第 14 卷 第 3 期 集 成 技 术 Vol. 14 No. 3
2025 年 5 月 JOURNAL OF INTEGRATION TECHNOLOGY May 2025

引文格式：
梁展雄, 孙旭东, 蔡湧达, 等. 基于 LOGO 计算框架的新型分布式机器学习算法库 [J]. 集成技术, 2025, 14(3): 24-37.
Liang ZX, Sun XD, Cai YD, et al. A new distributed machine learning algorithm library based on LOGO computing framework [J]. Journal of Integration Technology, 2025, 14(3): 24-37.

基于 LOGO 计算框架的新型分布式机器学习算法库

梁展雄[1] 孙旭东[1] 蔡湧达[1] 张育铭[1] 麦朗杰[1] 何玉林[2] 黄哲学[1,2*]

[1](深圳大学计算机与软件学院 深圳 518060)
[2](人工智能与数字经济广东省实验室 (深圳) 深圳 518107)

摘 要 LOGO 是一种“局部-全局 (LOcal-GlObal)”式的分布式计算框架，与流行的 MapReduce 计算框架不同，LOGO 框架下的大数据分布式计算分两步完成：LO 操作在节点虚拟机内运行串行算法，完成一个随机样本块的独立计算，产生局部计算结果；GO 操作将所有局部结果上传到主节点，在主节点内对局部结果做集成，得到大数据的近似计算结果。LOGO 计算框架在执行迭代算法时，消除了节点间的数据通信，极大地提高了分布式计算的效率，降低了内存需求，提高了数据扩展性。本文提出一种新的基于 LOGO 计算框架的分布式机器学习算法库，由 LO 操作执行的串行算法和 GO 操作执行的集成算法两部分组成。LO 操作直接执行已有的机器学习串行算法，无须按 MapReduce 编程模型对算法进行重写，GO 操作对串行计算结果进行集成。本文阐述了 LOGO 分布式计算的原理、算法库架构、串行算法封装和 GO 操作集成策略，展示了 Spark 实现、App 应用开发和多种算法测试结果。

关键词 大数据分布式计算；分布式机器学习算法库；近似计算；非 MapReduce 计算
中图分类号 TP319 **文献标志码** A doi: 10.12146/j.issn.2095-3135.20240224001
CSTR: 32239.14.j.issn.2095-3135.20240224001

A New Distributed Machine Learning Algorithm Library Based on LOGO Computing Framework

LIANG Zhanxiong[1] SUN Xudong[1] CAI Yongda[1] ZHANG Yuming[1] MAI Langjie[1]
HE Yulin[2] HUANG Zhexue[1,2*]

[1](*College of Computer Science and Software Engineering*, *Shenzhen University*, *Shenzhen* 518060, *China*)
[2](*Guangdong Laboratory of Artificial Intelligence and Digital Economy* (*SZ*), *Shenzhen* 518107, *China*)
[*]*Corresponding Author*: *zx.huang@szu.edu.cn*

Abstract Unlike the popular MapReduce computing framework, LOGO is a new distributed computing framework using a LOcal-GlObal computing paradigm. Under the LOGO framework, big data distributed

收稿日期：2024-02-24 **修回日期**：2024-05-09
基金项目：深圳市基础研究重点项目 (JCYJ20220818100205012)；广东省自然科学基金面上项目 (2023A1515011667)；深圳市科技重大专项项目 (KJZD20230923114809020)
作者简介：梁展雄，硕士研究生，研究方向为大数据近似计算； 孙旭东，博士研究生，研究方向为数据挖掘、近似计算、分布式计算和自然语言处理； 蔡湧达，博士研究生，研究方向为机器学习和数据挖掘； 张育铭，硕士研究生，研究方向为数据挖掘和大数据系统计算技术； 麦朗杰，硕士研究生，研究方向为大数据近似计算； 何玉林，博士，研究员，研究方向为大数据系统计算技术和多样本统计分析方法； 黄哲学 (通讯作者)，博士，教授，研究方向为大数据系统计算技术以及面向大数据的数据挖掘和机器学习算法及其应用，E-mail：zx.huang@szu.edu.cn。

computing is completed in two steps. The LO operation runs a serial algorithm in a number of slave nodes or virtual machines to process independently the random sample data blocks, generating local results. The GO operation uploads all local results to the master node and integrate them to obtain the approximate result of the big data set. The LOGO computing framework eliminates data communication between nodes during iterations of the algorithm, greatly improving computing efficiency, reducing memory requirements, and enhancing data scalability. This article proposes a new distributed machine learning algorithm library under the LOGO computing framework. A new distributed computing is divided into two parts: the serial algorithm executed by the LO operation and the ensemble algorithm executed in the GO operation. The LO operation can directly execute existing serial machine learning algorithms without the need to rewrite them according to MapReduce. The GO operation executes ensemble algorithms of different kinds depending on the ensemble tasks. In this article, the principle of LOGO distributed computing is introduced first, followed by the algorithm library structure, the method for packaging existing serial algorithms and the ensemble strategy. Finally, implementation in Spark, App development, and the results of performance tests for various algorithms are demonstrated.

Keywords　big data distributed computing; distributed machine learning algorithm library; approximate computing; non-MapReduce computing

Funding　This work is supported by Key Basic Research Foundation of Shenzhen (JCYJ20220818100205012), Natural Science Foundation of Guangdong Province (2023A1515011667), Science and Technology Major Project of Shenzhen (KJZD20230923114809020)

1　引　言

基于 MapReduce 计算框架的大数据智能分析系统，如 Hadoop MapReduce 和 Spark[1]等，在运行复杂迭代算法进行大数据分析时，存在算法执行效率低、数据扩展能力差和内存瓶颈等卡脖子问题，原因是 MapReduce 计算框架在执行迭代算法时会产生大量的数据通信开销。此外，技术人员需对串行算法按 MapReduce 编程模型进行重写，复杂算法的编程难度大，运行效率低。因此，现有的分布式大数据计算系统普遍存在分析算法匮乏的现象，极大地制约了机器学习方法在大数据分析中的应用。

LOGO 是最近新提出的一种基于 Non-MapReduce 范式的分布式计算框架[2-3]，在随机样本划分 (random sample partition，RSP) 大数据表达模型的支持下[4-5]，采用 LO 和 GO 两步操作完成大数据的分布式计算。LO 操作在节点的虚拟机中运行一个串行算法，对一个 RSP 数据块进行分析，得到大数据的一个局部结果。在这一步，LO 操作可以同时在许多虚拟机中执行同一种算法。集群的计算节点对不同的 RSP 数据块做独立并行计算，得到多个局部结果。GO 操作将局部结果读取到主节点，在主节点运行集成算法，对局部结果做融合操作，得到大数据的集成结果，作为大数据的近似计算结果。

