BIG DATA MINING AND ANYLISTIC
ISSN 1007-0214 0?/?? pp???-???
Volume XX, Number x, December xxxx

# LogoML: an Open Machine Learning Library for Distributed Big Data Analytics

Xudong Sun,Yongda Cai, Yi Tao, Langjie Mai, Joshua Zhexue Huang*

**Abstract:** Implementation of distributed machine learning algorithms in MapReduce faces three challenges: low execution efficiency of iterative algorithms, poor data scalability, and high complexity in decomposing a complicated algorithm into the rigid structure of Map and Reduce operations. The last challenge makes many useful algorithms for big data analysis unavailable in current distributed systems. In this paper, we propose LogoML, a library of machine learning algorithms for distributed big data analysis. LogoML is implemented on top of the LOGO distributed computing framework and the RSP data representation model. Unlike MapReduce algorithms, the algorithms in LogoML are sequential in nature. With LogoML, a data analysis task is carried out with two core operations in the LOGO distributed computing framework. The LO operation executes a data analytic algorithm in multiple nodes or virtual machines in parallel to process a set of random sample data blocks and generate a set of local results. The GO operation executes an ensemble algorithm to aggregate the local results into the final ensemble result. Through innovative design and implementation, the LogoML library exhibits the following merits in distributed big data analysis: It has rich algorithms because sequential algorithms are easy to implement in LogoML; It is efficient in execution of iterative algorithms and scalable to big data; Its open structure allows the user to add new algorithms to the library. The performance, efficiency, and data scalability of LogoML algorithms are demonstrated with the experiment results of both synthetic and real data sets.

**Keywords:** Machine learning library; Big data analysis; Approximate computing; Distributed computing

## 1 Introduction

As industries pursue intelligence in business operations to increase efficiency, cut costs, and reduce errors, data integration, pre-processing, analysis and model building are becoming compulsory business processes in modern enterprises[1, 2]. To deal with large and complex business data, distributed computing platforms, which enable big data analytics, play a key role in facilitating data-driven intelligent business operations[3]. Good examples include Hadoop*[4], Spark†[5], Flink[6], and Storm[7], which are widely used in many industry sectors.

An important requirement for a distributed data analysis platform in industry is a rich set of algorithms that are easy to use, efficient in processing data, scalable to big data, and can be used in various analytical tasks, such as data integration, pre-processing, feature engineering, model building, and visualization[8]. Although traditional data analysis systems[9], such as R, Matlab, Weka, Smile, and Sklearn, provide

• Xudong Sun is with College of Management, Shenzhen University, Shenzhen 518060, China. Yongda Cai is with College of Computer Science, Guangdong Polytechnic Normal University, Guangzhou 510665, China; Yi Tao, Langjie Mai and Joshua Zhexue Huang are with College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China. Joshua Zhexue Huang is also with Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ), Shenzhen 518107, China. E-mail: sunxudong2016@email.szu.edu.cn;caiyongda@gpnu.edu.cn; 2400101102@mails.szu.edu.cn; 2310273084@email.szu.edu.cn; zx.huang@szu.edu.cn.
∗ To whom correspondence should be addressed.
Manuscript received: 2025-06-30; revised: 2025-07-27; accepted: 2025-09-19

*https://hadoop.apache.org/
†https://spark.apache.org/

many algorithms for data processing and analysis, they cannot handle large data sets because the algorithms are sequential implementations and cannot run on parallel and distributed computing environments.

Distributed systems for big data analytics, e.g., Hadoop MapReduce[10] and Spark, also provide algorithm libraries, such as Mahout and MLlib, to facilitate implementations of data-analysis processes. However, unlike the rich algorithms in traditional data analysis systems, the data analytic algorithms currently available in these systems are very limited. For example, the Mahout[11] library for Hadoop only provides 27 machine learning algorithms, whereas MLlib[12] in Spark has 48 algorithms for classification, regression, clustering, and frequent pattern mining. Many algorithms necessary in data analytics, such as data exploration, data integration, data pre-processing, feature engineering, NLP, and visualization, are missing from the libraries. The inadequacy of necessary algorithms handicaps the use of these systems in diverse applications and increases the costs of developing application systems.

The paramount reason for the lack of necessary analytic algorithms in distributed systems is the difficulty of implementing distributed algorithms that can run efficiently on distributed big data sets. The MapReduce programming model is used to write distributed algorithms for processing big data on a cluster or cloud. However, implementing distributed algorithms in MapReduce has two problems: low execution efficiency for iterative algorithms and poor data scalability[13]. Another problem is the high complexity of decomposing a complicated algorithm into the rigid structure of Map and Reduce operations, which makes complex algorithms difficult to implement.

In this paper, we present LogoML, an open machine learning library for distributed big data analytics. LogoML is developed on two new underlying technologies: the random sample partition (RSP)[14, 15] data model and the distributed computing framework LOGO(Local Operation and Global Operation). The RSP data model represents a distributed data file as a set of RSP data block files, each being a random sample of the data file and analyzed independently[16]. The LOGO computing framework uses a non-MapReduce[17] computing paradigm to perform a data analytic task in two operations. The LO operation executes an analytic algorithm on an RSP data block to produce a local estimate of the big data file. In this stage, multiple RSP data blocks are processed by the same algorithm in parallel and independently on virtual machines. After the LO operation, all local estimates are transferred to the master node and aggregated to the final result with an ensemble algorithm executed by the GO operation. Since data communication between nodes is removed in the LO operation, the algorithms in LogoML can be efficiently executed on a cluster over a large distributed data file.

Built on the RSP and LOGO technologies, the LogoML algorithm library has the following characteristics: (1) Instead of running on the whole distributed data file, the LO operation executes the analytic algorithm on a random set of RSP data blocks in parallel, and each RSP data block is independently processed on a virtual machine without the need of data communication among the nodes. Therefore, execution is very efficient if the algorithm is iterative, and it is unnecessary to rewrite the sequential algorithm into a distributed version. (2) The GO operation executes an ensemble algorithm to aggregate the results of the RSP data blocks into the final result. Since it only requires one transformation of the local results from the slave nodes to the master node, and the ensemble algorithm does not have a heavy iterative process and a lot of data to compute the final result, the GO operation is also efficient to execute on the master node. (3) The LogoML library provides an open architecture that allows new algorithms to be easily added to the library.

The innovative design and implementation of the LogoML library address the problems of distributed analysis of extremely large data sets of many terabytes, which are frequently encountered in business sectors. Existing systems for distributed big data analysis are inadequate for this type of analysis task. We have tested the algorithms in LogoML on both real and synthetic data sets, and compared the results of LogoML algorithms with those of Spark distributed algorithms. The results demonstrate that LogoML algorithms produced more accurate results, were much more efficient than Spark distributed algorithms, and were scalable to big data sets.

The remaining sections are organized as follows. Section 2 presents related work. Section 3 introduces the underlying technology of the LOGO distributed computing framework. The LogoML library is proposed in Section 4. Section 5 presents the evaluation results of the LogoML algorithms. Section 6 concludes this paper.

## 2 Related Work

In this section, we review a set of libraries and systems that provide ready-to-use data analytic and machine learning algorithms to facilitate the implementation of data analytical processes in real applications. We discuss the advantages and shortcomings of the libraries in both sequential and parallel implementations.

### 2.1 Machine learning libraries of sequential implementations

Libraries containing ready-to-use algorithms can facilitate the implementation of data analysis processes and machine learning tasks and reduce application development costs[18]. There are a number of libraries of sequential algorithms available for data analysis. The most popular is the R system, which contains a rich set of algorithms for statistical analysis of data and machine learning. R is easy to use, but cannot handle big data[19].

Recently, Python has become popular in data analytics and machine learning. Quite a number of packages containing data analysis algorithms have been developed in Python[20], such as Scikit-learn[21], XGBoost[22], River[23], Darts[24], etc. These packages provide easy-to-use data analytic functions for users to perform data analysis and model-building tasks. There are also Python packages for deep learning algorithms, such as Pytorch[25], Theano[26], and Mxnet[27], which are widely used in the research community and industry.

Smile[‡] library contains a large number of algorithms for classification, regression, dimensionality reduction, data pre-processing, etc. Written in multiple languages, these algorithms can be easily integrated into applications developed in different languages. Data mining software tools, such as RapidMiner and Weka, also contain a rich set of machine learning algorithms. These tools provide integrated environments built on top of industry-standard programming languages to simplify the complex data analysis process.

The above-mentioned libraries of data analytic algorithms are all open-source packages widely accessible at repositories on GitHub. They are easy to use, but they were not developed for a distributed computing environment. Therefore, they cannot handle large distributed data sets.

### 2.2 Machine learning libraries of distributed implementations

Compared to the rich set of algorithms in the sequential libraries, the libraries containing data analytic and machine learning algorithms in parallel and distributed implementations are less. In this section, we review some of them that are widely used in the analysis of distributed data[28, 29].

Mahout[11] is the earliest distributed machine learning library running on the Hadoop platform. The algorithms in Mahout are written in the MapReduce programming model. Although Mahout is well known in the research community and industry, the applications of Mahout are hindered by the inherent weaknesses of MapReduce as follows: complexity in programming a complicated algorithm in MapReduce, e.g., graph algorithms and neural networks; low efficiency and poor data scalability in executing iterative MapReduce algorithms because of the heavy I/O overhead and data communication among the nodes. Haloop[30] and Mahout Samsara[31] proposed revisions to the executions of MapReduce algorithms to improve computational performance, but the inherent overhead in data communication has not been removed completely.

