下会大大加剧计算的负担.Jin[9]同样针对 VFDT 算法处理连续型属性的局限,改进提出了 NIP 算法,NIP 通过将连续属性值划分成不同的小片段,计算每个离散的小片段的信息熵,然后将有可能成为最优分割点的离散片段保留下来,将其余的片段删除.将连续属性离散化的方法,能够显著地缩小寻找最优分割点的范围和计算量,但是如果错误地删除了包含分割点的片段,整个决策树的构建和分类效果都将受到很大的负面影响.Anagnostopoulos 等人[10]提出一种基于概率估计法的流分类算法,但由于数据本身的影响,估计的结果会出现较大的偏差.

在数据流的隐含分布不断变化的情况下,模型的准确率和稳定性不能得到保证,即发生了概念漂移问题.文献[11]中通过时间序列分析方法对现有的处理概念漂移的策略进行了分类,并描述了自适应的学习过程.文献[12]中,Gama 提出时间滑动窗口机制,提高了模型训练的实时性.文献[13]提出的 CVFDT 算法是对 VFDT 算法的扩展,它保持了 VFDT 的速度和准确率,并且当数据流的隐含信息不断变化时,它能更好地保证模型的实时性和准确度.Kuncheva 等人[14]对流数据分类模型深入研究,发现模型的精确性不仅跟数据本身的质量和概念漂移发生的程度相关,滑动窗口的大小也会对准确率产生很大的影响.因此 Kuncheva 等人提出了基于可变滑动窗口的流数据分类方法.当数据流发生概念漂移时,滑动窗口能够自适应地调整窗口大小,使得分类模型能够更加有效地抵抗概念漂移,保证了分类模型的实时性和精确性.Liang 等人[15]在 CVFDT 算法的研究基础上,改进和扩展了 CVFDT,提出 puuCVFDT 算法,该算法可以对没有标签和属

性值不确定的数据进行处理.与其类似,Zheng[16]同样对 CVFDT 算法进行改进,提出了 CFDT 算法,该算法引入了一种新的树型索引结构,首先采用聚类方法对数据进行预处理,进而对聚类簇进行扫描并抽取出隐含的信息,从而对树的结构进行调整,这种方法使得 CFDT 树既能提高模型的实时性,也保证了较高的分类准确率.表 1 对比了 4 种常见的分类算法.

Table 1 The Comparison of Common Classification Algorithms

表 1 常见的分类算法比较

Algorithm	Training Speed	Error Rate	Incremental Taining	Concept-drift	Data Type
C4.5	Slow	Low	Not Support	No	Static
VFDT	Fast	Low	Support	No	Continuous
VFDTc	Slow	Low	Support	No	Continuous
CVFDT	Fast	Low	Support	Yes	Continuous

表 1 中提到的经典 C4.5 算法[17]是由 ID3 发展而来的决策树算法,相比于 ID3 算法,C4.5 的特点主要在于 C4.5 用信息增益率来选择属性,且 C4.5 能够完成对连续性数据的离散化处理,从而克服了 ID3 只能处理离散型数据的缺点.但是 C4.5 是针对离线的静态数据集进行信息的挖掘.VFDT 是基于 Hoeffding 树的流分类算法,适应了大数据环境下的流数据分类要求,但是它不能处理连续性数据且当数据流发生概念漂移时,模型的精度会大大下降.VFDTc 算法对 VFDT 所能处理的数据类型进行扩展,但是 VFDTc 算法对每一个可能的属性值均计算信息熵,这大大加剧了计算负担.CVFDT 算法解决了流数据的概念漂移问题,且可以加入连续属性的处理机制,但是对于决策树的建立是串行化的方法,在高速的流数据环境下,模型训练的速度不能达到要求.文献[18]中提出了一种分布式的实时情感大数据流分析算法;文献[19]提出了一种并行霍夫丁决策树的方法,提高了实时决策树分类算法的效率,但是并没有解决流数据环境下的分类处理需要考虑有限内存、时间序列和单次处理的特点.Storm 是 Twitter 开源的一个免费的分布式流计算系统,它能够对持续大流量的数据流进行实时分析和快速响应,并且支持大数据流的并行化处理.

针对 VFDT 算法在流数据环境下不能解决概念漂移的问题,以及 CVFDT 算法对于大数据环境下的有限内存和训练速度问题,结合分布式并行流处理平台 Storm 在实时流数据处理上的快速性、实时性和安全性的优点,本文在 CVFDT 算法研究的基础上,提出基于 Storm 平台的实时 P-HT 并行化分类算法.算法引入可变滑动窗口,当发生概念漂移时,窗口及时地收缩,有利于较快地适应概念漂移,防止精度的大幅降低.同时提出了一种并行 Hoeffding 树的方法,缩短了节点分裂时的计算时间,从而提高了决策树模型训练的速度.与 CVFDT 算法相比,P-HT 有相同的精度和分类效率,但是加入 Storm 集群的并行化计算使得算法的建树效率得到很大的提高.

1 CVFDT 分类算法和 Storm 流处理平台

1.1 CVFDT 算法

定义 1. Hoeffding 树.对一个真值随机变量 r,其取值范围为 R,假定我们对 r 取了 n 个独立的观察值,并计算了它们的平均值 $\bar{r}$,其 Hoeffding 约束对于可信度 $1-\delta$,变量 r 的真实值至少是 $\bar{r}-\varepsilon$,其中 $\varepsilon=\sqrt{\frac{R^2\ln(1/\delta)}{2n}}$,这里的 r 是信息增益,R 的取值范围是 $\mathrm{lb}(Classes)$,$Classes$ 是类别的数量.Hoeffding 树中每个叶节点的内存占用为 $O(dvc)$,其中,d 为属性数目,v 为每个属性可能的最大取值数目,c 为类别数目.

CVFDT 算法是一种基于 Hoeffding 不等式建立决策树的方法,基于小样本足以选择最优的分裂属性,使用 Hoeffding 边界量化叶节点中确定最优分裂属性所需要的样本个数.

定义 2. 信息增益.是用来衡量给定的属性区分训练样例的能力,计算公式如式(1)(2):

$$Entropy(S)=\sum_{i=1}^{c}-p_i\mathrm{lb}(p_i), \tag{1}$$

$$Gain(S,A)=Entropy(S)-\sum_{v\in Values(A)}\frac{|S_v|}{|S|}Entropy(S_v), \tag{2}$$

式(1)是信息熵的计算公式,其中 p_i 是在样例集 S 中不同类别 C_i 的样本的比例.式(2)是信息增益

的计算公式,其中 $Values(A)$ 是属性 A 所有可能值的集合,S_v 是 S 中属性 A 的值为 v 的子集(即 $S_v=\{s\in S|A(s)=v\}$).