与 MapReduce 相比，LOGO 计算框架在执行迭代算法做分布式计算时，有如下优点：(1) 算法执行效率高；(2) 大数据计算不再受内存限制；(3) 数据扩展能力强；(4) 迭代算法可以在

LO 操作中直接执行，不需按 MapReduce 编程模型重写。这些优点克服了 MapReduce 分布式计算的不足，提高了分布式大数据计算的效率和算法的多样性，丰富了分布式大数据分析系统的功能。

本文提出一种新的基于 LOGO 计算框架的分布式机器学习算法库 RSP-LOGOML。该算法库在开源框架 Spark 上开发，机器学习串行算法采用开源库 Smile 中的算法。为便于算法库的使用、管理、维护和扩展，本文设计开发了算法库的组织架构、算子封装标准、算子的输入和输出数据结构，以及 LO 操作和 GO 操作在应用程序 (APPs) 中的调用格式。这种标准化的算法库设计和开发极大地简化了大数据分析应用的编程复杂度。

本文选择了 7 个数据集对 RSP-LOGOML 算法库中的一些算法做了性能测试，并与 Spark 系统中相同算法的性能做了对比。结果表明，RSP-LOGOML 算法库中的算法运算效率远高于 Spark 中相同算法的运算效率。有监督学习算法的精度高于 Spark 中同类算法的精度，原因是本文采用集成学习策略，而 Spark 算法使用的是单个学习算法。

2 相关工作

分布式大数据计算平台是大数据存储与分析的重要支撑。以谷歌提出的 MapReduce 计算框架和编程模型为核心技术的主流大数据平台在大数据应用中发挥重要作用。但是，随着数据集的增大和分析算法复杂度的增加，现有分布式计算框架在执行迭代算法做分布式大数据分析时遇到了前所未有的挑战[2]，如受限于内存规模和繁重的 I/O、通信开销，从而导致其计算效率低下。近年来，为解决上述问题，某些学者提出了 Non-MapReduce 计算框架，消除了算法迭代过程中节点间的数据通信，提高了计算效率和数据扩展能力。本节对 Non-MapReduce 计算的相关技术进行概述，并对机器学习算法库和集成学习的集成策略进行简单回顾。

2.1 随机样本划分数据表达模型

Non-MapReduce 计算建立在 RSP 数据表达模型之上[4-6]，它将大数据文件表示成互不相交的随机样本块，称为 RSP 数据块。设 D 是一个大数据集文件，$D_1, D_2, \cdots, D_K$ 是 D 的 K 个 RSP 块，若满足以下数学条件，则称 D_i 是 D 的随机样本：

$$D = \cup_{i=1}^{K} D_i \tag{1}$$

$$D_i \cap D_j = \varnothing, 1 \leqslant i, j \leqslant K, i \neq j \tag{2}$$

$$E[F_{D_i}(x)] = F_D(x), 1 \leqslant i \leqslant K \tag{3}$$

其中，$F_{D_i}(x)$ 为 D_i 的累积分布函数；$F_D(x)$ 为 D 的累积分布函数。

可以看出，每个 RSP 数据块是 D 的一个随机样本，可进行独立计算，并可估计 D 的统计特征。为提高估计值的精度，随机抽取多个 RSP 数据块，独立计算多个估计值，根据中心极限定理，计算统计量的统计值，并作为 D 的估计值。这奠定了 Non-MapReduce 计算的统计基础。为解决分布式大数据文件向 RSP 数据表达模型的高效转换问题，Wei 等[7]提出了两阶段的 RSP 块生成算法。

2.2 LOGO 计算框架

LOGO 计算框架由深圳大学大数据技术与应用研究所开发[2,8]，是在 RSP 数据表达模型的基础上，新提出的一种基于 Non-MapReduce 范式的分布式计算框架。LOGO 计算框架将大数据分布式计算分为两阶段：LO 阶段 (局部阶段) 和 GO 阶段 (全局阶段)。在 LO 阶段，串行的数据分析算法在每个 RSP 分区中独立、并行地执行，该阶段不产生全局的数据交换，各分区产生各自的局部结果。LO 阶段完成后，这些结果通过网络传输至负责 GO 操作的主节点，在 GO 阶段中，该节点执行集成计算过程，得到全局结果。

整个过程中，数据通信从 LO 阶段到 GO 阶段之间只发生一次。在迭代任务中，迭代过程在各 RSP 分区上独立执行，不进行全局迭代。与 MapReduce 计算框架相比，LOGO 框架大大降低了多次迭代过程产生的数据通信开销，因此更适合迭代任务。

LOGO 是面向批处理场景设计的大数据计算框架。由于 Spark 引擎执行批处理任务时表现出较好的计算性能，因此在 Spark 上实现 LOGO 框架。虽然 Flink 同样支持批处理计算[9]，且与 Spark 相比，具有一定的性能优势[10]，但其性能优势更多地体现在对流数据的处理能力上[10-11]。所以，LOGO 计算框架并未基于 Flink 实现。

2.3　机器学习算法库

机器学习算法库是数据分析和挖掘的重要工具，主流的开源机器学习算法库 Weka 和 Scikit-learn 等包含的都是串行算法，在单机系统上运行。尽管它们算法丰富，但受限于单机内存，不能用于大数据分析。

主流的分布式大数据分析系统 Hadoop MapReduce 和 Spark 有自身的分布式机器学习算法库 Mahout[12]和 MLlib[13-14]。此外，Boden 等[15]利用 Spark 和 Flink 中的算子重新编写了一些常见的机器学习算法，这些算法提高了计算效率和数据扩展性。Smart-MLlib[16]采用类似 MapReduce 的 Smart 框架实现了常用的机器学习算法，在计算时减少了中间数据的生成，并使用共享内存降低了数据通信的开销。dislib[17]提供编程注解功能，用户使用这些注解对计算任务进行标记，对任务实现并行化计算，无须进行并行化编程。

上面介绍的这些分布式机器学习算法库 (如 Mahout、MLlib、Smart-MLlib 和 dislib) 的算法在 MapReduce 或者改进的 MapReduce 计算框架上运行，计算效率低，数据扩展能力差。此外，MapReduce 编程的限制导致可用的机器学习算法有限，不能满足大数据分析多样性的需求。

2.4　集成学习

LOGO 计算框架采用集成学习策略，即 LO 操作执行一个机器学习算法，对一组 RSP 数据块做并行计算，生成一组子模型；GO 操作执行一个集成算子，对子模型做集成计算，生成集成结果。