MLlib[12] is an open source distributed machine learning library in Apache Spark. It provides tools including machine learning algorithms, data featurization, operation pipelines, data persistence, and utilities to facilitate big data computation[32]. The advantage of MLlib is that it uses Resilient Distributed Dataset (RDD)[33] to execute iterative algorithms so that heavy I/O overhead is removed. However, the overhead of data communication still exists and poor data scalability remains.

FlinkML[34] is an open source machine learning library in Apache Flink, a distributed system for processing data streams. Compared to Mahout and MLlib, FlinkML has limited algorithms, low efficiency, and poor data scalability in the execution of iterative algorithms. In addition, Microsoft and Yahoo jointly developed the lightweight distributed online learning library, Vowpal Wabbit[35], which primarily targets at experienced developers and imposes strict constraints on data formats. Uber has open-sourced the Horovod framework[§], which is based on the AllReduce paradigm; however, it is predominantly used for multi-

[‡]haifengl.github.io

[§]github.com/horovod/

GPU training scenarios.

### 2.3 Inadequacies of current distributed machine learning libraries

Distributed machine learning libraries play a key role in the development of distributed applications of data analysis and decision making. However, in dealing with large data sets and complex applications, they have the following shortcomings.

- **Limited algorithms available.** The latest version of Spark MLlib has only about 50 machine learning and data analytic algorithms. This represents the richest distributed algorithm library currently available. In this library, classical algorithms, such as K-Nearest Neighbors and Neural Networks, are missing. In contrast, scikit-learn and Smile include more than 150 machine learning algorithms.

- **Low efficiency in iterative computation.** Machine learning algorithms for model training are iterative in nature. Current distributed algorithms for model training are implemented in the MapReduce computing paradigm. Execution of these algorithms on clusters or cloud is not efficient due to the heavy I/O and data communication overhead[36].

- **Poor data scalability.** The distributed machine learning algorithms rely on in-memory computing to achieve efficiency[37]. The available memory in distributed systems limits the data scale to be computed.

- **Complexity of programming in MapReduce.** To write an algorithm in MapReduce, it requires decomposing the computation process into a sequence of pairs of Map and Reduce operations. However, for complicated algorithms, e.g., graph computing and neural networks, it is either difficult or impossible to carry out the decomposition. Therefore, it is hard to implement computationally complex algorithms in MapReduce running on distributed platforms.

In the following sections, we present a new machine learning and data analytical algorithm library that removes the above shortcomings in distributed big data analysis.

## 3 Overview of LOGO Distributed Computing Framework

The LOGO distributed computing framework is an integrated implementation of the Non-MapReduce[17] paradigm and the RSP data representation model to speed up computing efficiency and improve data scalability in distributed big data analysis and machine learning. The RSP data model represents a distributed data file as a set of RSP data blocks, each being a random sample of the distributed data file so it can be analyzed independently of other RSP data blocks to produce an estimate result of the entire data file[14]. Based on the RSP data representation, the Non-MapReduce paradigm uses two steps to perform a data analysis task. The first step is to use an analytic algorithm to process an RSP data block and generate a result as an estimate. In this step, multiple RSP data blocks are selected randomly from the RSP data representation and analyzed by the same algorithm in parallel to generate multiple estimates. The second step ensembles the multiple estimates to the final result as the result of the whole data file[38, 39]. Compared to MapReduce, the Non-MapReduce distributed computing paradigm is much more efficient and scalable to big data in executing iterative data analytic algorithms because the first step processes multiple RSP data blocks with a sequential algorithm in parallel without requirements of data communication among the nodes and big memory for processing each RSP data block.

Fig.1 shows the system architecture for the implementation of the LOGO distributed computing framework. A distributed data file is represented as a set of RSP data blocks managed with HDFS[40]. InputRDD is an extension to Spark RDD to store RSP data blocks in memory with partitions of InputRDD. The executors are computing units that process the RSP data blocks in parallel. The results of the RSP data blocks are loaded to the master node and ensembled in it to generate the final result. The LOGO computing process is scheduled by the DAGScheduler and Task Scheduler to generate the final result efficiently. The DAGScheduler is responsible for transforming computing tasks into a DAG, which determines the sequence of task distribution. The TaskScheduler is responsible for distributing tasks, and this process is based on the principle of data locality—ensuring that the data location (data partition) is as close as possible

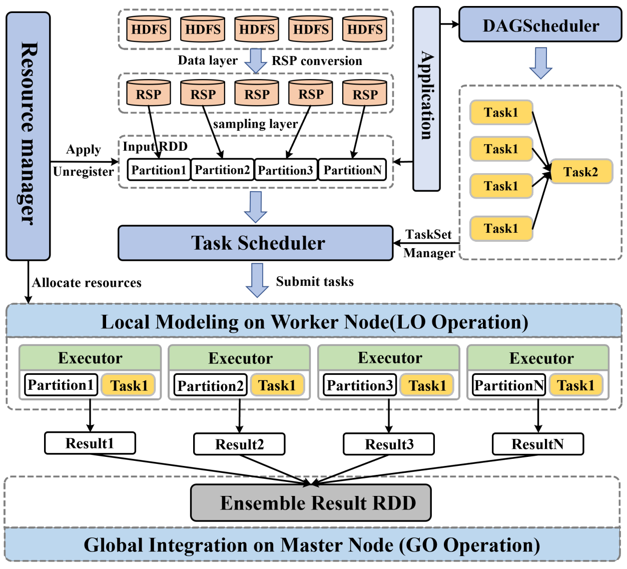


**Fig. 1** System architecture for implementation of the LOGO distributed computing framework.

to the computing unit (executor).

At the programming level, two general operations are defined to carry out the distributed computing process on the LOGO computing framework. The LO operation is defined to execute a sequential algorithm in executors in parallel to process the partitions of InputRDD storing multiple RSP data blocks and produce the local results of RSP data blocks[41].It is worth noting that this serial function is not limited to machine learning algorithms; it can also be any functional operation, such as data prediction, feature engineering, or even model prediction and application. The GO operation is used to execute an ensemble algorithm on the master node to integrate the local results from the LO operations into the final result. RDD is used as data formats for inputs and outputs of the LO and GO operations.

At the execution level, when a computing task, described in the analytic algorithm carried out by the LO operation and the ensemble algorithm by the GO operation, is submitted to LOGO, the DAGScheduler, triggered by an action operator, constructs the corresponding DAG based on the task content. Unlike Spark, in the LOGO computing framework, every application task generates only a single stage, that is, a single TaskSet. This design implies that identical tasks are executed in parallel and in one round on slave nodes. After the DAG is encapsulated by the TaskSet Manager, it is handed over to the TaskScheduler, which is responsible for assigning the subtasks contained in the TaskSet. For each task assigned to a computing node, the Resource Manager loads one data partition from the InputRDD and assigns a corresponding subtask, indicating that the subtask is executed on a single RSP data block. These subtasks are executed independently and in parallel, thereby speeding up the computation of the LO operation to generate the local results of RSP data blocks.

After the last subtask is completed, the Application Master transforms all local results to the master node, where the GO operation is performed to execute an ensemble algorithm on the local results to generate the final result saved in an RspRDD and written to an HDFS file for persistent storage.

When the number of RSP data blocks exceeds the available degree of parallelism, all RSP blocks will be processed in batches, and each batch will complete its computing tasks sequentially. Unprocessed RSP data blocks are computed only after the completion of the current execution batch. This batch-wise execution strategy is consistent with Spark's native scheduling. The difference is that Spark reads the HDFS block only to complete the computing task of the current stage. For complex computing tasks, it may involve multiple subsequent stages. In contrast, under LOGO, once the local computation tasks are completed in batches and corresponding local results are obtained in batches of RSP data blocks, only one GO operation is required to produce the final result.

The operational mechanism of LOGO allows the distributed computing framework to process large scale data effectively and efficiently, because LOGO consistently operates on RSP data blocks regardless of the source data size. As long as a single RSP data block can be processed in memory by a virtual machine or a slave node, LOGO can perform batch training to complete the GO operation on all RSP data blocks no matter how large the overall dataset is. Consequently, LOGO effectively breaks the memory bottleneck and demonstrates exceptional scalability, making it well-suited for large-scale machine learning tasks.

## 4 LogoML: an Open Algorithm Library for Distributed Data Analytics

In this section, LogoML, an open library of data analytic machine learning algorithms for distributed big data analytics, is proposed. First, the core operations of LogoML, LO and GO, are introduced. Then, the techniques for encapsulation of sequential algorithms for LO operation in LogoML are represented. Finally, the ensemble methods for the GO operation in LogoML are described.

## 4.1 Architecture of LogoML

The operational mechanism of the LOGO computing framework follows the divide-and-conquer strategy. In this strategy, the data set to be analyzed is divided into a set of RSP data blocks that are used as data samples, which can be independently analyzed by an analytic algorithm. A data analysis task is carried out in two stages: the local operation stage, which uses the algorithm to process the RSP data blocks in parallel and generate the local results, and the global operation stage, which uses an ensemble algorithm to integrate all local results into the ensemble result as the result of the data set. For any specific data analysis task, specific algorithms are used in the two stages.

Two operators are defined in LogoML as the core operations to carry out the two operation stages: LO operator that executes an algorithm in parallel on a set of partitions of the RDD holding the selected RSP data blocks in memory and generates the local results, and GO operator that executes an ensemble algorithm to integrate all local results into the ensemble result. This design follows the ensemble learning principle in which the LO operator generates the set of component models, and the GO operator generates the ensemble model. The LOGO computing framework enables sequential algorithms to be executed in the two core operations to simplify the implementation of the algorithms in LogoML.