CVFDT 算法过程包括 3 个部分:CVFDTGrow 过程、ForgetExample 过程、CheckSplitValidity 过程,其总体流程图如图 1 所示. CVFDT 算法中的 CVFDTGrow 过程与 Hoeffding Tree 算法的生成树类似,但是 CVFDT 通过保存每个节点上的统计数来检验原先的 HT 决策树的有效性(而不是像 VFDT 只统计叶子节点). 因为新的样本不断参与模型的生成导致 HT 树不断地生成和改变,删除旧的样本将出现困难. 因此,在建立一个节点时会分配一个单独的单调增加的 ID. 滑动窗口 $\boldsymbol{W}$ 是一个保存实时样本的先进先出队列,当新的样本进入滑动窗口,随之旧的样本将要从窗口中滑出时,旧样本所经过的所有内部节点的统计值 N_{ijk} 减 1,同时新的样本将从根节点开始遍历至最深的叶子节点,经过的所有节点统计值 N_{ijk} 加 1.

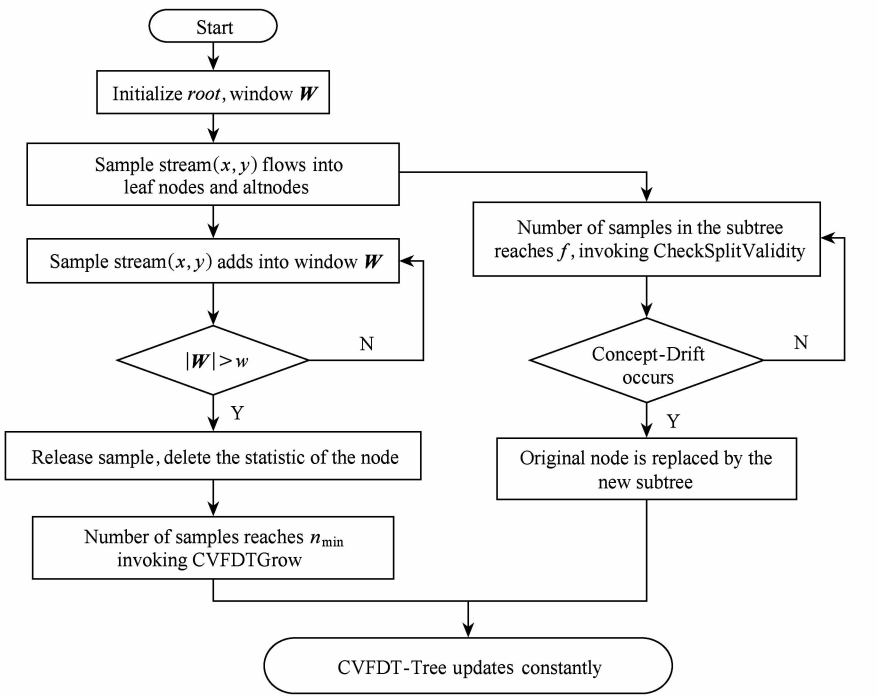

Fig. 1 The process of CVFDT algorithm

图 1 CVFDT 算法流程图

CheckSplitValidity 过程为:叶节点开始从数据流收集样本,随着样本数量的增多,能够以较高的置信度确定最佳划分属性时,则将该叶节点变成一个测试节点,然后对新的叶节点不断地重复该学习过程. CVFDT 维持一个训练样本的窗口,并通过在样本进入和流出窗口时更新已学习的决策树,使其与训练样本窗口保持一致. 当一个新样本到达之后,它将被加入到其所经过的所有决策树节点;而当将一个样本从决策树中去除时,它也需要从所有受其影响的节点中移除,并且所有的统计测试都需要重新进行. 当 CVFDT 怀疑有概念漂移发生时,它就并行在该节点生成一棵备选子树. 当备选子树的精度远大于原先的子树时,原始的子树被替换并释放.

1.2 Storm 实时处理平台

Storm 是 Twitter 支持开发的分布式、开源的、实时的流处理平台. Storm 集群采用基于 YARN 的管理模式,集群的架构为主从式(master/slaves),由一个主节点和多个工作节点组成,Storm 基于 YARN 的集群的架构如图 2 所示:

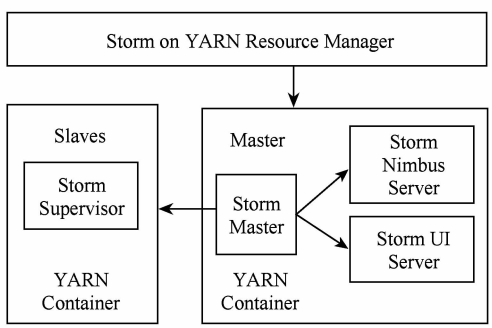

Fig. 2 Storm on YARN cluster architecture

图 2 Storm on YARN 集群架构

其中,Nimbus 和 Supervisor 都是快速失败、无状态的,所以某一个节点宕机立即重启不会影响系统的运行,主节点和工作节点之间的协调是通过 Zookeeper 而完成的.

1) Nimbus. Nimbus 是主节点,客户端上传的 jar 包会上传到 Nimbus,Nimbus 进行代码分发、任务分配、集群状态监控等.

2) Zookeeper. 负责集群的协调、共有数据的存放(如心跳信息),主节点把任务分配信息写到 Zookeeper,各个工作节点会不时地从 Zookeeper 获取自己的任务,某一个节点连接超时,则认为该节点失败.

3) Supervisor. 对应一台物理节点,用于启动 worker.

在 Storm 中,一个实时的计算应用程序被封装成一个任务拓扑(topology),类似于 Hadoop 的 MapReduce Job,Storm 拓扑由 Spout、Bolt 组件和 Streams 组成. 其中,Spout 是整个 topology 的数据源,将数据发送给相应的 Bolt;Bolt 负责对接收到的数据进行计算,实现过滤、查询等功能,可以级联,也可以向外发送数据流. 每个 Spout,Bolt 在集群中都是多线程运行的,消息的传递根据 StreamGrouping 完成,如图 3 所示. 一个实时应用的计算任务被打包成任务拓扑后发布,任务拓扑一旦提交后将会一直运行着,除非显式去中止. 数据流是 Storm 对数据进行的抽象,它是时间上的无穷的 Tuple 元组序列,数据流是通过流分组(stream grouping)所提供的不同策略实现在任务拓扑中流动. 此外,为了满足确保消息能够且能被计算一次的需求,Storm 还提供了事务任务 topology.