常用的集成学习方法 Bagging[18-19]、Boosting[20-21]、Stacking[22-23]和新的集成方法 Dagging、RandomSubspace[19]采用不同的集成策略计算集成结果。分类算法集成常用投票法，如多数投票、加权投票和 Approval 投票法[22]等。回归算法集成常用平均法，如加权平均和简单平均等[24]。聚类算法集成的传统方法有协关联矩阵法、超图划分法和基于二次互信息法等[25-27]，但是这些方法无法扩展到大数据聚类集成领域中。最近，Mahmud 等[25-26, 28]提出的多随机样本分布式聚类集成方法采用不同的集成策略进行大数据聚类，如 ACEM-GM、MSFC-GS 和簇球模型等，为分布式聚类集成提供了新的解决方案。针对基于 MapReduce 的分布式频繁项集挖掘算法[12, 29]计算效率低和数据扩展性差的问题，Sun 等[8]提出的分布式集成方法获得了近似的频繁项集挖掘结果，减少了通信开销，提高了数据可扩展性。

3　算法库的设计与实现

本章介绍基于 Spark 开源系统设计与实现 RSP-LOGOML 算法库的详细步骤。在 Master-Slave share nothing 系统架构下，LOGO 计算框架如图 1 所示，其中 f 表示机器学习算法，LO (f, RSP) 表示基于 RSP 数据块串行运行机器学习算法 f，GO (f, {LO (f, RSP), LO (f, RSP), LO (f, RSP)}) 表示对局部计算结果进行集成计算得到全局计算结果。分布式大数据以 RSP 数据块形式随机存储在 Slave 节点上。对大数据的分析操作用两步完成：第一步通过 LO 操作在 Slave

节点上对 RSP 数据块做计算，产生局部计算结果；第二步将 Slave 的局部结果上传到 Master 节点，用 GO 操作对局部结果做集成计算，产生最终的集成结果。

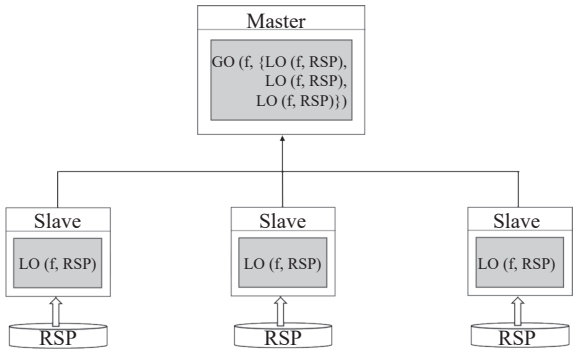

图 1 基于 Master-Slave 架构的 LOGO 计算框架

Fig. 1 LOGO computing framework based on Master-Slave architecture

基于 LOGO 计算框架的 RSP-LOGOML 算法库系统由算法库、算法封装和 Apps 编程 3 部分组成：

(1) 算法库包含所有机器学习算法的源码和可执行代码，源码用 Scala 和 Java 编写，编译成独立的 Jar 包。

(2) 算子库包含所有算法通过标准化封装的算子，在 Apps 中通过 LO 和 GO 操作调用执行，其中封装的标准化包括算子名称、输入和输出数据结构、参数名及其排列顺序等，方便 Apps 开发。

(3) Apps 编程包括 3 个主要组件：输入数据处理、LO 操作算子和 GO 操作集成算子。对于特殊的集成操作，可以在 Apps 中编写集成函数作为集成算子。

RSP-LOGOML 算法库系统的架构如图 2 所示。

3.1 算法库

RSP-LOGOML 算法库可以封装用 Java 或 Scala 编写的任何数据分析和机器学习算法。本文首先选取开源系统 Smile 中的机器学习算法作为首批机器学习算子。根据分析功能划分为分类、回归、聚类、关联规则、特征工程和集成策略等多个种类，如图 3 所示。算法源码在串行算法库中按类别分目录管理，方便代码维护和管理。

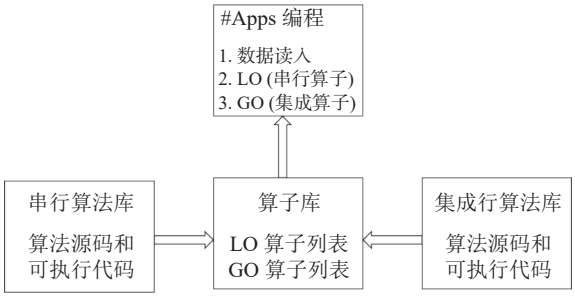

图 2 RSP-LOGOML 算法库架构

Fig. 2 Architecture of RSP-LOGOML algorithm library

RSP-LOGOML 可以灵活增加和删除算法，新算法可以添加到现有类别中，也可以另增新的类别，如自然语言处理算法类别。算法库中的算法代码可以先在外部独立开发和测试，确认无误后再加入算法库，以降低算法的开发和测试难度。

3.2 算法封装

LO 操作可用于通用算子和自定义算法的封装。用户将外部串行算法通过 LO 操作进行封装，形成可直接调用的算子，通过设置合理的参数，就可以进行大数据处理和分析建模。

由于外部算法的源代码不按统一的命名标准和数据结构开发，因此需要对算法做标准化封装，做成便于被外界调用的标准化算子供 LO 操作调用，从而简化 Apps 开发和调试的难度，避免产生执行错误。

如图 4 所示，算法的封装采用固定的标准化模式。首先设计新的算子类，每个算子类包含与输入数据结构相同的算子。例如：分类的输入数据包括独立特征和类标特征。外部的分类代码不一定采用相同的输入数据格式。但是，系统的 RSP 数据块格式是标准的，若某个算法的输入数据格

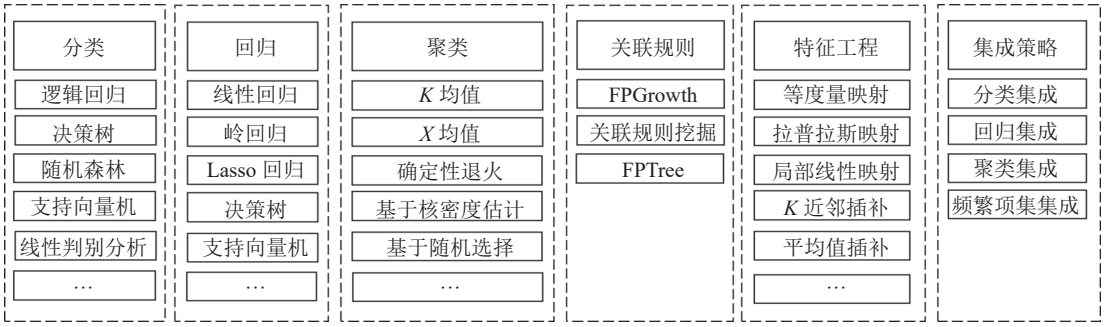

图 3　RSP-LOGOML 算法库的算法分类

Fig. 3　Classification of algorithms in RSP-LOGOML algorithm library

式与 RSP 数据块的格式不同，则需在算子的封装代码中将 RSP 的数据块格式转换成算法源代码的输入数据格式，调用源代码进行运算。同理，若源代码的输出数据格式与系统定义的输出数据格式不同，则需在封装代码中完成数据格式转换。