In real data analysis and model building, different tasks, such as classification, regression, and clustering, are performed by specific algorithms, and the same task can be performed by different algorithms. Therefore, in LogoML, the algorithms are organized in tasks and the tasks are organized in two categories executed by the LO and GO operators, respectively. Fig. 2 shows the architecture of the LogoML library. The task categories here are used only to group algorithms. In execution, the algorithms are executed directly in the LO and GO operators.

Fig.3 shows the general workflow of performing a data analysis task using operators in LogoML. The input data set is represented in RspDataset, an RDD storing a set of RSP data blocks in partitions. An algorithm suitable for this task is selected for the LO operation, and an ensemble algorithm is selected for

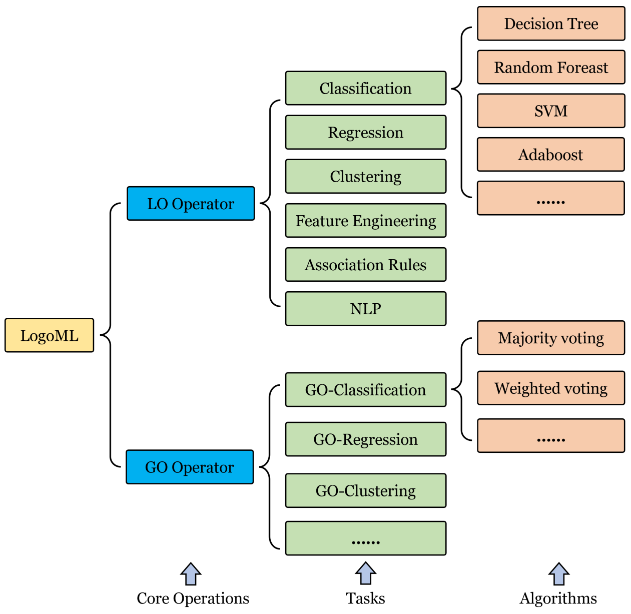


**Fig. 2** Architecture of the LogoML library.

the GO operation. In execution, the LO operator takes RspDataset RDD as input data and executes the algorithm on partitions in parallel and saves its local results to RspRDD[LocalResults]. Then, the GO operator takes RspRDD[LocalResults] as input and executes the ensemble algorithm to integrate the local results in RspRDD to the ensemble result and saves it to RspRDD[EnsembleResults]. The result RDD can be written to a file in HDFS for later use.

## 4.2 Encapsulation of algorithms in LogoML

With support of the LOGO computing framework, the two core operations of LO and GO execute sequential algorithms to process distributed data stored in partitions of an RDD. With standard data representation formats in RDDs for different categories of data analysis tasks, standard operators with standard data inputs and outputs are defined to encapsulate the source codes of specific algorithms obtained from different sources. For example, the task of building a classification model requires a training data set with class labels. A standard TrainRDD is defined as input data to the standard operators of classification algorithms. For a specific source code of a classification algorithm, if the input data format is not consistent with the TrainRDD format, the Train RDD is converted to the format of the input data of the algorithm and the converted data is used as input data to run the algorithm. Similarly, the result format of the algorithm is also converted to the format of the ResultRDD if necessary.

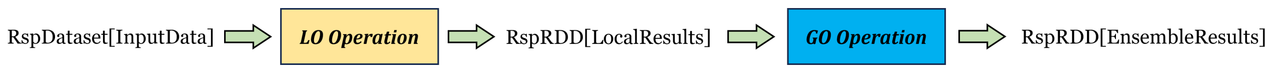


**Fig. 3** Data flow of an analysis task performed by LO and GO operations.

```
def LO_Decision_Tree(params*, trainDF)
{
   finalData = dataConvert(trainDF)
   dTree.fit(params*, finalData)
}
```

**Fig. 4** Encapsulation template for classification algorithms in LogoML.

Using this method, any code of a sequential algorithm can be encapsulated as a standard operator in LogoML.

Fig.4 shows a template for encapsulating an existing decision tree algorithm called Dtree in the standard operator called LO_Decision_Tree in LogoML. The standard input to LO_Decision_Tree is trainRDD, and params* sets the parameters for the decision tree algorithm. If the input data format of dTree is different from the partition representation trainDF of trainRDD, the dataConvert function is used to convert trainDF to finalData with the same format as the data format of algorithm dTree. Then, finalData is used as the input data of dTree to build the decision tree model. LO_Decision_Tree returns a decision tree model saved in a partition of the model RDD for the GO operator to process. Different templates are used to encapsulate algorithms in different task categories. Generally speaking, encapsulating a sequential algorithm in LogoML is carried out in the following steps.

1. Find the source code of a sequential algorithm from a repository or an open source library such as Smile, scikit-learn, etc. or a self-programming algorithm. The code must be tested on some sample data sets for its correctness.

2. Define the standard operator in LogoML, and the input and output formats of the operator. Create a template to encapsulate the algorithm.

3. Check if the input and output formats of the algorithms are consistent with the defined formats of the operator. If not, write a data conversion function to transform the input format of the operator to the input format of the algorithm, and put the converted data format as the input data of the algorithm.

4. Use a compile tool (such as Maven) to compile the encapsulated codes and related dependencies into the LogoML JAR package.

The operators in LogoML are used as standard APIs to call the encapsulated algorithms in applications. An example is shown in Code 1. This application builds a random forest[42] model from a distributed data file using LO_Decision_Tree and GO_Tree_Ensemble operators in LogoML. The application is written in 4 lines of codes in Spark. Line 1 reads a distributed data file of RSP data blocks into trainRDD. Line 2 calls the LO operator to execute LO_Decision_Tree on partitions of trainRDD to build decision tree models. Line 3 calls GO operator to execute GO_Tree_Ensemble that uses the popular voting algorithm to generate the ensemble result from the multiple decision tree models built by LO_Decision_Tree using the dTree algorithm. Line 4 saves the ensemble random forest model to disk for later use. This example demonstrates that LogoML simplifies the development of applications for distributed big data analysis.

### 4.3 Ensemble methods for GO operation in LogoML

In LogoML, the GO operator executes an algorithm to aggregate the local results, independently generated by the LO operator from a set of RSP data blocks, to the global ensemble result. These results, computed by an analytical algorithm, can be a set of statistical values, e.g., means or variances, or estimated distributions of random variables, or models of classification or regression. Different ensemble algorithms are required to ensemble the local results in response to the types of results generated by the analytic algorithms executed by the LO operator. In this section, three ensemble methods are presented.

#### 4.3.1 Ensemble methods for supervised learning

Supervised learning represents a class of machine learning algorithms that are used to build models of classification, prediction, and regression from training datasets in which objects are labeled with a number of known classes. Commonly used ensemble methods for supervised learning include popular voting, weighted popular voting[43], average[44], weighted average[45, 46], and stacking[47], which are well studied in

Code 1 Application template of LogoML

```
val trainRDD = spark.rspRead.parquet(filepath, blockNums)
val localResults = trainRDD.LO(
    trainDF => LO_Decision_Tree(maxDepth, nodeSize, ntrees, trainDF)
    )
val globalModel = localResults.GO(Go_Tree_Ensemble)
goTable.saveAsTextFile(rspfilepath)
```

ensemble learning.

```
Def GO_Tree_Ensemble(localRspRDD,ensembleType)
{
   case “popVoting”: popularVoting(localRspRDD)
   case “WeVoting”: weighedVoting(localRspRDD)
   ……
}
```

**Fig. 5** Encapsulation template for ensemble algorithms of supervised learning in LogoML.

In statistical terms, if the local results of RSP data blocks are considered as distributions of some random variables, the ensemble algorithm, which is executed by GO operator, calculates the estimates of statistics of these random variables, including means, variance, histograms, etc. Therefore, LogoML can also include algorithms for statistical analysis with LO and GO computing paradigm to handle big distributed data.

The ensemble algorithms are also encapsulated as operators in LogoML, which are executed by the GO operator. Fig. 5 shows a template for encapsulating an ensemble algorithm in an ensemble operator. The ensemble algorithm implements both popular voting and weighted voting methods to generate the ensemble decisions from the decision results of the set of decision tree models stored in localRspRDD. The two ensemble methods are implemented as two functions of popularVoting and weightedVoting. The user uses an ensemble Type parameter to select one method to use in applications.

Unlike the operators for the LO operator which take a partition as input, the operators for the GO operator take a localRspRDD as input. In implementation, the ensemble algorithm is executed on the master node to process the RDD and generate the ensemble result. It requires transferring all partitions of localRspRDD to the master node before the ensemble process starts. Therefore, an ensemble algorithm is also a sequential algorithm in nature. Using the template in Fig. 5,

```
Def GO_reClustering(localCentriods, Data)
{
   goCentroids = kmeans.fit(localCentriods)
   Data.labeling(goCentroids)
}
```

**Fig. 6** Encapsulation template for ensemble algorithms of unsupervised learning in LogoML.

it is straightforward to encapsulate other ensemble algorithms for supervised learning. For example, if localRspRDD contains a set of regression models, an averaging method can be implemented to calculate the average of all values computed by all local regression models in localRspRDD as the ensemble result.