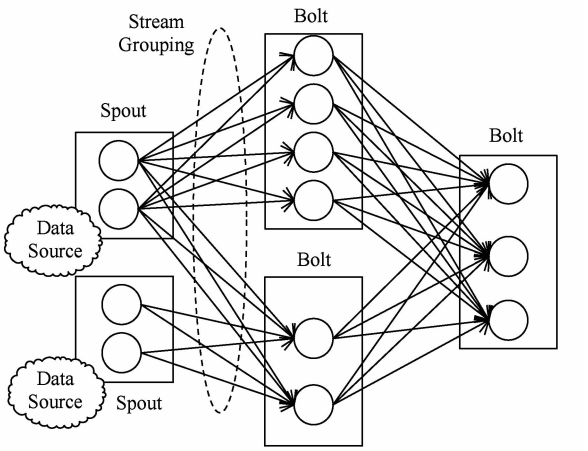

Fig. 3 Storm stream grouping

图 3 Storm 数据流

2 基于 Storm 平台的 P-HT 并行化算法

2.1 可变滑动窗口

滑动窗口模型随着数据流中样本的到达,新的数据不断插入到滑动窗口中,窗口的时间差或者数据元素数量保持不变,因此可以保证基于滑动窗口中的样本所生成的模型是随着数据流的变化而不断更新的,这保证了模型的实时性和准确性. 通常窗口的大小在模型训练中是固定不变的,这是由用户根据经验初始化的一个阈值. 然而在数据流的隐含分布不断变化的情况下,初始化的窗口大小值 *windowsize* 并不是最优的. 初始化的值过大会导致模型不能较好地抵抗概念漂移,初始值太小会使得模型训练样本不充足而导致准确率下降.

因此本文结合 Storm 的分布式特点,提出一种实时的并行化窗口方案,基于 Storm 的并行化窗口方案初始化多个窗口将实时样本流分为 S_1 和 S_2,分别监测 2 条流的隐含信息分布,根据泊松过程和 Hoeffding 不等式得到式(3):

$$Pr\{\boldsymbol{X}\}\geqslant(1+\varepsilon)E[\boldsymbol{X}]\leqslant \exp\{-((1+\varepsilon)\ln(1+\varepsilon)-\varepsilon)E[\boldsymbol{X}]\}, \tag{3}$$

其中,$E(\boldsymbol{X})$是样本流的期值,ε 是需要检测概念漂移的极限. 根据泰勒级数式(4)和泊松过程期望的式(5):

$$\ln(1+x)=\sum_{n=1}^{\infty}(-1)^{n+1}\times\frac{x^n}{n}=x-\frac{x^2}{2}+\frac{x^3}{3}-\cdots, \tag{4}$$

$$E[\boldsymbol{Y}_n]=nE[\boldsymbol{X}_n]=n\lambda_x, \tag{5}$$

得出判断样本流发生了概念漂移的阈值:$\varepsilon=\sqrt{2\lambda\ln(2/\delta)(1/n_1+1/n_2)}$,文中将样本流分成 2 条流 S_1 和 S_2,则 n_1 为样本流 S_1 中的样本数目,n_2 为样本流 S_2 中的样本数目. δ 为 Hoeffding Bounds 的置信度,λ 为样本流内样本的均值. 初始时,用户给定一个窗口的最大值 *windowsize*,流数据样本按照到达时间先后依次进入滑动窗口中并被记录下来. 若样本流 S_1 和 S_2 内样本的期望值 $\mu_1-\mu_2\leqslant\varepsilon$ 时,说明可能有突变型概念漂移的发生,P-HT 算法将适当缩小窗口的大小;反之,若 $\mu_1-\mu_2>\varepsilon$ 说明数据流的隐含信息分布并没有发生变化,此时 P-HT 算法将适当地放大窗口以接收更多的训练样本,从而提高模型的稳定性和精确性. 滑动时间窗口的工作过程如图 4 所示:

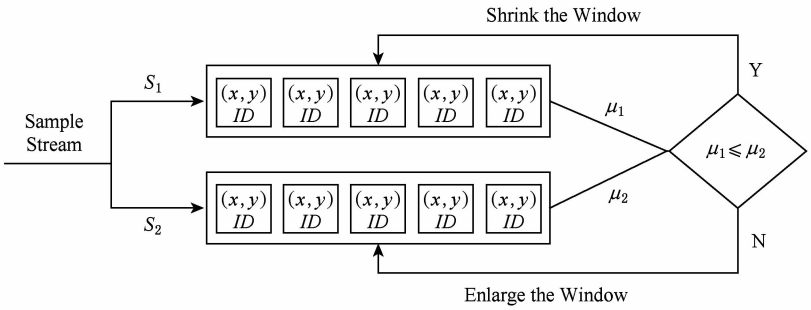

Fig. 4 The process of sliding window

图 4 滑动窗口示意图

2.2 垂直并行化

决策树的构建是一个反复迭代的过程,用传统的串行方法来建树,规模很小的数据量也需要花费大量的时间和资源,大数据环境下的流分类算法更是需要有效地提高模型的训练速度,减小资源消耗.为了解决流数据分类算法面临的有限内存和训练效率问题,最有效的方式是采用分治法,将原有的计算任务分解成若干个相同的子任务来处理,使得每个计算机节点均衡负载.根据分治法思想,结合决策树的生长过程,可以提出 3 种并行化的策略:任务并行化、水平并行化和垂直并行化.

1) 任务并行化的思想是各个树节点之间的并行.当内部节点进行分裂后将产生若干个不同的叶子节点,各个叶子节点再次进行分裂任务时,是不存在任何的依赖关系.这种并行方式不仅仅局限于兄弟节点之间,其他兄弟节点的孩子节点也是相同的并行性关系.但是这种方式的并行逻辑关系较为复杂,且使得各个计算机节点之间的通信带变得很复杂,也加大了交互过程的资源消耗.

2) 水平并行化的思想是基于数据集的并行方法.将原始数据集划分成 N 个子集,各个训练样本子集独立地在不同的计算机节点上工作,对于每一个计算机节点上的数据都调用相同的生长过程,最终合并得到完整的模型.然而水平并行需要大量的可用内存,因为算法的复制模型需要在本地统计观察,并且花费大量的时间来计算每个属性的信息增益.

3) 在垂直并行中,计算机节点不存储本地模型,每一个逻辑节点只存储分发到的数据的统计信息,并且计算信息增益,最后将聚集本地的统计信息得到最终的决策树.垂直并行更适合于具有很多属性的实例,因为它把大部分的时间开销在每个属性的信息增益的计算上.相比于水平并行,它占用更少的内存,因为它不需要在每一个逻辑节点上复制模型.