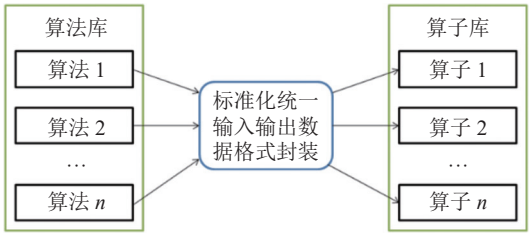

图 4　算法封装示意图

Fig. 4　Diagram of algorithm encapsulation

LOGO 计算框架根据 RSP 数据模型的特点，设计了继承类 RspRDD，并作为分布式数据的标准表达格式。RspRDD 也是 LO 操作的输入数据结构。同样，LO 操作的输出也用 RspRDD。根据算法的输出不同，RspRDD 的分区表达也不同。分区表达的结构由 GO 操作执行的集成算子自动识别，进而完成集成计算。

3.3　Apps 编程

算子封装好后，用户可以用算子库中的算子编写应用程序 Apps，用以对分布式大数据做分析。

在 LOGO 计算框架下，Apps 的开发主要由 3 部分组成：数据预处理、LO 操作调用串行算子和 GO 操作调用集成算子。Apps 的编程流程如图 5 所示。

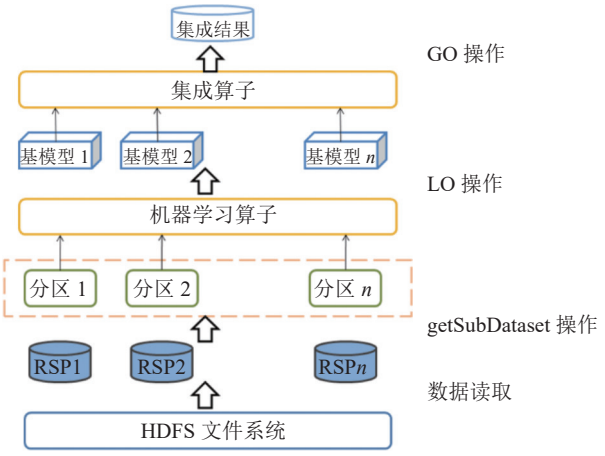

图 5　Apps 流程示意图

Fig. 5　Diagram of Apps procedure

本节基于一个简单应用程序的样例介绍 LO 操作和 GO 操作在 LOGO 计算框架下的运行原理。

3.3.1　getSubDataset 操作

getSubDataset 操作用于从 RSP 块组成的 HDFS 数据中采样，这些数据最初以 HDFS 数据块的形式存储在 HDFS 文件系统中，随后通过 RSP 转换算法转换为 RSP 数据块，并作为 HDFS 文件存储在 HDFS 文件系统中。该操作需要指定参数 Num，指随机从 HDFS 文件中读取 Num 个 RSP 块作为 RspRDD 的分区，用于后续的 LOGO 建模。

3.3.2　LO 操作

LO 操作用来向集群的虚拟机分发一个串行算法及相应参数值，算法的输入数据通过 RspRDD 的分区传递。

当一个分布式大数据读入 RDD 后，首先通过数据预处理转换成 RspRDD 的标准格式，这一过程用 Spark 提供的算子完成。LO 操作执行算子的输入数据通过 RspRDD 的分区自动传入。Apps 开发者仅需设定算子的参数值即可，可以直接选用默认的参数值。

LO 操作是串行算法进行并行计算的桥梁。计算过程中，Master 节点将 LO 操作中的任务分发到各个 Slave 节点的虚拟机，算法在多个虚拟机中独立并行地执行。在这个过程中，算法在各虚拟机内迭代处理各自的 RSP 样本数据块，得到各自的建模结果。每个结果作为 RspRDD 的一个分区存储起来，待 GO 操作处理。

3.3.3 GO 操作

GO 操作作为集成算子，对 LO 操作输出的 RspRDD 的分区结果执行模型的集成操作，输出集成结果。对不同类型的算法，RSP-LOGOML 算法库内置了一些集成算子。例如：对有监督学习的分类算子，提供了多数投票集成算子，基于特定的评估指标优选基模型集成算子[16]；对有监督学习的回归算法，提供了回归模型的均值集成算子；对无监督学习的 *K*-均值聚类算法，提供了基于簇中心平均的集成算子[17]；对分布式频繁项集计算，提供了多数投票集成算子[8]。

除了算法库内置集成算子外，还可以在应用程序中编写新的集成算子函数，然后在 GO 操作中执行集成算子函数，生成集成结果。用户可以通过集成算子函数灵活地实现不同的集成方法。

3.3.4 Apps 程序例子

一个简单的应用程序例子如表 1 所示。第 1 行将指定路径的数据集读取为 RspRDD；第 2 行从数据集中随机采样 *m* 个 RSP 块，用于后续流程；第 3 行将采样的数据按标准数据处理流程转换，得到的 transactionRDD 中的数据格式是输入数组的形式，并使用算法库中的 logoFPGrowth 算法得到多个局部频繁项集结果；第 4 行使用频繁项集的集成方法得到最后的全局频繁项集结果。

表 1 RSP-LOGOML 应用程序示例

Table 1 Example of RSP-LOGOML application

序号	代码展示
1	valinputDataset＝spark.rspRead.parquet("datas/....")
2	valtransactions＝inputDataset.getSubDataset(*m*)
3	vallocalTable＝transactionsRDD. LO(trainDF＝> logoFPGrowth(support,trainDF))
4	valgoTable＝localTable.GO(mergeFPGrowth)

4 实验验证

本实验采用 7 个数据集对 RSP-LOGOML 算法库中的分类、聚类、回归和频繁项集 4 类典型机器学习算法进行了测试，同时还测试了 Spark MLlib 库中的同类算法和单机运行的 Smile 相同算法。算法性能和计算效率实验结果表明，RSP-LOGOML 算法库算法的性能与 Spark MLlib 库和 Smile 相同算法的性能相似，但计算效率和数据处理能力较优。