### 4.3.2 Ensemble methods for unsupervised learning

There are many clustering algorithms available in unsupervised learning. Traditional ensemble clustering methods usually generate component clustering results from the same data set with different parameters of the same clustering algorithm or different clustering algorithms[48, 49]. Based on the association matrix of objects and clusters in the result of the component clustering, different consensus functions are developed to generate the ensemble clustering result[50, 51]. However, traditional ensemble clustering methods are not efficient in dealing with big data.

To overcome the bottleneck of clustering big data, we use the same clustering algorithm to generate component clusters from a set of RSP data blocks. Since the RSP data blocks do not intersect each other, each object is clustered to one cluster only in one component result, and the association matrix cannot be built for consensus functions. However, because RSP data blocks have similar cluster distributions, we can use the features of component clusters, such as cluster centroids, to generate the ensemble clustering result[52].

Fig.6 shows a template for encapsulating an ensemble

```
Def GO_approxFIMs(localFIMs, minSup)
{
   votedFIMs = localFIMs.voting(0.5)
   filteredFIMs = voteFIMs.filter(minSup)
   goFIMs = calTrue(filteredFIMs)
}
```

**Fig. 7** Encapsulation template for ensemble algorithms of frequent itemset mining in LogoML.

clustering algorithm using the reclustering method. Assume that localCentriods is a localRspRDD that contains the set of cluster centers of all component clusters by a clustering algorithm executed by the LO operator. The algorithm kmeans clusters the set of cluster centers and generates a set of new cluster centers as the cluster centers of the ensemble clustering result. Then, function labeling uses the ensemble cluster centers stored in goCentroids to assign all objects in Data to cluster ids by distances.

#### 4.3.3 Ensemble methods for FIM algorithms

Frequent itemset mining (FIM) algorithms are widely used in real applications, such as association rule mining, market basket analysis, and product recommendation[53]. Although some FIM algorithms are widely used, they are not efficient in dealing with a large number of business transactions and are not scalable to big transaction data.

In LogoML, LO_FP_Growth[54] operator is defined and a sequential FP_Growth algorithm is encapsulated to generate a set of frequent itemsets from an RSP data block containing a set of business transactions. In executing the LO_FP_Growth operator on multiple partitions of RSP data blocks, multiple sets of frequent itemsets are generated. An ensemble algorithm to aggregate the multiple sets into one set of frequent itemsets is developed, which uses popular voting to determine whether an itemset is put to the final set or not, and the average of the support values of the frequent itemset in the multiple sets as the support value of the itemset in the final set. The final set of frequent itemsets is an approximate set of true frequent itemsets in the distributed transaction data set.

Fig.7 shows the template for encapsulating the frequent itemset mining ensemble algorithm in LogoML. The GO_approxFIMs operator takes localFIMs as the input RDD in which each partition contains a set of frequent itemsets discovered from one RSP data block by the FP_Growth algorithm executed by the LO operator. The parameter minSup specifies the minimal support threshold for the final set of frequent itemsets. The ensemble process is carried out by three functions. The voting function checks the appearance of an itemset in all sets of frequent itemsets in all RSP data blocks. If the number of sets containing the itemset is greater than 50% of the total number of sets, the itemset is added to the final set and its support is the average of all support values in the multiple sets; otherwise, the itemset is dropped. The final set is saved in votedFIMs. The function filter removes itemsets with support values smaller than the minimal support threshold minSup. The final set is saved in filteredFIMs.

Based on the description above, everyone can build their own personalized algorithm library. To make this process more convenient, we have open-sourced the standard basic algorithm library on GitHub[¶].

## 5 Evaluations

This section presents the results of experiments conducted on both real and synthetic data sets as evaluations of performance of the algorithms in LogoML. The results demonstrate the advantages of LogoML in three aspects: more algorithms available for distributed big data analysis, high efficiency in processing big data, and scalability to big data.

### 5.1 Experiment settings

**Computing environment**. In these experiments, two computing systems were used. A desktop server with an Intel i7-11700 CPU and 64 GB of RAM was used to generate the experiment results of sequential algorithms. A cluster of 30 computing nodes was used to generate the experiment results of distributed data analysis with LogoML and MLlib in Spark. The cluster has 24 nodes with 2 Intel Xeon E5-2650 CPUs and 128 GB memory, and 6 nodes with 2 Intel Xeon E5-2630 CPUs and 128 GB memory. Spark version 3.5.0 and YARN version 3.0.0 were installed to execute the algorithms in LogoML and Spark MLlib.

For experiments with small data files, 20 executors were configured to process the distributed data sets, and each executor was allocated 4 GB of memory and 1 CPU core. For experiments of big distributed data files, a maximum of 100 executors were used, and each executor was configured with 16 GB of memory and 1 or 4 CPU cores. In the experiments, each data analysis task was repeated 11 times. The worst and best results were removed, and the average of the remaining results

[¶]https://github.com/SZU-LOGO/mllib

was used as the result of the task evaluation.

**Comparison methods**. In performance evaluation, we choose the following three computing methods as competitors of LogoML:

- **Smile**: an open source library containing a large number of machine learning and data analytical algorithms. In the experiments, the corresponding algorithms in Smile were run on the single server to process the entire data sets.
- **SparkML**: the corresponding machine learning algorithms in MLlib in Spark were run on the cluster to process the entire data sets.
- **SparkML-OS**: the corresponding machine learning algorithms in MLlib in Spark were run on the sample data sets, and the samples were taken online with the Spark's built-in sampling operator.

Since the LOGO distributed computing paradigm computes an approximate result, LogoML algorithms used 5% of the RSP data blocks in each analysis task. It is worth noting that the 5% sampling rate is not fixed. For large-scale datasets, a lower sampling rate is sufficient, whereas for small-scale datasets, a higher sampling rate is required. This is consistent with statistical theory. SparkML-OS also uses a sample of 5% of the entire data set. However, the sample was obtained by record-level random sampling of the entire data set, but LogoML used block-level random sampling of RSP data blocks, which was more efficient[55]. The total number of 15 algorithms were used in the experiments. The parameter settings for these algorithms are listed in Table 1.

**Data sets**. Two real data sets, HIGGS and MNIST_PCA with class labels, were used to evaluate the computing efficiency and the result quality of the supervised and unsupervised learning algorithms in LogoML. Since the sizes of these data sets are not large, they can be analyzed with a sequential algorithm on a single server.

Two groups of synthetic data sets were used to evaluate the performance and data scalability of the algorithms in LogoML. One group consists of a sequence of training data sets with 100 features and two classes. The sizes of these data sets range from 100GB to 10TB. These data sets were used to evaluate the performance and data scalability of the supervised and unsupervised learning algorithms in LogoML. The other group contains a sequence of transaction data sets ranging from $10^5$ transactions to $10^9$ transactions. These data sets were used to evaluate the performance and data scalability of the approximate frequent itemset mining algorithms in LogoML. The characteristics of the data sets are listed in Table 2.

All data sets were converted to RSP data representation and saved as HDFS files, and the relevant experimental codes have been open-sourced on GitHub[‖] to facilitate others to reproduce this experiment.

## 5.2 Evaluation metrics

Computing efficiency, model quality, and data scalability are three performance indicators in comparison to big data computing methods. The execution time of the computing process for an analysis task is used to evaluate computing efficiency. The accuracy of classification on test data set is used to evaluate the quality of classification models.

Regression models are evaluated as follows:

$$\mathsf{R}^2 = 1 - \frac{SSE}{SST} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{1}$$

where SSE is the residual sum of squares of difference between the model-predicted value and the true value, and SST is the total sum of squares of difference between the true value and the mean, reflecting the fluctuation of the predicted variable.

The results of clustering algorithms are evaluated as follows:

$$\mathsf{Purity} = \frac{1}{N}\sum_{k=1}^{K}\max_{j}(n_{kj}) \tag{2}$$

where $N$ is the total number of data points, $K$ is the number of clusters, and $n_{kj}$ represents the number of data points in the $k$-th cluster that belong to the true cluster $j$.

Recall and precision are used to evaluate the quality of the results of the LogoML FIM ensemble algorithm. The two metrics are calculated as follows:

$$\mathsf{Recall} = \frac{|\mathcal{F}_{\text{Approx}} \cap \mathcal{F}_{\text{True}}|}{|\mathcal{F}_{\text{True}}|} \tag{3}$$

where $\mathcal{F}_{\text{Approx}}$ and $\mathcal{F}_{\text{True}}$ represent an approximate set and a true set of frequent itemsets, respectively.

$$\mathsf{Precision} = \frac{|\mathcal{F}_{\text{Approx}} \cap \mathcal{F}_{\text{True}}|}{|\mathcal{F}_{\text{Approx}}|} \tag{4}$$