因此本文提出基于垂直并行化思想的 P-HT 算法,并行化原理如图 5 所示,每一个方框代表一个处理页(processing item, PI),方框里的数字代表并行度.Model-aggregator PI 包含决策树模型,它通过属性流和控制流连接 local-statistic PI,它通过属性流发送叶子 ID、属性值、类别值等属性内容事件至 local-statistic PI,当节点需要分裂时,Model-aggregator PI 通过控制流发送计算内容事件到 local-statistic PI,每一个 local-statistic PI 通过计算所属属性的 $G(\boldsymbol{X}_i)$ 来确定最大和次大属性 x_a, x_b,然后将结果返回给 Model-aggregator PI,当所有的本地信息到达 Model-

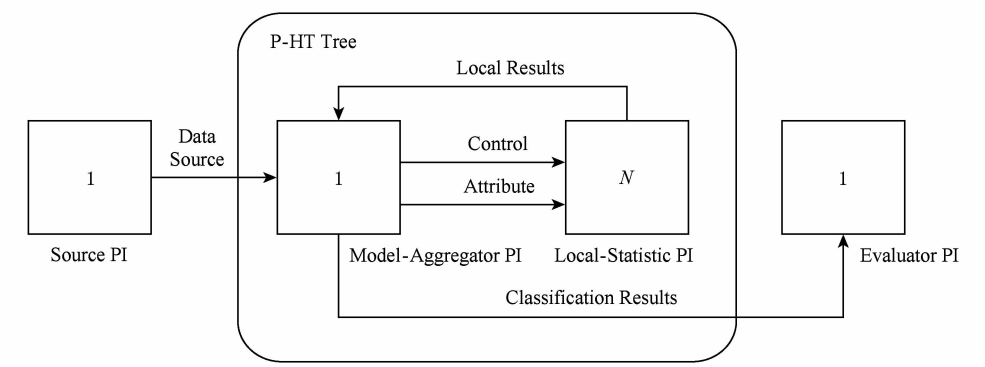

Fig. 5 Vertical parallelization of P-HT

图 5 P-HT 垂直并行化原理

aggregator PI时,会通过计算 Hoeffding 不等式来决定是否进行分裂并将决策结果发送给 evaluator PI 或者其目的 PI.

2.3 P-HT 树生长过程

定义 3. 统计量 N_{ijk}. P-HT 树的每个内部节点都对经过的样本信息进行统计,其中 i 代表属性,j 代表属性取值,k 代表类别. N_{ijk} 表示该节点对于每个属性 i 的每个属性值 j,属于 k 类的样本的个数.

定义 4. 训练数据流. 设 D 为数据流训练样本的集合,所有的样本数据集被划分成 m 个类$\{C_1, C_2, \cdots, C_m\}$,$m$ 为正整数. 又设每个样本用一个 d 维属性向量 $\boldsymbol{X}_i=(x_{1i}, x_{2i}, \cdots, x_{di})$和一个类标号 y_i 表示,其中 $x_{1i}, x_{2i}, \cdots, x_{di}$ 是第 i 个样本在 d 个属性 $A_1, A_2, \cdots, A_d$ 上的值. 令 $N=|D|$ 表示 D 中样本数据集的成员个数.

P-HT 算法首先初始化一个树根节点 *root* 和替代子树集,初始化统计量 N_{ijk},初始化窗口 W. 同决策树内部节点一样,替代子树也是 PNode 类型的,算法在每个 P-HT 树内部节点建立时,都会初始化一个替代子树集合 *altNodes*,每当一个节点找到最佳分裂属性时,也就是当最佳属性的$\overline{G_l(x_a)}-\overline{G_l(x_b)}\geqslant\varepsilon$ 或者 $\varepsilon\leqslant\tau$ 且 $\Delta\bar{G}\geqslant\tau/2$ 时,就在节点处为其产生替代子树. 样本不断进入窗口并沿着 P-HT 树的每个节点向下遍历,树的每个内部节点都对其进行划分测试,根据样本的每个属性取值进入不同的分枝,最终到达一个叶子节点. 当叶子节点中的样本数达到判断分裂的最小值时,据统计量 N_{ijk} 并行地计算每个属性的信息增益 $G(\boldsymbol{X}_i)$,选出最佳属性 x_a,对于 x_a 的每一个取值建立新的叶子节点. 随着样本的不算流入,新建立的叶子节点和所有的替代子树都继续相同的生长过程.

每一个训练样本(x, y)进入窗口后,都会记录下它所到达的最大叶子节点的 *ID*,当窗口内样本数达到阈值需要删除旧的样本时,通过减去样本在 P-HT 树中每个 *ID* 小于存储的 *ID* 的节点计数来实现舍弃旧样本及更新模型的目的.

P-HT 算法初始化时会设置一个检测有效性间隔,当样本数达到检测间隔时,对于每一个内部节点,P-HT 统计接下来到达的 m 个测试样本,用它们来比较在此节点下所有替代子树的精度. 如果最佳替代子树对于测试样本拥有更高的分类准确率,则替代子树会取代原先的节点;如果替代子树的精确度在一段时间内没有提高,P-HT 也会删除没有进展的替代树. P-HT 树的完整生长过程伪代码见算法 1:

算法 1. P-HT 树生长过程.

输入:当前决策树 P-HT、样本流 E 的样本(x, y)、信息增益函数$G(x, y)$、预设的置信度 δ、检测增长需要的样本数 $n_{\min}$、用户预设 ties 值 τ、属性并行计算的并行度 P_{num};

输出:实时决策树.