4.1 实验设置

数据集：7 个数据集的特征如表 2 所示。HIGGS 和 MNIST-PCA 数据集是包含 2 类和 10 类对象的真实数据，DS1 是包含 2 个类别的人工仿真大数据集，用于展示算法处理大数据的能力，这 3 个数据集用于测试分类和聚类算法。DS2 和 DS3 是人工生成数据，用于测试回归算法。TH2 和 Kaggle 是真实的交易数据集，用于测试频繁项集挖掘算法。

实验设置：选择 RSP-LOGOML 中的 11 个分类算法和 3 个聚类算法对 3 个分类和聚类的数据集进行测试，同时选择 Spark MLlib 中的 4 个分类算法和 1 个聚类算法，Smile 中的 9 个分类算法和 3 个聚类算法进行对比实验。同理，在 RSP-LOGOML 算法库中选择 6 个回归算法，对

表 2　数据集

Table 2　Datasets

数据集	RSP 块大小/MB	特征维度	记录数	测试任务
HIGGS	64	28	11 000 000	分类、聚类
MNIST-PCA	64	86	8 100 000	分类、聚类
DS1	100	20	327 360 000	分类、聚类
DS2	64	20	11 000 000	回归
DS3	64	50	11 000 000	回归
TH2	19	—	26 496 645	频繁项集
Kaggle	4	—	3 214 874	频繁项集

注：—处指该数据集不存在特征维度

表 3　不同算法的参数设置

Table 3　Parameter setting of different algorithms

算法	参数设置值
逻辑回归	maxIter＝500, regParam＝0.1
决策树	maxDepth＝10, nodeSize＝10
随机森林	ntrees＝10, maxDepth＝10, nodeSize＝10
支持向量机	regParam＝0.1, tol＝10^{-6}
线性回归	—
回归树	maxDepth＝10, nodeSize＝10
回归森林	ntrees＝10, maxDepth＝10, nodeSize＝10
K-均值	maxIter＝100, tol＝10^{-4}
FPGrowth	support(Kaggle)＝0.002, support(TH2)＝0.02

注：—处指算法无须设置参数

两个回归数据集进行测试，在 Spark MLlib 算法库中选择 3 个同类回归算法和在 Smile 中选择 4 个回归算法进行对比实验。针对两个交易数据集，在 3 个库中选择 FP-Growth 和 Associaton Rule mining 算法进行测试。

本实验设定的 RSP 样本数据块的大小如表 2 所示，RSP-LOGOML 算法采用数据集 10% 的数据做近似计算，而 MLlib 和 Smile 算法在整个数据集上做计算。由于计算资源限制，Smile 的有些算法不能完成对一些数据集的计算，而相同算法在 LOGO 框架下则不会受到计算资源的限制。

对同一算法采用相同的参数设置，具体参数值如表 3 所示。在频繁项集算法实验中，TH2 数据集上的最小支持度设为 0.02，Kaggle 数据集上的最小支持度设为 0.002，以确保能返回足够的频繁项集。

实验环境：分布式计算在 32 个节点的集群上完成，其中 24 节点的配置为两个 IntelXeonE5-2650 CPU 和 128 GB 内存，每个 CPU 有 16 个核，频率为 2.6 GHz。其余 8 个节点配置为两个 IntelXeonE5-2630 CPU 和 128 GB 内存，每个 CPU 有 12 个核，频率为 2.6 GHz。运行环境为 Spark 2.4.0 和 YARN 3.0.0。运行 Spark MLlib 和 RSP-LOGOML 算法时，配置 20 个 executor，每个 executor 分配 16 GB 内存资源。Smile 中的串行算法则在 Master 节点上运行，每个算法配置 48 GB 内存。

4.2　评估指标

运行时间：本文用 RSP-LOGOML、Spark MLlib 和 Smile 算法的运行时间度量计算效率。RSP-LOGOML 算法的运行时间记为从 LO 操作开始到 GO 操作结束；Spark MLlib 和 Smile 算法的运行时间记为算法开始运行到结束。

性能：分类算法用分类精度 (ACC) 评估性能，表示如下：

$$\mathrm{ACC}=\frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}} \tag{4}$$

其中，TP 为正类正确分类的样本数；FP 为正类错误分类的样本数；TN 为负类正确分类的样本数；FN 为负类错误分类的样本数。

回归算法用线性回归决定系数 R^2 评估性能，计算公式为

$$R^2=1-\frac{\mathrm{SSE}}{\mathrm{SST}} \tag{5}$$

其中，SSE 为回归平方和；SST 为总平方和。

聚类算法用纯度 (Purity) 作为性能度量，表示如下：

$$\text{Purity}=\frac{1}{N}\sum_{k=1}^{K}\max_{i\in\{1,2,\cdots,I\}}|\omega_k\cap c_i| \tag{6}$$

其中，N 为样本总数；I 为真实类别的个数；K 为实际聚类的个数；ω_k 为第 k 个聚类集合，$k\in\{1,2,\cdots,K\}$；c_i 为第 i 个真实类别，$i\in\{1,2,\cdots,I\}$；$|\omega_k\cap c_i|$ 为聚类集合 ω_k 和真实类别 c_i 的交集的样本数量。

由于 RSP-LOGOML 算法库中的频繁项集算法采用近似计算方法，以 Spark MLlib 或 Smile 的算法计算结果为真值，算法性能按 Sun 等[8]提出的分布式集成方法评估，性能评估采用精确度 (Precision) 和召回率 (Recall) 计算如下：

$$\text{Precision}=\frac{\text{TP}}{\text{TP}+\text{FP}} \tag{7}$$

$$\text{Recall}=\frac{\text{TP}}{\text{TP}+\text{FN}} \tag{8}$$

其中，TP 为 RSP-LOGOML 算法计算结果中为真的频繁项集的数量；FP 为真的频繁项集没有出现在 RSP-LOGOML 计算结果的数量；FN 为频繁项集在 RSP-LOGOML 的计算结果中预测为非频繁项集的数量。

4.3 实验结果

4.3.1 分类、聚类实验结果

在 RSP-LOGOML 中选用 11 个分类算法和 3 个聚类算法对 HIGGS 数据集进行测试，结果如表 4 所示。算法名在最左列显示，最后 3 个算法为聚类算法。RSP-LOGOML、Spark MLlib 和 Smile 下面的两列为算法运行时间和性能。先看前 4 个分类算法的运行时间，RSP-LOGOML 算法库的运行时间小于 Spark MLlib 和 Smile 的二分之一。其中，Smile 的支持向量机算法没有产生最后结果。再看算法性能，在 RSP-LOGOML 算法库中，逻辑回归和支持向量机的表现优于 Spark MLlib 同类模型的精度，因为 RSP-LOGOML 算法库产生的是集成模型。在 3 个算法库中，Spark MLlib 和 Smile 算法库的性能结果相近，但 RSP-LOGOML 算法库的模型只用了 10% 的数据，Spark MLlib 和 Smile 算法库用全量