[‖]https://github.com/SZU-LOGO/LOGOML_V2

**Table 1** Parameter settings

| Algorithms | | Parameters | Values | Descriptions |
|---|---|---|---|---|
| **Supervised learning** | Decision Tree | maxDepth | 10 | The maximum depth of the tree. |
| | | nodeSize | 10 | The minimum number of instances in the splitted node. |
| | Random Forest | nTrees | 20 | The number of trees in ensemble. |
| | | maxDepth | 5 | The maximum depth of the tree. |
| | Logistic Regression | maxIter | 500 | The maximum number of iterations. |
| | | regParam | 0.1 | Regularization parameter. |
| | SVM | MaxIter | 10 | The maximum number of iterations in Spark. |
| | | RegParam | 0.1 | The soft margin penalty parameter. |
| | | Tol | $10^{-6}$ | The convergence tolerance of iterations. |
| | LDA | prior | empirical | The Prior probability distribution of each class. |
| | | Tol | $10^{-4}$ | The tolerance of the covariance matrix. |
| | QDA | prior | empirical | The Prior probability distribution of each class. |
| | | Tol | $10^{-4}$ | The tolerance of the covariance matrix. |
| | RDA | prior | empirical | The Prior probability distribution of each class. |
| | | Tol | $10^{-4}$ | The tolerance of the covariance matrix. |
| | FLD | L | 20 | The dimensionality of mapped space. |
| | | Tol | $10^{-4}$ | The tolerance of the covariance matrix. |
| | RBF | normalized | true | Whether to normalize the network output. |
| | Adaboost | nTrees | 20 | The number of trees in ensemble. |
| | | maxDepth | 20 | The maximum depth of the tree. |
| | | maxNodes | 6 | The maximum number of nodes in a tree. |
| | | nodeSize | 1 | The minimum number of instances in the splitted node. |
| **Unsupervised learning** | Kmeans | k | 2 | The number of clusters. |
| | | maxIterations | 100 | The maximum number of iterations. |
| | | Tol | $10^{-4}$ | The convergence tolerance of iterations. |
| | G-means | k | 2 | The number of clusters. |
| | | maxIterations | 100 | The maximum number of iterations. |
| | | Tol | $10^{-4}$ | The convergence tolerance of iterations. |
| | X-means | k | 2 | The number of clusters. |
| | | maxIterations | 100 | The maximum number of iterations. |
| | | Tol | $10^{-4}$ | The convergence tolerance of iterations. |
| | Bisecting Kmeans | k | 2 | The number of clusters. |
| | | maxIterations | 10 | The maximum number of iterations. |
| **Frequent itemset mining** | FP-Growth | minSupport | 0.15 | The minimal support threshold of the frequent pattern. |
| | | numPartitions | Q | The number of partitions used by parallel FP-growth. |

**Table 2** Characteristics of the real and synthetic datasets

| **Dataset** | **Size** | **Features** | **RSP size** | **Classes** |
|---|---|---|---|---|
| HIGGS | 7.48GB | 28 | 13.92MB | 2 |
| MNIST_PCA | 6.79GB | 87 | 17.17MB | 10 |
| DS1-DS28 | 100GB–10TB | 100 | 128MB | 2 |
| DS29-DS48[a] | $10^5$ - $10^9$ | - | 50000 | - |

[a]The size for these datasets is the number of transactions.

## 5.3 Evaluation results

### 5.3.1 Evaluation on small data sets

The two real data sets, HIGGS and MNIST_PCA, were used to evaluate the computing efficiency of the supervised and unsupervised learning algorithms in LogoML, and to compare the results with the results of SparkML and Smile. A total of fourteen algorithms were used in the experiment, among them, ten for

**Table 3** Comparisons of computing efficiency of supervised and unsupervised learning algorithms in execution time in seconds.

| **Algorithms** | ***HIGGS*** | | | ***MNIST_PCA*** | | |
|---|---|---|---|---|---|---|
| | ***LogoML*** | ***SparkML*** | ***Smile*** | ***LogoML*** | ***SparkML*** | ***Smile*** |
| Decision Tree | 12.0263 | 100.8525 | 118.3261 | 12.5201 | 204.6725 | 318.4387 |
| Random Forest | 21.5392 | 177.5010 | 226.4373 | 23.5213 | 320.8744 | 659.5860 |
| Logistic Regression | 16.6979 | 93.7865 | 49.3711 | 33.8754 | 292.5655 | 983.1162 |
| SVM | 227.7361 | 1862.5087 | O | × | × | × |
| LDA | 8.4780 | - | 16.8549 | 27.8300 | - | 115.2506 |
| RDA | 9.0730 | - | 31.8379 | × | - | × |
| QDA | 18.2842 | - | 18.0782 | × | - | × |
| FLD | 8.9165 | - | 21.4127 | 15.7635 | - | 112.4257 |
| RBF | 12.9134 | - | O | 17.9828 | - | O |
| Adaboost | 114.3640 | - | 9729.3896 | 163.1616 | - | 18713.1391 |
| K-means | 25.9149 | 99.00385 | 152.6613 | 38.5627 | 375.8538 | O |
| G-means | 82.5387 | - | 324.7654 | 64.4043 | - | O |
| X-means | 33.5600 | - | 342.3568 | 48.5896 | - | O |
| Bisecting k-means | 50.7253 | 298.195 | 221.8153 | 35.8564 | 1103.4421 | O |

**Note:** The 'O' indicates memory overflow. The '×' indicates that the execution is not supported. The '-' indicates that the algorithm is not available.

classification and four for clustering. The execution time was recorded to evaluate the computing efficiency of the algorithms. The classification accuracy was used to evaluate the model quality of the classification algorithms, and purity was used to evaluate the results of the clustering algorithms. In this experiment, the SparkML and Smile algorithms processed all data sets of HIGGS and MNIST_PCA, and the Smile algorithms ran on a single server. SparkML-OS method was not tested because it runs on sample data.

Table 3 shows the computing efficiency of the fourteen algorithms in analyzing the two data sets with the three computing methods. We can see that many well-known algorithms are missing in SparkML. Although the HIGGS data set was not very large, it was still difficult to analyze it entirely with SVM and RBF on a single machine. For the MNIST_PCA data set, algorithms SVM, RDA and QDA could not be used because they work on data sets of two classes. Smile's algorithms, RBF, K-means, G-means and X-means, could not generate the final results due to the memory limit. For other results which are comparable among the three computing methods, the LogoML method is the most efficient. The larger the data set and the more complex the algorithm, the more efficient the execution of the algorithm in LogoML.

In the comparison of the results between SparkML and Smile, the efficiency improvement of SparkML algorithms is not significant because the MapReduce-type implementation of the iterative machine learning algorithms in SparkML runs a big data communication overhead. Therefore, the execution time of the Smile algorithms Logistic Regression and Bisecting k-means on HIGGS is shorter than the execution time of the corresponding SparkML algorithms.

Table 4 shows the model quality of the fourteen algorithms in analyzing the two data sets with the three computing methods. For the ten algorithms of classification, LogoML method performed better than or equal to other two methods in model accuracy. This is because the LogoML models were built with the ensemble learning method, whereas the models using the other two methods are single models built from the entire data sets. For example, the Decision Tree and Logistic Regression models built with the LogoML algorithms are much better than the corresponding models of the other two methods, although the algorithms in LogoML used only 5% of the entire data set.

For the four clustering algorithms, the clustering results of the two data sets by the three methods are very close in the purity measure. The advantage of LogoML is that it contains more algorithms for clustering and is efficient in processing large data sets.

The two figures show that LogoML's algorithms exhibit the best performance in computing efficiency

**Table 4** Comparisons of model quality of supervised and unsupervised learning algorithms in accuracy and purity.

| **Algorithms** | ***HIGGS*** | | | ***MNIST_PCA*** | | |
|---|---|---|---|---|---|---|
| | ***LogoML*** | ***SparkML*** | ***Smile*** | ***LogoML*** | ***SparkML*** | ***Smile*** |
| Decision Tree | 0.6804 | 0.6431 | 0.6427 | 0.6431 | 0.6001 | 0.608 |
| Random Forest | 0.7061 | 0.7041 | 0.7042 | 0.7101 | 0.6924 | 0.6911 |
| Logistic Regression | 0.6438 | 0.6223 | 0.6241 | 0.8619 | 0.8017 | 0.8034 |
| SVM | 0.6428 | 0.5934 | O | × | × | × |
| LDA | 0.6521 | - | 0.6414 | 0.7781 | - | 0.7699 |
| RDA | 0.6567 | - | 0.6481 | × | - | × |
| QDA | 0.6582 | - | 0.6448 | × | - | × |
| FLD | 0.6487 | - | 0.6338 | 0.8095 | - | 0.7943 |
| RBF | 0.5377 | - | O | 0.4991 | - | O |
| Adaboost | 0.717 | - | 0.7121 | 0.8388 | - | 0.8094 |
| K-means | 0.5033 | 0.5014 | 0.5023 | 0.9219 | 0.9207 | O |
| G-means | 0.5012 | - | 0.5011 | 0.8982 | - | O |
| X-means | 0.5022 | - | 0.5019 | 0.9007 | - | O |
| Bisecting K-means | 0.5049 | 0.5051 | 0.5038 | 0.9301 | 0.9227 | O |

**Note:** The 'O' indicates memory overflow. The '×' indicates that the execution is not supported. The '-' indicates that the algorithm is not available.

and model quality in the fourteen supervised and unsupervised learning algorithms. The primary reason for model quality is that LogoML method uses ensemble learning which usually generates more accurate models than any single model from the same data set. The reason for computing efficiency is the non-MapReduce computing paradigm adopted in LogoML which removes the data communication overhead when executing iterative algorithms in distributed computing to build the local models. The other factor is that only 5% of the entire data set is randomly selected in model building, whereas both SparkML and Smile use the entire data set.

### 5.3.2 Evaluation on big datasets

This experiment used the two groups of synthetic data sets to evaluate the performance of LogoML algorithms when dealing with big data sets. The first 20 data sets of the first group, from DS1 to DS20, with data sizes ranging from 100GB to 2TB, were used to evaluate supervised and unsupervised algorithms. The 10 data sets of the second group from DS29 to DS38 with the number of transactions ranging from 0.5 million to 5 million were used to evaluate the frequent itemset mining algorithm, FP-Growth, on the generation of the approximate set of frequent itemsets from multiple RSP data blocks using the LogoML computing method. In this experiment, Smile algorithms were not tested because they cannot run on big data sets.