① 初始时 P-HT 是根节点 *root*;
② for 样本流 E 中的每一个训练样本(x, y) do
③ 样本(x, y)沿着 P-HT 从上到下遍历,树的每个内部节点都对其进行划分测试,根据样本的每个属性取值进入不同的分枝,最终到达一个叶子节点;
④ 样本(x, y)经过的每一个节点对于样本统计值 N_{ijk} 加 1;
⑤ 叶子节点中的大多数样本为同一类则认为其类为 l ;
⑥ if $n_l \bmod n_{\min}=0$ 且 l 中的样本类不能由大多数样本类决定 then
⑦ 通过统计值 N_{ijk},基于并行度 P_{num} 将属性合理地分配给多个线程计算属性 $\boldsymbol{X}_i$ 的信息增益函数 $G(\boldsymbol{X}_i)$;
⑧ 记 x_a 为 G 值最大的属性,x_b 为 G 值次大的属性;
⑨ 通过 δ 和 n_l 计算 Hoeffding 值 $\varepsilon=\sqrt{\frac{R^2\ln(1/\delta)}{2n_l}}$;
⑩ if $x_a\neq x_b$ 且$\overline{(G_l(x_a))}-\overline{(G_l(x_b))}>\varepsilon$ 或者 $\varepsilon<\tau$ then
⑪ 对于最佳属性 x_a 的每一个取值建立新的叶子节点,并且叶子节点的属性集合为$\{\boldsymbol{X}-x_a\}$,初始化替代子树集合 $altNodes=\varnothing$,每个替代子树的统计值 $N_{ijk}=0$;
⑫ end if
⑬ end if
⑭ end for
⑮ for 每个新的叶子节点 do
⑯ 递归调用 P-HTGrow 函数;
⑰ end for
⑱ for 替代子树集合中的替代子树 do
⑲ 递归调用 P-HTGrow 函数;
⑳ end for

2.4 基于 Storm 的 P-HT 算法实现

为了使快速决策树算法能够适应 Storm 实时平台的流数据处理,本文提出并实现了在 Storm 平台上实现 P-HT 并行化算法,该算法在保证精确度不低于传统静态 C4.5 分类算法的情况下,有效地提升了算法的执行效率,并且可以根据用户的需求灵活选择滑动窗口的大小和并行度,大大提高算法的适用性和处理速度.

Storm 计算平台提供了 Spout 和 Bolt 编程接口,Spout 组件作为流数据的输入,Bolt 组件作为流数据的处理逻辑.我们利用 Kafka 消息中间件产生实时数据流,在数据准备阶段我们将训练数据集和的属性集先交给一个 AttAllocateSpout 进行读取,并将训练样本集通过 Kafka 模拟成数据流的形式,在算法 Topology 中利用 KafkaSpout 接口接收从 Kafka 消息队列中传来的数据流,传递给 TransformBolt. 在 TransformBolt 中,数据流将从字符串类型被转换成 instacnce 类型,这是 weka 提供的一种数据结构,样本作为 instance 在 Bolt 之间高速传递并建立决策树. P-HTBolt 初始化树根节点 *root* 和替代子树集 $altNodes$、初始化统计量 N_{ijk}、初始化窗口 $\boldsymbol{W}$. ParallelBolt 根据 AttAllocateSpout 分配的并行度,将所有属性均衡分配给 Storm 集群的每个物理节点,根据统计量 N_{ijk} 计算每个属性的信息增益 $G(\boldsymbol{X}_i)$ 并传递给 CheckSplitBolt 进行分裂条件判断,CheckSplitBolt 接收每个属性的 $G(\boldsymbol{X}_i)$,找出最大和次大 $G(\boldsymbol{X}_i)$,并根据 Hoeffding 边界条件执行分裂. 基于 Storm 的 P-HT 算法实现架构图如图 6 所示:

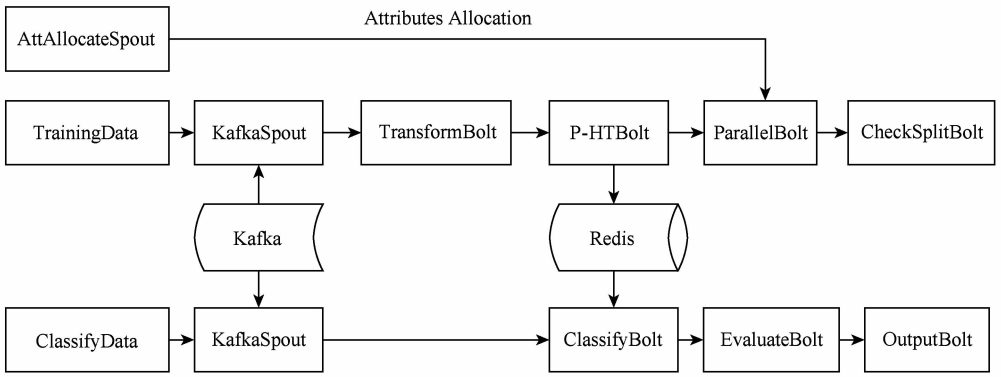

Fig. 6 The architecture of P-HT algorithm based on Storm

图 6 基于 Storm 的 P-HT 算法实现架构

在进行分类时,同样利用 Kafka 消息中间件产生实时数据流,利用 KafkaSpout 接口接收 Kafka 消息队列中传来的待分类样本,传递给 ClassifyBolt 进行分类,ClassifyBolt 从 Redis 数据库中读取实时的分类器对待分类样本进行分类,并将分类结果传递给 EvaluateBolt 进行准确率的计算. OutputBolt 将分类结果和评价结果输出到文件或者数据库中供用户分析.用户可以通过 Redis 数据库配置算法变量需求,可以通过在 Redis 数据库中建立 *key* 值为 *windowsize* 的整数来控制算法中滑动窗口的大小;可以通过建立 *key* 值为 P_{num} 的整数来控制算法的并行度;也可以设置替代子树的检测间隔和检测持续样本数等变量. 算法在 P-HTBolt 中初始化时从 Redis 中读取用户设置的窗口大小 *windowsize* 和并行度 P_{num} 以及检测间隔. 并行度 P_{num} 和 Storm 集群的计算机节点数决定了属性增益的计算所消耗的时间. 基于 Storm 平台的 P-HT 并行化算法伪代码见算法 2:

算法 2. 基于 Storm 的 P-HT 算法.

输入: KafkaSpout 接收的训练数据流 $S=\{\langle \boldsymbol{X}_1, C_1\rangle, \langle \boldsymbol{X}_2, C_2\rangle, \cdots, \langle \boldsymbol{X}_i, C_i\rangle\}$、并行计算的并行度 P_{num}、窗口大小 *windowsize*、检测概念漂移的间隔 *checkinterval*、检测树增长的样本数 $n_{\min}$;

输出: P-HT 决策树.