表 4 HIGGS 数据集的实验结果

Table 4 Experimental results on HIGGS dataset

算法	RSP-LOGOML		Spark MLlib		Smile	
	时间/s	性能	时间/s	性能	时间/s	性能
逻辑回归	25.695 7	0.643 8	94.123 0	0.622 7	51.474 2	0.640 9
支持向量机	455.472 1	0.642 8	2 125.017 3	0.593 4	×	×
决策树	15.252 6	0.702 5	111.284 9	0.704 8	120.392 6	0.700 7
随机森林	73.548 3	0.706 1	141.222 0	0.704 8	238.465 3	0.702 5
线性判别分析	16.956 0	0.642 7	—	—	18.374 9	0.644 4
正则判别分析	18.146 0	0.649 7	—	—	33.859 9	0.647 2
二次判别分析	36.568 3	0.645 1	—	—	17.050 2	0.644 4
Fisher 线性判别	17.832 9	0.639 7	—	—	19.438 7	0.636 7
多层感知器神经网络	105.132 8	0.525 4	—	—	2 322.064 8	0.527 9
径向基函数网络	25.826 7	0.527 7	—	—	×	×
自适应提升算法	228.727 9	0.717 0	—	—	13 759.990 0	0.710 6
K-均值	39.949 8	0.500 3	101.547 7	0.501 4	165.618 3	0.501 3
G-均值	165.077 5	0.500 1	—	—	333.999 4	0.501 3
X-均值	67.120 1	0.500 1	—	—	374.466 1	0.501 3

注：×处指算法缺乏足够的资源运行；—处指 Spark MLlib 不存在该算法

数据。

分类算法只能比较 RSP-LOGOML 和 Smile 算法库的结果，可以看出，两个库中的算法模型精度基本相同，除了判别分析算法的计算时间近似之外，在其他时空复杂度高的算法上，RSP-LOGOML 要比 Smile 算法的计算时间短得多。

对于聚类算法，Spark MLlib 中只有 *K*-均值算法做对比。3 个库中的算法聚类性能基本相同，但 RSP-LOGOML 算法的运行时间要比其他两个库中算法短得多。

MNIST-PCA 数据集的测试结果如表 5 所示。由于这个数据集有 10 个类别，无法采用二分类的算法，因此表中只列出 7 个分类算法的结果。由前 3 个算法的结果可知：与 Spark MLlib 和 Smile 算法库的单个模型相比，RSP-LOGOML 算法库的集成模型较优；与 RSP-LOGOML 的决策树算法相比，Spark MLlib 的决策树算法略好，因为 RSP-LOGOML 只用了 10% 的数据。从运算时间上看，RSP-LOGOML 算法耗时要远短于其他两个库的算法。

对于其他分类算法，Smile 的径向基函数网络无法处理 MNIST-PCA 数据集，RSP-LOGOML 算法的模型精度和计算时间都近似或者优于 Smile 的同类算法。

对于聚类算法，Smile 的 *K*-均值、*G*-均值、*X*-均值算法均因为内存资源不足而无法处理 MNIST-PCA 数据集，Spark MLlib 只有 *K*-均值可用，聚类性能与 RSP-LOGOML 的 *K*-均值相似，但运行时间长得多。

人工合成的 DS1 数据集的实验结果如表 6 所示，由于 DS1 数据集很大，Smile 无法完成单机建模，因此，表 6 中没有 Smile 的实验结果。

从运行时间上来看，RSP-LOGOML 算法库比 Spark MLlib 算法库耗时短得多。对比较小的 HIGGS 和 MNIST-PCA 数据集，RSP-LOGOML 算法库在 DS1 数据集上的计算效率优势更显著。在性能上，RSP-LOGOML 算法库的结果与 Spark MLlib 算法库的结果相近。

4.3.2　回归实验结果

6 个回归算法在数据集 DS2 和 DS3 上的实验结果如表 7～表 8 所示。Spark MLlib 只有 3 个回归算法可用。从性能上来看，这 3 个库的回归算法性能相似，但 RSP-LOGOML 算法库只用 10% 的数据建模，而其他两个算法库用全量数据建

表 5　MNIST-PCA 数据集的实验结果

Table 5　Experimental results on MNIST-PCA dataset

算法	RSP-LOGOML		Spark MLlib		Smile	
	时间/s	性能	时间/s	性能	时间/s	性能
逻辑回归	88.820 4	0.862 1	119.111 0	0.801 8	823.621 1	0.862 0
决策树	51.500 7	0.713 0	111.284 9	0.721 6	278.838 7	0.693 2
随机森林	107.395 1	0.833 0	366.528 7	0.816 8	637.472 7	0.787 0
线性判别分析	58.443 0	0.778 1	—	—	100.822 4	0.778 1
Fisher 线性判别	33.103 4	0.800 5	—	—	99.965 3	0.799 5
径向基函数网络	37.763 8	0.499 1	—	—	×	×
自适应提升算法	342.639 4	0.839 4	—	—	20 793.259 0	0.806 5
K-均值	68.432 1	0.529 2	134.407 7	0.533 0	×	×
G-均值	135.249 2	0.523 2	—	—	×	×
X-均值	102.038 2	0.495 1	—	—	×	×

注：×处指算法缺乏足够的资源运行；—处指 Spark MLlib 不存在该算法

表 6 DS1 数据集的实验结果

Table 6 Experimental results on DS1 dataset

算法	RSP-LOGOML		Spark MLlib	
	时间/s	性能	时间/s	性能
逻辑回归	76.367 0	0.913 5	1 038.067 8	0.913 5
支持向量机	2 418.025 3	0.913 8	10 323.559 0	0.913 6
决策树	133.311 5	0.883 2	553.686 7	0.876 1
随机森林	212.409 2	0.885 6	1 076.255 9	0.889 3
K-均值	64.613 0	0.913 5	1 040.570 0	0.913 7