Fig. 8 illustrates the computing efficiency of three classification algorithms, two clustering algorithms and one frequent itemset mining algorithm in processing the 30 big data sets mentioned above. The six algorithms are available in SparkML, so they were used in performance comparisons. The vertical axis indicates the execution time in seconds. The horizontal axis in the first five figures represents the number of GBs of the big data sets. The horizontal axis in the last figure is the number of transactions of the 10 data sets starting from 0.5 million transactions to 5 million transactions. In the experiment, SparkML algorithms processed the entire data set in each run, but SparkML-OS algorithms processed only the online-sampled data, which was only 5% of the entire data set. LogoML algorithms selected only 5% of the RSP data blocks to process in each run. The vertical dashed line in each figure indicates the size of the largest data set that SparkML algorithms can process to generate the result. After that point, the data set hits the memory limit.

Compared with the other two methods, SparkML algorithms were clearly not efficient in processing the big data sets. Its execution time increased exponentially when the size of the data set was approaching to the memory limit due to the heavy overheads in data communication among the nodes and in data swapping between memory and hard disks, significantly affecting the computing efficiency of the algorithm execution.

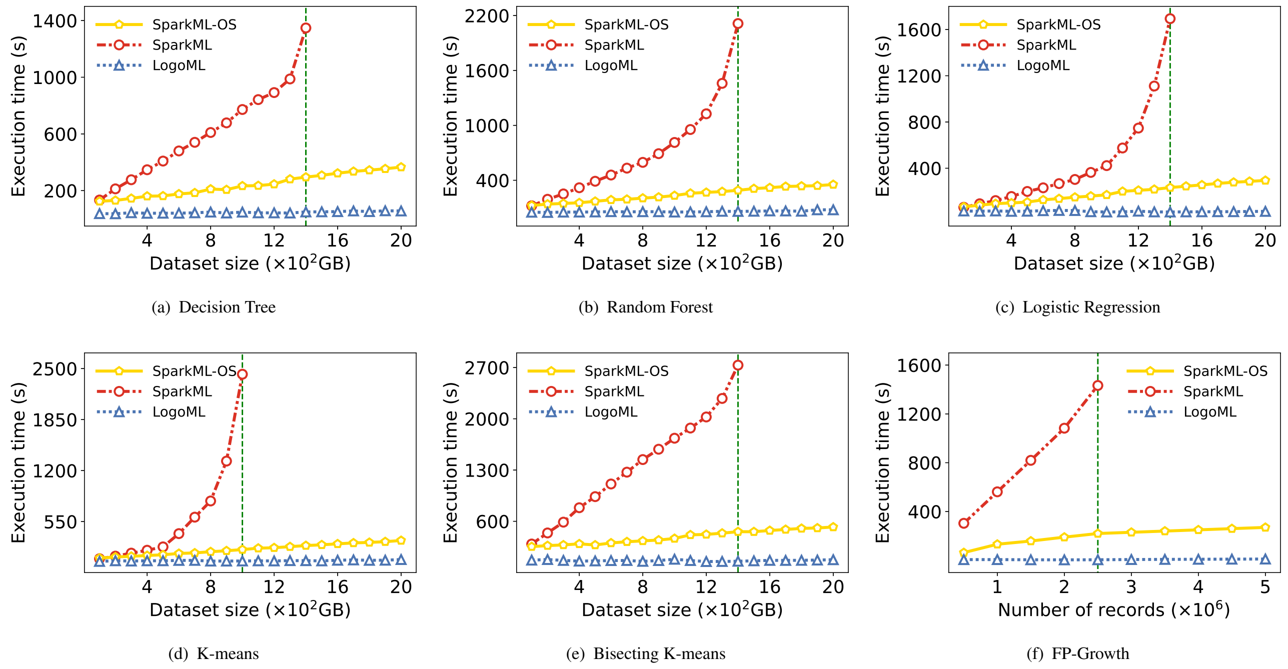


**Fig. 8** Computing efficiency of six algorithms executed on the big synthetic data sets.

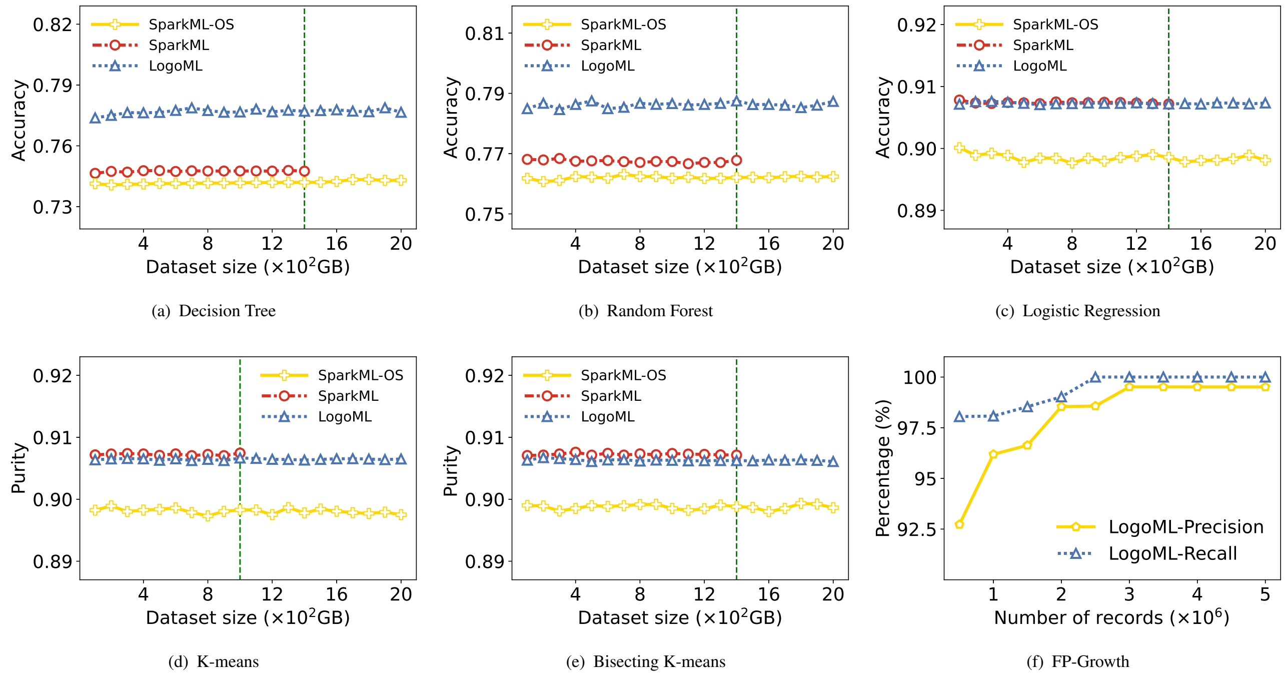
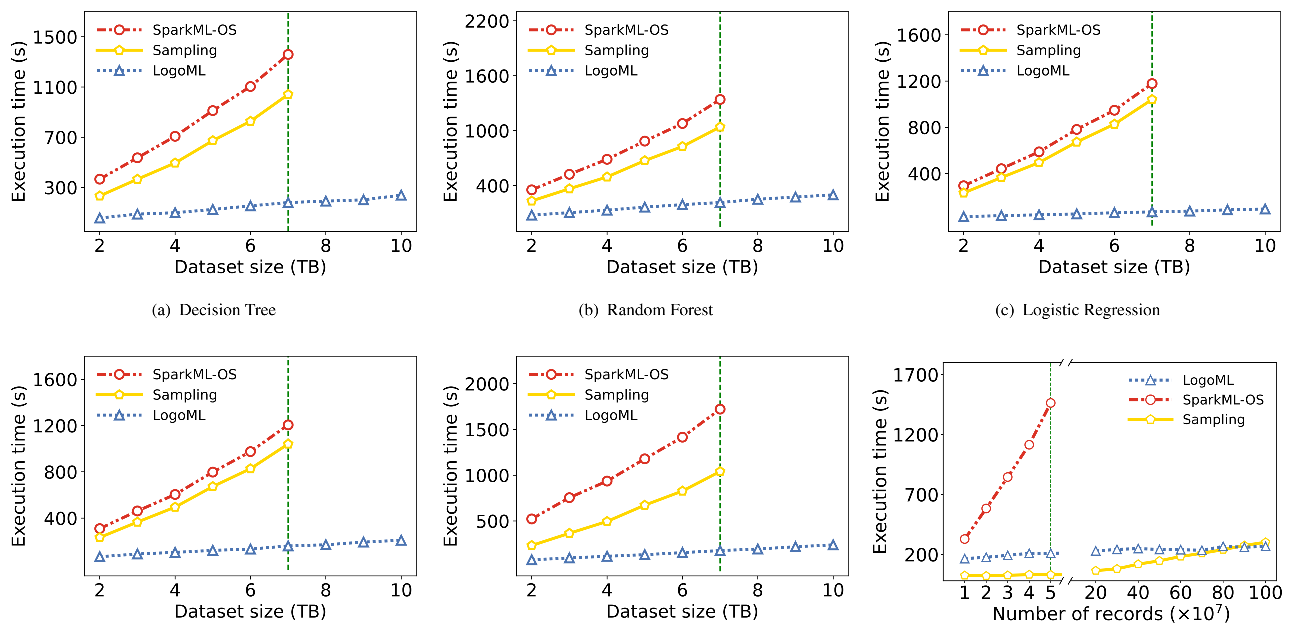


**Fig. 9** Model quality of six algorithms executed on the big synthetic data sets.

In contrast, the LogoML and SparkML-OS algorithms ran more efficiently because they only processed 5% of the entire data set in each run. As the size of the data set increased, the execution time of the SparkML-OS algorithms increased linearly due to online sampling. The execution time of LogoML algorithms remained almost the same as the size of the data set increased. The reason is that LogoML computing method uses block-level sampling to randomly select RSP data blocks, which is much faster than record-level sampling used in SparkML-OS.