① TransformBolt 接收 KafkaSpout 传来的数据流,将数据流转换成 Instance 类型的实例子;

② AttAllocateSpout 解析训练集属性向量 ***Attribute***,并根据并行度 P_{num} 平均分配属性;

③ P-HTBolt 实时接收实例,初始化树根节点 *root* 和替代子树集 $altNodes$,初始化统计量 N_{ijk},初始化窗口 $\boldsymbol{W}$;

④ if $n_l \bmod n_{\min}=0$ then

⑤ 将当前决策树 classifier,当前达到分裂条件的节点传给 ParallelBolt;

⑥ ParallelBolt 根据 AttAllocateSpout 分配的并行度,将所有属性均衡分配给 Storm 集群的每个物理节点,根据统计量 N_{ijk} 计算每个属性的信息增益 $G(\boldsymbol{X}_i)$;

⑦ CheckSplitBolt 接收每个属性的 $G(\boldsymbol{X}_i)$,找出最大和次大 $G(\boldsymbol{X}_i)$,并根据 Hoeffding 边界条件执行分裂;

⑧ 对于数据流 S 中的每一条样本$((x,y),ID)$,通过 P-HT 树和所有节点的替代子树进行分类排序;

⑨ ID 为样本所经历的叶子集合中最大的叶子 ID 号,依次添加样本$((x,y),ID)$到窗口中;

⑩ end if

⑪ if 窗口内的样本数 $W'>w$ then

⑫ 释放窗口中的最后一个样本,更新窗口 $\boldsymbol{W}$,

⑬ P-HT 树继续生长;

⑭ end if

⑮ if 如果分裂有效性计数器达到给定的检测间隔 *checkinterval* then

⑯ 检测是否发生概念漂移;

⑰ 递归调用 P-HTGrow 函数;

⑱ end if

2.5 复杂度分析

基于 Storm 平台的 P-HT 并行化分类算法充分利用了 Storm 集群的实时流处理系统优点,同时也利用决策树算法生长的垂直并行化思想,大大提高了算法对流数据的处理效率和算法吞吐量,算法维持的时间滑动窗口 $\boldsymbol{W}$ 提高了模型的实时性,也提升了算法的准确率,同时加入垂直并行计算属性增益的思想,对算法的模型训练时间有了大大的缩减.在基于 Storm 的 P-HT 算法中,算法 2 所需的内存复杂度由决策树中节点的个数 M、属性个数 D、每个属性值的最大值 V 和类的个数 C 决定.传统的串行建树 VFDT 算法的复杂度为 $O(mdvc)$,相比与传统的串行建树的算法 VFDT,若 VFDT 计算 N 个属性的信息增益所需时间为 T,则当属性个数 D 大于并行度 P_{num} 时,在 Storm 集群上进行计算所需的时间为 T/P_{num},即算法 2 所需的内存大小为 $O(mdvc)/P_{\text{num}}$.

3 实验结果与分析

为验证所提出的基于 Storm 的 P-HT 并行化算法在保证算法精度的基础上,大大提高了模型的训练效率.本文采用 2 种数据来测试:1)通过 Storm 的 Spout 读取文件产生数据流,来测试算法在不同 UCI 数据集下的准确率;2)通过 Kafka 模拟真实数据流,分别从算法的并行化效果、抗概念漂移性能分析、算法的吞吐量 3 个方面来分析算法在 Storm 平台上的并行化效果.

3.1 算法的准确率

1) 实验数据

本文在 UCI 机器学习数据集仓库中随机选择 7 个 categorical 类型的分类数据集供 P-HT 算法在 Storm 集群上进行测试,提取数据的属性集和数据集并分别存入集群上相应的文件夹供 Spout 读取.并用数据挖掘软件 weka-3-7 对 C4.5 分类器和 Naive Bayes 分类器进行测试,所使用的 UCI 数据集的基本情况如表 2 所示:

Table 2 Description of UCI Data

表 2 UCI 数据集描述

Number	DataSet	Samples	Attributes	Category
1	Car Evaluation	1 728	6	3
2	Breast_cancer	699	10	2
3	Abalone	4 177	8	5
4	Cmc	1 376	9	2
5	Statlog	1 310	16	7
6	Yeast	1 484	6	4
7	Nursery	12 960	8	3

2) 实验环境

① 软件环境

Storm 版本:apache-storm-0.9.1;

Redis 版本:Redis-2.8.13 64 位;

Java 版本:JDK1.7.0_55.

② 硬件环境

Storm 集群由 master,node1~node4 等 5 个物理节点组成,分别负责 Storm 中拓扑程序的控制和计算工作,其中控制节点为 master,上面运行 Storm 的 Nimbus 进程,计算节点为 node1-4,上面运行 Storm 的 Supervisor 后台进程及算法运算的 Worker 进程.

3) 实验结果

利用 weka-3-7 Explorer 内置的 J48(即 C4.5)分类器和 Naive Bayes 分类器对 UCI 数据集进行测试,同时将 UCI 数据集的属性集和样本集存入 txt 文件,放入 Storm 集群的主节点文件夹中,使用

storm jar 命令将封装好的 VFDT 算法和 P-HT 算法 jar 包提交到 Storm 集群上运行拓扑，这里设置 P-HT 算法的并行度为 4. 通过 PrintBolt 输出的日志文件得到算法的准确率，准确率的计算公式为

$$Accuracy = N_{correct}/N_{total}, \quad (6)$$

其中，$N_{correct}$ 代表被分类器正确分类的样本数，N_{total} 代表参与分类的所有样本数目. 计算结果得出不同数据集的算法准确率实验结果如表 3 所示：

Table 3 Accuracy of the Classifiers

表 3 分类器准确率实验结果 %

DataSet	C4.5	Naive Bayes	VFDT	P-HT
Car Evaluation	77.31	75.69	70.22	75.54
Breast_cancer	68.72	67.51	65.33	68.21
Abalone	78.89	78.52	76.63	78.28
Cmc	81.36	80.85	78.65	80.24
Statlog	65.55	68.38	62.52	64.37
Yeast	79.34	78.67	76.57	78.88
Nursery	83.12	82.56	80.23	83.32
Mushroom	78.21	77.35	75.56	78.11

通过实验结果的对比可以看出，相比传统的静态数据集分类算法 C4.5 和 Naive Bayes，P-HT 算法的精度并没有大幅的下降，基本都在可以接受的误差范围之内，证明传统的决策树分类算法在 Storm 上的实现是有效的；同时在高速的流数据环境下，相比于 VFDT 算法，P-HT 算法的准确率要高，所以替代子树机制和垂直并行化处理机制的应用对基于 Storm 的并行化 P-HT 算法的准确率有明显的提升效果.