表 7 DS2 数据集的实验结果

Table 7 Experimental results on DS2 dataset

算法	RSP-LOGOML		Spark MLlib		Smile	
	时间/s	性能	时间/s	性能	时间/s	性能
线性回归	16.298 9	0.999 9	94.115 8	0.999 9	41.904 4	0.999 9
回归树	41.140 2	0.820 5	109.396 5	0.798 4	107.513 6	0.765 5
回归森林	46.401 8	0.844 3	141.297 5	0.830 2	202.027 0	0.807 5
支持向量机 (回归)	385.180 3	0.999 0	—	—	49 264.364 4	0.999 3
岭回归	49.656 8	0.999 9	—	—	24.456 7	0.999 9
Lasso 回归	60.837 5	0.999 9	—	—	244.569 1	0.999 9

注：—处指 Spark MLlib 不存在该算法

表 8 DS3 数据集的实验结果

Table 8 Experimental results on DS3 dataset

算法	RSP-LOGOML		Spark MLlib		Smile	
	时间/s	性能	时间/s	性能	时间/s	性能
线性回归	28.411 8	0.999 7	146.859 6	0.999 7	122.338 2	0.999 9
回归树	93.166 3	0.759 7	160.882 2	0.708 4	254.999 1	0.765 5
回归森林	80.567 9	0.782 9	183.982 3	0.756 0	440.295 1	0.726 1
支持向量机 (回归)	2 167.530 0	0.999 3	—	—	131 108.801 5	0.999 5
岭回归	68.559 7	0.999 7	—	—	98.526 1	0.999 7
Lasso 回归	71.288 3	0.999 7	—	—	1 402.935 7	0.999 8

注：—处指 Spark MLlib 不存在该算法

模。在运算时间上，RSP-LOGOML 的耗时更短。由于运算的复杂度高，因此相比其他算法，Smile 的支持向量机和 Lasso 回归在单机上处理 DS2 和 DS3 数据集需要大量时间。

4.3.3 频繁项集挖掘实验结果

用 FP-Growth 和 Associaton Rule Mining 算法对交易数据集 TH2 和 Kaggle 进行测试，TH2 数据集的测试结果如表 9～表 10 所示，性能用 Precision 和 Recall 度量。

首先运行 Spark MLlib 的 FP-Growth、Smile 的 FP-Growth 和 Associaton Rule Mining 算法，计算数据集的频繁项集，并作为真值；然后运行

表 9　TH2 数据集的性能结果

Table 9　Evaluation results on TH2 dataset

算法	Precision	Recall
FP-Growth	0.977 2	100%
Associaton Rule Mining	0.984 5	100%

表 10　TH2 数据集各算法的运行时间

Table 10　Runing time of different algorithms on TH2 dataset

算法	RSP-LOGOML	Spark MLlib	Smile
	时间/s	时间/s	时间/s
FP-Growth	53.831 2	72.515 4	459.632 7
Associaton Rule Mining	48.263 6	—	528.119 9

注：—处指 Spark MLlib 不存在该算法

表 11　Kaggle 数据集的性能结果

Table 11　Evaluation results on Kaggle dataset

算法	Precision	Recall
FP-Growth	0.900 3	0.996 6
Associaton Rule Mining	0.896 2	0.994 9

表 12　Kaggle 数据集各算法的运行时间

Table 12　Runing time of different algorithms on Kaggle dataset

算法	RSP-LOGOML	Spark MLlib	Smile
	时间/s	时间/s	时间/s
FP-Growth	25.925 5	66.295 6	18.477 9
Associaton Rule Mining	26.276 4	—	16.296 3

注：—处指 Spark MLlib 不存在该算法

RSP-LOGOML 的 FP-Growth 和 Associaton Rule Mining 算法，计算近似的频繁项集，利用式 (7)～式 (8) 计算 Precision 和 Recall。两个算法的 Precision 和 Recall 结果如表 9 所示：两个算法都得到 100% 的召回率，说明 RSP-LOGOML 可以得到所有真实的频繁项集；精度均大于 97%，满足实际应用要求。

由表 10 可知，RSP-LOGOML 算法的计算效率比 Spark MLlib 算法稍有优势，比 Smile 单机算法有较大优势。

Kaggle 数据集的实验结果如表 11～表 12 所示。RSP-LOGOML 算法同样有较高的召回率和约 90% 的精度。与 TH2 数据集相比，Kaggle 数据集较小，设置的 RSP 数据块也较小，造成算法精度下降，同时，分布式计算的时间也长于单机计算时间。由此表明：分布式计算在处理大数据时在计算效率上有很大优势，特别是单机无法处理的数据；在小数据集计算上，不需要分布式计算。

4.4　讨论与分析

Smile 串行算法库的优点是包含丰富的算法供数据分析时选用，但数据处理能力受单机计算资源的限制。随着数据集的不断增大，单机串行算法分析模式已经难以满足大数据分析的需求。

分布式并行计算是大数据分析的必要手段。但是，基于谷歌 MapReduce 编程模型和计算框架在进行大数据分布式并行计算时存在如下瓶颈：(1) 由于复杂串行算法难以用 MapReduce 编程模型改写，因此算法库中可用的分布式并行计算算法不足；(2) MapReduce 计算框架在执行迭代算法时，节点间的数据通信开销大导致算法执行效率较低，数据处理能力受内存约束，大数据扩展性差。

LOGO 计算框架从根本上消除了 MapReduce 的计算瓶颈，开辟了串行算法无须改写、直接高效执行分布式并行计算的新模式，极大地丰富了分布式并行计算的算法库。

RSP-LOGOML 算法库是大数据分布式计算的一种新技术。本文实验结果表明，对于较大的数据集来说，RSP-LOGOML 算法库中算法的计算效率远高于同类算法的单机计算效率和 Spark MLlib 中 MapReduce 编程同类算法的计算效率。RSP-LOGOML 算法库中的算法性能等价或略好于 Spark MLlib 中的算法，但可用算法的数量

远大于 Spark MLlib。因此，RSP-LOGOML 算法库为数据分析科学家提供了更好和更多的算法工具。

分布式并行计算的优势在于可处理大规模数据集。对于小数据来说，单机串行算法的计算效率更高，因为分布式并行计算会产生节点间的通信开销。这一点可以从表 12 中 Kaggle 数据集的实验结果看出。因此，单机串行算法库在小数据分析中优势明显，而 RSP-LOGOML 算法库将单机串行算法库移植到分布式计算平台上，为大数据的处理与分析提供了更丰富的算法资源。

5 结 论

本文介绍了 LOGO 分布式计算框架下的机器学习算法库 RSP-LOGOML。LOGO 采用的是 Non-MapReduce 的计算范式，消除了迭代算法运行时节点间的数据通信，在 RSP 分布式数据表达模型支持下，实现了串行算法在 LOGO 框架下的直接分布式运行，丰富了大数据分布式并行计算的算法工具。

RSP-LOGOML 算法库是对大数据分布式并行计算的框架模型创新和分析工具创新。依据的近似计算方法、LOGO 计算框架和 RSP 数据表达模型都是近年出现的原创性成果。未来，本研究团队将采用更大的数据集和真实应用的数据集对 RSP-LOGOML 算法库的性能进行测试，为大数据分析提供更丰富、更高效的算法库。

参 考 文 献

[1] Salloum S, DautovR, Chen XJ, et al. Big data analytics on Apache Spark [J]. International Journal of Data Science and Analytics, 2016, 1(3-4): 145-164.

