Received February 11, 2021, accepted March 1, 2021, date of publication March 4, 2021, date of current version March 11, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3063929

An Efficient Rolling Bearing Fault Diagnosis Method Based on Spark and Improved Random Forest Algorithm

LANJUN WAN[1,2], KUN GONG[1,2], GEN ZHANG[1,2], XINPAN YUAN[1,2], CHANGYUN LI[2], AND XIAOJUN DENG[1]

[1]School of Computer Science, Hunan University of Technology, Zhuzhou 412007, China
[2]Hunan Key Laboratory of Intelligent Information Perception and Processing Technology, Hunan University of Technology, Zhuzhou 412007, China

Corresponding author: Xinpan Yuan (yuanxinpan@hut.edu.cn)

This work was supported in part by the National Natural Science Foundation for Young Scientists of China under Grant 61702177, in part by the Natural Science Foundation of Hunan Province, China, under Grant 2019JJ60048 and Grant 2019JJ60054, in part by the Major Project for New Generation under Grant 2018AAA0100400, in part by the National Key Research and Development Project under Grant 2018YFB1700204 and Grant 2018YFB1003401, and in part by the Key Research and Development Project of Hunan Province under Grant 2019GK2133.

ABSTRACT The random forest (RF) algorithm is a typical representative of ensemble learning, which is widely used in rolling bearing fault diagnosis. In order to solve the problems of slower diagnosis speed and repeated voting of traditional RF algorithm in rolling bearing fault diagnosis under the big data environment, an efficient rolling bearing fault diagnosis method based on Spark and improved random forest (IRF) algorithm is proposed. By eliminating the decision trees with low classification accuracy and those prone to repeated voting in the original RF, an improved RF with faster diagnosis speed and higher classification accuracy is constructed. For the massive rolling bearing vibration data, in order to improve the training speed and diagnosis speed of the rolling bearing fault diagnosis model, the IRF algorithm is parallelized on the Spark platform. First, an original RF model is obtained by training multiple decision trees in parallel. Second, the decision trees with low classification accuracy in the original RF model are filtered. Third, all path information of the reserved decision trees is obtained in parallel. Fourth, a decision tree similarity matrix is constructed in parallel to eliminate the decision trees which are prone to repeated voting. Finally, an IRF model which can diagnose rolling bearing faults quickly and effectively is obtained. A series of experiments are carried out to evaluate the effectiveness of the proposed rolling bearing fault diagnosis method based on Spark and IRF algorithm. The results show that the proposed method can not only achieve good fault diagnosis accuracy, but also have fast model training speed and fault diagnosis speed for large-scale rolling bearing datasets.

INDEX TERMS Fault diagnosis, random forest, rolling bearing, spark platform, sub-forest optimization.

I. INTRODUCTION

Rolling bearings are the most critical and easily damaged components in rotating machinery, the availability, reliability, and productivity of rotating machinery depend on the health state of rolling bearings, therefore the rolling bearing fault diagnosis is very vital to the stable, reliable, and efficient operation of rotating machinery [1], [2]. In a production environment, rolling bearings usually operate in complicated and various working conditions, and the vibration signals collected by sensors are increasing rapidly during the operation of rolling bearings, how to accurately and quickly perform rolling bearing fault diagnosis is an important issue to be solved.

In recent years, with the rapid development of machine learning algorithms and deep learning algorithms, the data-driven fault diagnosis methods have been paid more and more attention. Li *et al.* [3] used hierarchical symbol dynamic entropy and binary tree support vector machine for rolling bearing fault diagnosis. Zhou *et al.* [4] proposed a rolling

The associate editor coordinating the review of this manuscript and approving it for publication was Xiaowen Chu.