3.2 算法的并行化效果和吞吐量

3.2.1 实验数据

测试数据源为 UCI 数据仓库中的数据集 Nursery，数据集属性集描述如表 4 所示：

Table 4 Attributes Description of Nursery Dataset

表 4 Nursery 数据集属性集描述

```
@ relation nursery
@ attribute parents{usual,pretentious,great_pret}
@ attribute has_nurs {proper,less_proper,improper,critical,
  very_crit}
@ attribute form{complete,completed,incomplete,foster }
@ attribute children {1,2,3,more}
@ attribute housing {convenient,less_conv,critical}
@ attribute social {nonprob,slightly_prob,problematic}
@ attribute health {recommended,priority,not_recom}
@ attribute class {not_recom,recommend,very_recom,priority,
  spec_prior}
```

利用 Kafka-2.8.0 产生消息队列模拟实时数据流，分别建立训练数据流和测试数据流 2 个 topic，供建树 Topology 和分类拓扑中的 KafkaSpout 读取. 使用 storm jar 命令将封装好的 P-HT 算法 jar 包提交到 Storm 集群上运行拓扑，这里设置 P-HT 算法的并行度为 4. 利用 Storm UI 观察算法运行的线程压力和吞吐量.

3.2.2 实验环境

1) 本地模式

CPU 2.13 GHz、3.11 GB 内存；

Eclipse Release 4.2.0；

JRE1.7.05_25；

Redis-2.4.5.

2) 集群模式

① 软件环境

Storm 版本：apache-storm-0.9.1；

Kafka 版本：2.8.0-0.8.1.1；

Redis 版本：Redis-2.8.13 64 位；

Java 版本：JDK1.7.0_55；

Zookeeper 版本：Zookeeper-3.4.6.

② 硬件环境

Storm 集群由 master，node1～node4 等 5 个物理节点组成，分别负责 Storm 中拓扑程序的控制和计算工作，其中控制节点为 master，上面运行 Storm 的 Nimbus 进程，计算节点为 node1～node4，上面运行 Storm 的 Supervisor 后台进程及算法运算的 Worker 进程.

3.2.3 算法参数配置

1) 当几个属性的信息熵 G 相同或者差别很小，引入 ties 值，主动选择是否可以分裂形成新的子树，$tieConfidence = 0.05$.

2) 用于计算 Hoeffding bounds 条件的可信度 $\delta = 1E-4$.

3) 初始滑动窗口大小 $windowsize = 200\,000$.

4) 检测节点分裂有效性间隔样本数为 20 000.

5) 检测概念漂移间隔为 1 000.

6) 检测概念漂移时采样的样本数为 1 000.

3.2.4 实验结果

1) 并行化效果分析

P-HT 算法分类拓扑在进行分类时是利用从 Redis 中读取的实时更新的分类器，将 Kafka 发送的待分类数据分配给多个线程进行分布式地打标签，并且在分类结果处理 Bolt 将合并统计所有 Map 线程分类的结果. 直接测量算法每秒处理的数据量

是不准确的,所以本节将控制流入拓扑的数据流速,并且相应改变 P-HT 算法的并行度,通过 Storm 的 UI 界面观测各线程的线程处理压力 *capacity* (*capacity* 等于 Bolt 调用 excute 方法处理的消息数量乘以消息的平均时间除以时间区间). 我们控制每秒 KafkaSpout 发送待分类数据样本 20 000 条. 分别修改分类 Bolt 的线程数为 1,2,4 时各线程的处理压力,并行度分别为 1,2,4 时线程 *capacity* 的测试结果见图 7 所示:

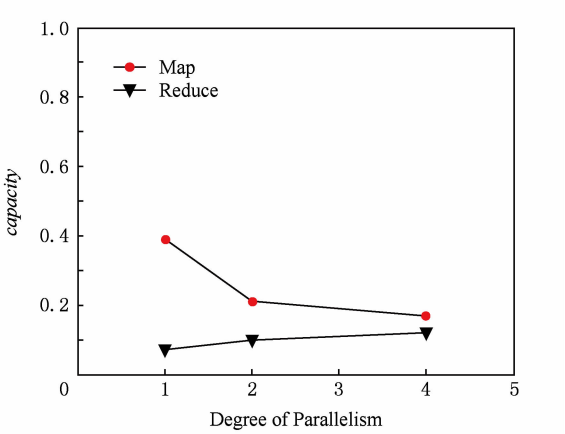

Fig. 7 Relation graph of thread *capacity* and parallelism

图 7 线程 *capacity* 与并行度关系图

由测试结果可以看出,单个 Map 线程的处理压力随着并行度的增加,呈倒数减小趋势;而 Reduce 线程的压力随着 Map 线程的增加,呈近线性增加趋势. 该测试结果与理论分析一致:n 个线程分布式计算与单机模式相比,处理相同数量的数据,单个 Map 线程的处理压力约降为 $1/n$;而由于每个分布式线程发送相同数目的分类结果供合并线程合并,Reduce 线程的压力随着并行度的提高呈线性增加趋势,但 Map 线程和 Reduce 线程的压力均始终维持在 0.4 以下,因此垂直并行化的方法保证了算法的效率.

2) 抗概念漂移性能分析

由于流数据环境下,数据的分布式不稳定,会随时出现概念漂移的情况,本文采用 Nursery 数据集模拟发生概念漂移的数据流,对 VFDT 算法、CVFDT 算法和 P-HT 算法分别进行准确率的测试,实验结果如图 8 所示.

由图 8 可知,数据量达到 40 000 条之前,3 种算法的准确率差别不大,在数据量达到 40 000 条时由于出现了概念漂移,3 种算法的准确率均出现了不同程度的下降,P-HT 算法的准确率突降的幅度是 3 种算法中最小的. 由于采用了替代子树检测概念漂移机制,CVFDT 算法准确率下降的幅度要小于 VFDT 算法,由于 P-HT 算法引入了可变的滑动窗口机制,其准确率不仅下降的幅度最小,同时在准确率突降之后数据流恢复平稳时,P-HT 算法准确率的提升速度也是 3 种算法中最快的.