Fig. 9 shows the performance of the model quality of the six algorithms in processing the 30 big data sets. The performance is measured in accuracy for



(a) Decision Tree (b) Random Forest (c) Logistic Regression

(d) K-means (e) Bisecting K-means (f) FP-Growth

**Fig. 10** Comparison of data scalability of LogoML, SparkML and SparkML-OS in processing terabyte scale data sets.

three classification algorithms and in purity for the two clustering algorithms. For the approximate set generated by the FP-Growth algorithm in LogoML, the result is measured by recall and precision. We can see that the models built with the classification algorithms in LogoML are more accurate than the models built by the algorithms in SparkML and SparkML-OS because of the ensemble method used in LogoML. For clustering algorithms, LogoML and SparkML algorithms produced similar results, whereas the results of SparkML-OS algorithms are slightly worse. Regarding the results of the LogoML frequent itemset mining algorithm, we can see the trend that both recall and precision increased as the size of the data set increased, and they converged to the point close to 100%. Generally, all results are above 97%.

### 5.3.3 Evaluation on data scalability

The nine data sets, DS20-DS28, with data sizes ranging from 2TB to 10TB in the first data group were used to test the data scalability of the three classification algorithms and the two clustering algorithms. Data sets DS30-DS48 with transactions ranging from $10^7$ to $10^9$, were used to evaluate the data scalability of the frequent itemset mining algorithms. In this experiment, only LogoML and SparkML algorithms were used to process the data sets because SparkML algorithms could not run due to the memory limit.

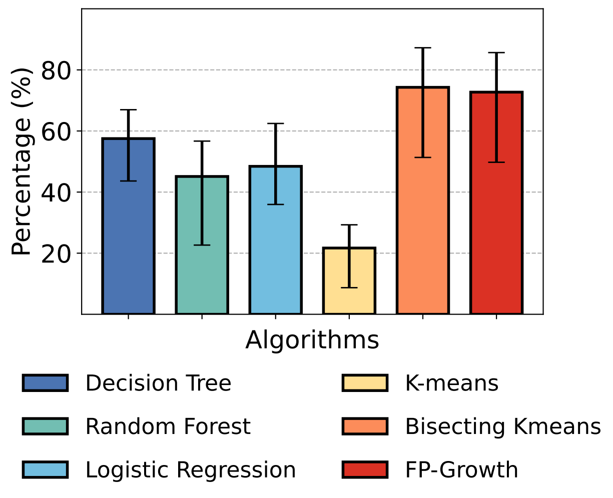


**Fig. 11** Percentage of execution time reduced on data communication by the six algorithms in LogoML.

The results of the experiment are shown in Fig. 10. The vertical dashed line in each figure indicates the data size that hits the memory limit. In the SparkML-OS experiment, the time spent on data sampling was recorded separately. In each run, both LogoML and SparkML methods used 5% of the data set to generate the results. From the figures, it can be seen clearly that LogoML algorithms were efficient in processing very large data sets in terabytes and scalable to big data sets because of its non-MapReduce computing paradigm. Comparatively, SparkML-OS used the algorithms in

Spark MLlib, which are distributed implementations in MapReduce. They consume more memory and have communication overheads in processing large distributed data sets.

Data communication among nodes is a significant overhead that affects the computing efficiency of distributed learning algorithms. Compared with the distributed learning algorithms in Spark MLlib, the learning algorithms executed by LO operator do not generate data communication overhead, therefore, the time spent on data communication is saved. Fig. 11 shows the percentage of the total execution time that the six algorithms in Spark MLlib spent on data communications. The Bisecting K-means and FP-Growth algorithms reduced 72.3% and 72.7% of the total execution time, respectively, if the communication overhead was removed. On average, approximately 50% of the total execution time can be saved when LogoML algorithms are used to replace the corresponding algorithms in Spark MLlib.

## 6 Conclusions

In this paper, we present LogoML, an open library of data analytic and machine learning algorithms for distributed big data analysis. Empowered with the RSP data representation model and the LOGO distributed computing framework, the LogoML library has the following merits over the machine learning algorithm libraries, e.g., Spark MLlib and Mahout, for distributed big data analysis:

(1) It is computationally efficient because of its non-MapReduce computing paradigm that reduces the data communication and I/O overheads significantly in executing iterative algorithms.

(2) It is scalable to big data because it uses sample data blocks in data analysis, instead of the entire data set.

(3) It has a rich collection of algorithms because it uses sequential algorithms to process large data sets in parallel and distributed data analysis. It has an open architecture for new algorithms to be added to the library.

LogoML represents a new type of algorithm library for distributed computing of data analysis and machine learning. It addresses the problems of extremely large data sets in terabytes and beyond. Currently, we are exploring the technology that deploys LogoML in multiple clusters to analyze the big data sets stored in geo-distributed data centers. At the same time, we are also exploring and designing new computing architectures to enable LogoML to support deep learning algorithms.

**Acknowledgment**

This paper was supported by the Science and Technology Major Project of Shenzhen (KJZD20230923114809020) and Natural Science Foundation of Guangdong Province (2023A1515011667).

**References**

[1] A. Jamarani, S. Haddadi, R. Sarvizadeh, M. Haghi Kashani, M. Akbari, and S. Moradi, “Big data and predictive analytics: A systematic review of applications,” *Artificial Intelligence Review*, vol. 57, no. 7, p. 176, 2024.

[2] M. K. Saggi and S. Jain, “A survey towards an integration of big data analytics to big insights for value-creation,” *Information Processing & Management*, vol. 54, no. 5, pp. 758–790, 2018.

[3] J. S. Ng, W. Y. B. Lim, N. C. Luong, Z. Xiong, A. Asheralieva, D. Niyato, C. Leung, and C. Miao, “A comprehensive survey on coded distributed computing: Fundamentals, challenges, and networking applications,” *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1800–1837, 2021.

[4] S. Yang, W. Jin, Y. Yu, and K. F. Hashim, “Optimized hadoop map reduce system for strong analytics of cloud big product data on amazon web service,” *Information Processing & Management*, vol. 60, no. 3, p. 103271, 2023.

[5] J. G. Shanahan and L. Dai, “Large scale distributed data science using apache spark,” in *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 2323–2324, 2015.

[6] A. Katsifodimos and S. Schelter, “Apache flink: Stream analytics at scale,” in *2016 IEEE international conference on cloud engineering workshop (IC2EW)*, p. 193, 2016.

[7] M. H. Iqbal, T. R. Soomro, *et al.*, “Big data analysis: Apache storm perspective,” *International journal of computer trends and technology*, vol. 19, no. 1, pp. 9–14, 2015.

[8] G. Nguyen, S. Dlugolinsky, M. Bobák, V. D. Tran, Á. L. García, I. Heredia, P. Malík, and L. Hluchý, "Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey," *Artif. Intell. Rev.*, vol. 52, no. 1, pp. 77–124, 2019.

[9] S. Tufail, H. Riggs, M. Tariq, and A. I. Sarwat, "Advancements and challenges in machine learning: A comprehensive review of models, libraries, applications, and algorithms," *Electronics*, vol. 12, no. 8, p. 1789, 2023.

[10] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[11] R. Anil, G. Çapan, I. Drost-Fromm, T. Dunning, E. Friedman, T. Grant, S. Quinn, P. Ranjan, S. Schelter, and Ö. Yilmazel, "Apache mahout: Machine learning on distributed dataflow systems.," *Journal of Machine Learning Research*, vol. 21, no. 127, pp. 1–6, 2020.

[12] X. Meng, J. K. Bradley, B. Yavuz, E. R. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. B. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar, "Mllib: Machine learning in apache spark," *Journal of Machine Learning Research*, vol. 17, pp. 1–7, 2016.

[13] X. Sun, Y. He, D. Wu, and J. Z. Huang, "Survey of distributed computing frameworks for supporting big data analysis," *Big Data Mining and Analytics*, vol. 6, no. 2, pp. 154–169, 2023.

[14] S. Salloum, J. Z. Huang, and Y. He, "Random sample partition: A distributed data model for big data analysis," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 11, pp. 5846–5854, 2019.

[15] C. Wei, S. Salloum, T. Z. Emara, X. Zhang, J. Z. Huang, and Y. He, "A two-stage data processing algorithm to generate random sample partitions for big data analysis," in *International Conference on Cloud Computing*, vol. 10967, pp. 347–364, 2018.

[16] Z. Huang, Y. He, C. Wei, and X. Zhang, "Random sample partition data model and related technologies for big data analysis," *Journal of Data Acquisition and Processing*, vol. 34, no. 3, 2019.

[17] X. Sun, L. Zhao, J. Chen, Y. Cai, D. Wu, and J. Z. Huang, "Non-mapreduce computing for intelligent big data analysis," *Eng. Appl. Artif. Intell.*, vol. 129, p. 107648, 2024.