[2] Sun XD, He YL, Wu DM, et al. Survey of distributed computing frameworks for supporting big data analysis [J]. Big Data Mining and Analytics, 2023, 6(2): 154-169.

[3] Sun XD, Zhao LX, Chen JQ, et al. Non-MapReduce computing for intelligent big data analysis [J]. Engineering Applications of Artificial Intelligence, 2024, 129: 107648.

[4] Salloum S, Huang JZ, He YL. Random sample partition: a distributed data model for big data analysis [J]. IEEE Transactions on Industrial Informatics, 2019, 15(11): 5846-5854.

[5] 黄哲学, 何玉林, 魏丞昊, 等. 大数据随机样本划分模型及相关分析计算技术 [J]. 数据采集与处理, 2019, 34(3): 373-385.
Huang ZX, He YL, Wei CH, et al. Random sample partition data model and related technologies for big data analysis [J]. Journal of Data Acquisition & Processing, 2019, 34(3): 373-385.

[6] Mahmud MS, Huang JZ, Salloum S, et al. A survey of data partitioning and sampling methods to support big data analysis [J]. Big Data Mining and Analytics, 2020, 3(2): 85-101.

[7] Wei CH, Salloum S, Emara TZ, et al. A two-stage data processing algorithm to generate random sample partitions for big data analysis [C] // Proceedings of the 2018 International Conference of Cloud Computing, 2018: 347-364.

[8] Sun XD, Ngueilbaye A, Luo KJ, et al. A scalable and flexible basket analysis system for big transaction data in Spark [J]. Information Processing & Management, 2024, 61(2): 103577.

[9] Thakur BS, Bansal KL. Performance evaluation of Apache Hadoop, Apache Spark, and Apache Flink [J]. Advances in Management, Social Sciences and Technology, 2021: 93-100.

[10] Sharma A, Puri D, Kumar M, et al. Implementation and comparison of big data analysis on large dataset using Spark and Flink [C] // Proceedings of the 4th International Conference on Communications and Cyber Physical Engineering, 2022: 385-394.

[11] Hueske F, Kalavri V. Stream processing with Apache Flink: fundamentals, implementation, and operation of streaming applications [M]. United States of America: O'Reilly Media, 2019.

[12] Luna JM, Padillo F, Pechenizkiy M, et al. Apriori-versions based on MapReduce for mining frequent patterns on big data [J]. IEEE Transactions on Cybernetics, 2018, 48(10): 2851-2865.

[13] Assefi M, Behravesh E, Liu GC, et al. Big data machine learning using Apache Spark MLlib [C] // Proceedings of the 2017 IEEE International Conference on Big Data, 2017: 3492-3498.

[14] Meng XR, Bradley J, Yavuz B, et al. MLlib: machine learning in Apache Spark [J]. Journal of Machine Learning Research, 2016, 17(34): 1235-1241.

[15] Boden C, Spina A, Rabl T, et al. Benchmarking data flow systems for scalable machine learning [C] // Proceedings of the 4th ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond, 2017: 1-10.

[16] Siegal D, Guo J, Agrawal G. Smart-MLlib: a high-performance machine-learning library [C] // Proceedings of the 2016 IEEE International Conference on Cluster Computing, 2016: 336-345.

[17] Cid-Fuentes JA, Sola S, Alvarez P, et al. dislib: large scale high performance machine learning in Python [C] // Proceedings of the 15th International Conference on eScience, 2019: 96-105.

[18] 徐继伟, 杨云. 集成学习方法: 研究综述 [J]. 云南大学学报 (自然科学版), 2018, 40(6): 1082-1092.
Xu JW, Yang Y. A survey of ensemble learning approaches [J]. Journal of Yunnan University (Natural Sciences Edition), 2018, 40(6): 1082-1092.

[19] 张春霞, 张讲社. 选择性集成学习算法综述 [J]. 计算机学报, 2011, 34(8): 1399-1410.
Zhang CX, Zhang JS. A survey of selective ensemble learning algorithms [J]. Chinese Journal of Computers, 2011, 34(8): 1399-1410.

[20] 吉根林, 凌霄汉, 杨明. 一种基于集成学习的分布式聚类算法 [J]. 东南大学学报 (自然科学版), 2007, 37(4): 585-588.
Ji GL, Ling XH, Yang M. Distributed clustering algorithm based on ensemble learning [J]. Journal of Southeast University (Natural Science Edition), 2007, 37(4): 585-588.

[21] Diwakar T, Kumar SA, Ramachandra RB, et al. Credit scoring models using ensemble learning and classification approaches: a comprehensive survey [J]. Wireless Personal Communications, 2022, 123(1): 785-812.

[22] Mienye ID, Sun YX. A survey of ensemble learning: concepts, algorithms, applications, and prospects [J]. IEEE Access, 2022, 10: 99129-99149.

[23] Dong XB, Yu ZW, Cao WM, et al. A survey on ensemble learning [J]. Frontiers of Computer Science, 2020, 14: 241-258.

[24] Campagner A, Ciucci D, Cabitza F. Aggregation models in ensemble learning: a large-scale comparison [J]. Information Fusion, 2023, 90: 241-252.

[25] Mahmud MS, Huang JZ, García S. Clustering approximation via a fusion of multiple random samples [J]. Information Fusion, 2024, 101: 101986.

[26] Mahmud MS, Huang JZ, Ruby R, et al. Approximate clustering ensemble method for big data [J]. IEEE Transactions on Big Data, 2023, 9(4): 1142-1155.

[27] Minaei-Bidgoli B, Topchy A, Punch WF. A comparison of resampling methods for clustering ensembles [C] // Proceedings of the 2004 International Conference on Artificial Intelligence, 2004: 939-945.

[28] Mahmud MS, Huang JZ, Ruby R, et al. An ensemble method for estimating the number of clusters in a big data set using multiple random samples [J]. Journal of Big Data, 2023, 10(1): 40.

[29] Xiao W, Hu J. Paradigm and performance analysis of distributed frequent itemset mining algorithms based on MapReduce [J]. Microprocessors and Microsystems, 2021, 82: 103817.