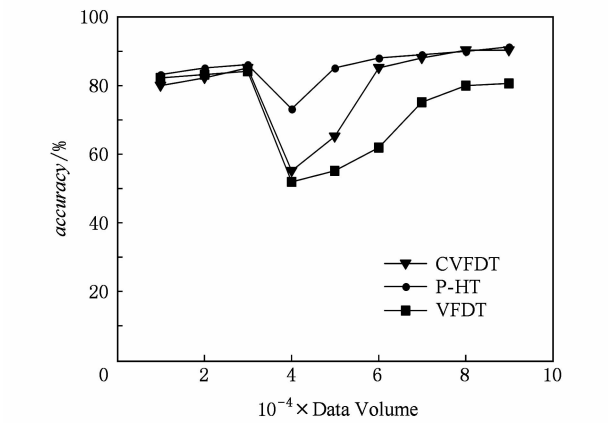

Fig. 8 Performance analysis of anti concept drift of P-HT algorithm

图 8 P-HT 算法抗概念漂移性能分析

3) 吞吐量分析

本文基于决策树的垂直并行化思想对于串行建树的流分类 CVFDT 算法进行了改进,在决策树内部节点的分裂过程中,利用 Storm 集群的特点并行的计算分裂节点的最佳分裂属性,实验将 CVFDT 算法和 P-HT 算法拓扑提交到 Storm 集群上,利用相同的 Nursery 数据集产生数据流,对于 2 个算法处理的数据量进行统计,实验结果如图 9 所示:

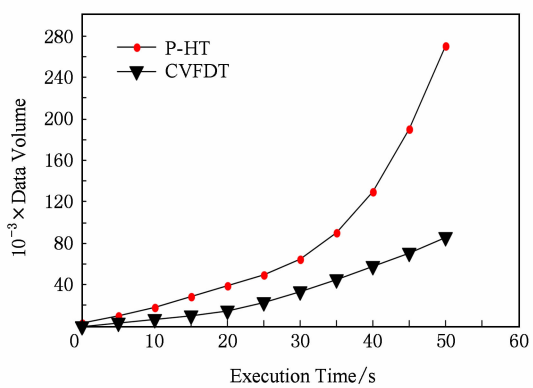

Fig. 9 Comparison of parallel P-HT and serial CVFDT

图 9 并行 P-HT 与串行 CVFDT 吞吐量对比

从图 9 可以看出,在算法的初始化阶段,由于数据量没有达到决策树的生长所需的最小样本数,P-HT 算法和 CVFDT 算法都处于样本流进入窗口阶段,因此串行 CVFDT 算法和并行化 P-HT 算法的

处理数据速度并没有较大的差别;但是随着算法拓扑在 Storm 集群中运行时间的增加,数据量对算法的速率要求开始提高,可以看出并行化 P-HT 算法所处理的数据量要明显大于串行处理方式下的 CVFDT 算法,体现了本文提出的垂直并行化方法对于流数据分类算法的速率有显著的效果.

4 结束语

本文介绍了传统分类算法和 Storm 平台的特点,在深入研究流分类 VFDT 算法和 Storm 平台的基础上,结合 Storm 实时流处理平台的天然优势,提出基于 Storm 平台的流分类 P-HT 并行化算法.该算法引入了时间滑动窗口模型,保证了分类模型的实时性和准确率.同时,算法结合了 Storm 集群的分布式的快速、高效特点,实现了传统决策树算法的并行化建树,提高了算法的训练速度和处理效率,使得传统的决策树分类算法在大数据流环境下得到应用.实验结果表明,基于 Storm 的 P-HT 算法在拥有和离线分类挖掘相当的准确率的同时,比串行的流分类算法拥有更大的处理速度和吞吐量.

参 考 文 献

[1] Suzuki Y, Kido K. Big-data streaming applications scheduling with online learning and concept drift detection [C] //Proc of Design, Automation & Test in Europe. Piscataway, NJ: IEEE, 2015: 1547-1550

[2] Wang Tao, Li Zhoujun, Yan Yuejin, et al. A survey of classification of data streams [J]. Journal of Computer Research and Development, 2007, 44(11): 1809-1815 (in Chinese)
(王涛, 李舟军, 颜跃进, 等. 数据流挖掘分类技术综述[J]. 计算机研究与发展, 2007, 44(11): 1809-1815)

[3] Gaber M M, Zaslavsky A, Krishnaswamy S. Data Stream Mining [G] //Data Mining and Knowledge Discovery Handbook. Berlin: Springer, 2009: 759-787

[4] Quinlan J. Induction of decision trees [J]. Machine Learning, 1986, 1(1): 81-106

[5] Street W N, Kim Y S. A streaming ensemble algorithm (SEA) for large-scale classification [C] //Proc of the 7th ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2001: 377-382

[6] Domingos P, Hulten G. Mining high-speed data streams [C] //Proc of the 6th ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2002: 71-80

[7] Gama J, Rocha R, Medas P. Accurate decision trees for mining high-speed data streams [C] //Proc of the 9th ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2003: 523-528

[8] Gama J, Fernandes R, Rocha R. Decision trees for mining data streams [J]. Intelligent Data Analysis, 2006, 10(1): 23-45

[9] Jin Ruoming. Efficient decision tree construction on streaming data [C] //Proc of the 9th ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2003: 571-576

[10] Anagnostopoulos C, Tasoulis D K, Adams N M, et al. Temporally adaptive estimation of logistic classifiers on data streams [J]. Advances in Data Analysis & Classification, 2009, 3(3): 243-261

[11] Kuncheva L I. Classifier ensembles for changing environments [G] //Multiple Classifier Systems. Berlin: Springer, 2004: 1-15

[12] Gama J. A survey on learning from data streams: Current and future trends [J]. Progress in Artificial Intelligence, 2012, 1(1): 45-55

[13] Hulten G, Spencer L, Domingos P. Mining time-changing data streams [C] //Proc of the ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2001: 97-106

[14] Kuncheva L, Žliobaitė I. On the window size for classification in changing environments [J]. Intelligent Data Analysis, 2009, 13(6): 861-872

[15] Liang Chunquan, Zhang Yang, Shi Peng, et al. Learning very fast decision tree from uncertain data streams with positive and unlabeled samples [J]. Information Sciences, 2012, 213(23): 50-67

[16] Zheng Wenhua. Constructing decision trees for mining high-speed data streams [J]. Chinese Journal of Electronics, 2012, 21(2): 215-220

[17] Quinlan J R. Improved use of continuous attributes in C4.5 [J]. Journal of Artificial Intelligence Research, 1996, 4(1): 77-90

[18] Rahnama A H A. Distributed real-time sentiment analysis for big data social streams [C] //Proc of Int Conf on Control, Decision and Information Technologies (CoDIT). Piscataway, NJ: IEEE, 2014: 789-794

[19] Cal P, Woźniak M. Parallel hoeffding decision tree for streaming data [G] //Distributed Computing and Artificial Intelligence. Berlin: Springer, 2013: 27-35

Ji Yimu, born in 1978. Professor. His main research interests include P2P network optimization, cloud computing security and stream data query in big data.

Zhang Yongpan, born in 1994. Master. His current research interests include stream query and mining in big data.

Zhang Dianchao, born in 1990. Master. His main research interests include the application of big data platform architecture, and computing security.

Lang Xianbo, born in 1991. Master. His current research interest is stream query of join in big data.

Wang Ruchuan, born in 1943. Professor. His main research interests include IoT, cloud computing and big data.