[18] M. Dilhara, A. Ketkar, and D. Dig, "Understanding software-2.0: A study of machine learning library usage and evolution," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 4, pp. 1–42, 2021.

[19] H. Wickham and J. Bryan, *R packages.* " O'Reilly Media, Inc.", 2023.

[20] A. Khandare, N. Agarwal, A. Bodhankar, A. Kulkarni, and I. Mane, "Analysis of python libraries for artificial intelligence," in *Intelligent computing and networking: proceedings of ic-icn 2022*, pp. 157–177, Springer, 2023.

[21] J. Hao and T. K. Ho, "Machine learning made easy: a review of scikit-learn package in python programming language," *Journal of Educational and Behavioral Statistics*, vol. 44, no. 3, pp. 348–361, 2019.

[22] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

[23] J. Montiel, M. Halford, S. M. Mastelini, G. Bolmier, R. Sourty, R. Vaysse, A. Zouitine, H. M. Gomes, J. Read, T. Abdessalem, *et al.*, "River: machine learning for streaming data in python," *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 4945–4952, 2021.

[24] J. Herzen, F. Lässig, S. G. Piazzetta, T. Neuer, L. Tafti, G. Raille, T. Van Pottelbergh, M. Pasieka, A. Skrodzki, N. Huguenin, *et al.*, "Darts: User-friendly modern machine learning for time series," *The Journal of Machine Learning Research*, vol. 23, no. 1, pp. 5442–5447, 2022.

[25] P. Wu, "Pytorch 2.0: The journey to bringing compiler technologies to the core of pytorch (keynote)," in *Proceedings of the 21st ACM/IEEE International Symposium on Code Generation and Optimization*, p. 1, 2023.

[26] G. Kanagachidambaresan and N. Bharathi, "Python packages for learning algorithms," in *Learning Algorithms for Internet of Things: Applying Python Tools to Improve Data Collection Use for System Performance*, pp. 21–75, Springer, 2024.

[27] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library

for heterogeneous distributed systems," *CoRR*, vol. abs/1512.01274, 2015.

[28] D. Peteiro-Barral and B. Guijarro-Berdiñas, "A survey of methods for distributed machine learning," *Progress in Artificial Intelligence*, vol. 2, no. 1, pp. 1–11, 2013.

[29] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *ACM computing surveys (CSUR)*, vol. 53, no. 2, pp. 1–33, 2021.

[30] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "Haloop: Efficient iterative data processing on large clusters," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 285–296, 2010.

[31] S. Schelter, A. Palumbo, S. Quinn, S. Marthi, and A. Musselman, "Samsara: Declarative machine learning on distributed dataflow systems," in *NIPS Workshop MLSystems*, 2016.

[32] M. Assefi, E. Behravesh, G. Liu, and A. P. Tafti, "Big data machine learning using apache spark mllib," in *2017 ieee international conference on big data (big data)*, pp. 3492–3498, 2017.

[33] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *9th USENIX symposium on networked systems design and implementation (NSDI 12)*, pp. 15–28, 2012.

[34] M. Mezati and I. Aouria, "Flink-ml: machine learning in apache flink," *Brazilian Journal of Technology*, vol. 7, no. 4, pp. e74577–e74577, 2024.

[35] S. Vidhyut, S. K. Uppada, and B. SivaSelvan, "An online supervised learning framework for crime data," in *Conference Proceedings of ICDLAIR2019*, pp. 21–36, Springer, 2021.

[36] Y. Benlachmi and M. L. Hasnaoui, "Big data and spark: Comparison with hadoop," in *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, pp. 811–817, 2020.

[37] Y. Benlachmi and M. L. Hasnaoui, "Big data and spark: Comparison with hadoop," in *2020 Fourth World conference on smart trends in systems, security and sustainability (Worlds4)*, pp. 811–817, IEEE, 2020.

[38] S. Salloum, J. Z. Huang, and Y. He, "Empirical analysis of asymptotic ensemble learning for big data," in *Proceedings of the 3rd IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*, pp. 8–17, 2016.

[39] S. Salloum, J. Z. Huang, Y. He, and X. Chen, "An asymptotic ensemble learning framework for big data analysis," *IEEE Access*, vol. 7, pp. 3675–3693, 2019.

[40] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pp. 1–10, 2010.

[41] Y. He, J. Z. Huang, H. Long, Q. Wang, and C. Wei, "I-sampling: A new block-based sampling method for large-scale dataset," in *2017 IEEE International Congress on Big Data (BigData Congress)*, pp. 360–367, 2017.

[42] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[43] S. Khalifa, P. Martin, and R. Young, "Label-aware distributed ensemble learning: A simplified distributed classifier training model for big data," *Big Data Research*, vol. 15, pp. 1–11, 2019.

[44] K. Sid and M. Batouche, "Ensemble learning for large scale virtual screening on apache spark," in *6th IFIP International Conference on Computational Intelligence and Its Applications (CIIA)*, vol. 522, pp. 244–256, 2018.

[45] Z. Fang, Y. Wang, L. Peng, and H. Hong, "A comparative study of heterogeneous ensemble-learning techniques for landslide susceptibility mapping," *International Journal of Geographical Information Science*, vol. 35, no. 2, pp. 321–347, 2021.

[46] A. Galicia, R. L. Talavera-Llames, A. T. Lora, I. Koprinska, and F. Martínez-Álvarez, "Multi-step forecasting for big data time series based on ensemble learning," *Knowledge-Based Systems*, vol. 163, pp. 830–841, 2019.

[47] Y. Wang, D. Wang, N. Geng, Y. Wang, Y. Yin, and Y. Jin, "Stacking-based ensemble learning of decision trees for interpretable prostate cancer detection," *Applied Soft Computing*, vol. 77, pp. 188–204, 2019.

[48] T. Alqurashi and W. Wang, "Clustering ensemble method," *International Journal of Machine Learning*

*and Cybernetics*, vol. 10, no. 6, pp. 1227–1246, 2019.

[49] G. Yu, L. Ren, J. Wang, C. Domeniconi, and X. Zhang, “Multiple clusterings: Recent advances and perspectives,” *Computer Science Review*, vol. 52, p. 100621, 2024.

[50] K. Golalipour, E. Akbari, S. S. Hamidi, M. Lee, and R. Enayatifar, “From clustering to clustering ensemble selection: A review,” *Engineering Applications of Artificial Intelligence*, vol. 104, p. 104388, 2021.

[51] M. R. Mahmoudi, H. Akbarzadeh, H. Parvin, S. Nejatian, V. Rezaie, and H. Alinejad-Rokny, “Consensus function based on cluster-wise two level clustering,” *Artificial Intelligence Review*, vol. 54, no. 1, pp. 639–665, 2021.

[52] A. L. Fred and A. K. Jain, “Combining multiple clusterings using evidence accumulation,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 27, no. 6, pp. 835–850, 2005.

[53] X. Sun, A. Ngueilbaye, K. Luo, Y. Cai, D. Wu, and J. Z. Huang, “A scalable and flexible basket analysis system for big transaction data in spark,” *Inf. Process. Manag.*, vol. 61, no. 2, p. 103577, 2024.

[54] K. Wang, L. Tang, J. Han, and J. Liu, “Top down fp-growth for association rule mining,” in *Pacific-Asia conference on knowledge discovery and data mining*, pp. 334–340, Springer, 2002.

[55] M. S. Mahmud, J. Z. Huang, S. Salloum, T. Z. Emara, and K. Sadatdiynov, “A survey of data partitioning and sampling methods to support big data analysis,” *Big Data Mining and Analytics*, vol. 3, no. 2, pp. 85–101, 2020.

**Xudong Sun** is Post-doctoral Fellow with College of Management at Shenzhen University, China. He received his Ph.D. degree in computer science and technology at Shenzhen University in 2024, and a Master degree in computer science and technology at Shenzhen University China in 2019. He received a Bachelor’s degree in Software Engineering from Tianjin Normal University in 2011. His research concerns big data mining, distributed and parallel computing, and distributed big data approximate computing. He is currently serving as a guest editor for the top journal Information Fusion.

**Yongda Cai** received his Ph.D. degree in Computer Science and Technology from Shenzhen University in 2025. He is currently a teacher with the College of Computer Science, Guangdong Polytechnic Normal University. His current research interests include machine learning, data mining and big data technologies.

**Yi Tao** is currently pursuing a Master’s degree in Computer Science and Technology at Shenzhen University. She obtained her Bachelor’s degree in Computer Science and Technology from Jiangxi Normal University in 2024. Her current research interests include distributed parallel computing and distributed big data approximate computing.

**Langjie Mai** is currently a Master’s candidate at the Big Data Institute, Shenzhen University. He received his Bachelor’s degree at South China Agricultural University in 2023. His current research is focused on distributed data-parallel computing and approximate computing for big data analysis.

**Joshua Zhexue Huang** is a distinguished professor and director of Big Data Institute in the college of computer science & software engineering at Shenzhen University. He received a Ph.D. degree from Royal Institute of Technology, Sweden in 1993. He has more than 400 publications in conferences and journals, and is known for his contributions to the development of a series of k-means-type clustering algorithms in data mining, such as k-modes, fuzzy k-modes, k-prototypes, and w-k-means that are widely cited and used. He is a Stanford University’s career-long impact and single-year impact world’s top 2% scientist, and an Elsevier’s highly cited Chinese researcher.