bearing fault diagnosis method based on K-Means clustering algorithm and k-nearest neighbor algorithm. Chen *et al.* [5] combined permutation entropy of variational mode decomposition (VMD) and decision tree to diagnose rolling bearing faults. Wan *et al.* [6] combined back-propagation neural network (BPNN) optimized by quantum particle swarm optimization (QPSO) algorithm and Dempster-Shafer evidence theory to diagnose rolling bearing faults. Xie *et al.* [7] developed a rolling bearing fault diagnosis method based on deep belief network optimized by Nesterov momentum. Wen *et al.* [8]–[10] improved three two-dimensional convolutional neural networks (CNN) including LeNet-5 [11], VGG-19 [12], and ResNet-50 [13], and applied them to rolling bearing fault diagnosis. Wang *et al.* [14] transferred a well-known AlexNet model [15] into rolling bearing fault diagnosis, and adopted eight different time-frequency images to validate the effectiveness of the fault diagnosis model. Liu *et al.* [16] applied the improved auto-encoder based on recurrent neural network to fault diagnosis of rolling bearing. The above researches built rolling bearing fault classifiers with high precision based on traditional machine learning algorithms or deep learning algorithms, while ensemble learning can combine multiple base classifiers into an ensemble classifier with higher precision and stronger generalization ability.

RF is the most popular ensemble learning algorithm in recent years [17] and has been widely used in the field of fault diagnosis. Wang *et al.* [18] extracted fault features of rolling bearing through wavelet packet decomposition and conducted dimensionless processing, and five best features are selected to construct the feature subspace of RF, which achieves good diagnosis accuracy and robustness under noise environment. Xu *et al.* [19] adopted the continuous wavelet transform to convert vibration signals into two-dimensional gray images, and the features extracted from gray images by CNN are input into RF to effectively diagnose rolling bearing faults. Tang *et al.* [20] extracted fault features by improved fast spectral correlation method and optimized the super-parameters of RF with PSO algorithm, which improves the fault diagnosis accuracy of rolling bearing. Han and Jiang [21] utilized VMD to decompose vibration signals to build an autoregressive (AR) model, and the parameters and residuals of AR model are used as the input of RF to obtain a rolling bearing fault diagnosis model with high diagnosis accuracy. Kundu *et al.* [22] proposed a RF regression method based on ensemble decision tree for remaining useful life prediction (RUL), and the correlation coefficient values obtained from the vibration signals collected by multiple sensors are fused to improve the prediction accuracy. The results show that the proposed method can effectively predict the RUL of spur gears under natural pitting progression. Pang *et al.* [23] developed a generalized multi-scale dynamic time warping algorithm to extract fault features from vibration signals of wind turbine gearbox (WTGB), the Laplace Score method is used to select the sensitive features to construct the eigenvector, and RF is adopted to perform fault state classification. The results prove that the proposed method can not only accurately and efficiently identify different fault states of WTGB, but also provide a higher accuracy compared with the other fault state classification methods.

With the improvements of automation and intelligent levels of mechanical equipment and the expansion of production scale, the amount of data generated by using sensors to monitor the operation states of mechanical equipment is growing explosively, and fault diagnosis in the industrial big data environment has become a research hot issue [24], [25]. In recent years, the researches related to fault diagnosis under the background of industrial big data [26]–[31] mostly use MapReduce [32] or Spark [33] to parallelize fault diagnosis models, which greatly improve the training speed and fault diagnosis speed of fault diagnosis models. Spark is a big data parallel processing platform based on memory computing, which has faster computing speed than MapReduce and is more suitable for rapid processing of large-scale rolling bearing vibration data. At present, many researchers have studied the parallelization of RF algorithm based on Spark, and have successfully applied them in clinical service guiding [34], weather forecast [35], credit classification [36], insurance data analysis [37], and recommendation system [38]. These works show that Spark can effectively parallelize RF algorithm and improve the training speed of RF models.

RF as an ensemble classifier usually combines more base classifiers to obtain better fault diagnosis accuracy, but it must wait for all base classifiers in random forest to complete the diagnosis before getting the final diagnosis results. When processing large-scale rolling bearing datasets, too many base classifiers will seriously affect the fault diagnosis speed and more easily lead to repeated voting. To solve the problems of slower diagnosis speed and repeated voting of traditional RF algorithm in rolling bearing fault diagnosis under the big data environment, an efficient rolling bearing fault diagnosis method based on Spark and IRF algorithm is proposed in this paper, which can not only significantly improve the speed of fault diagnosis, but also improve the fault diagnosis accuracy to a certain extent.

The main contributions of this paper are as follows.

- An IRF algorithm based on sub-forest optimization is proposed. By eliminating the decision trees with low classification accuracy and those prone to repeated voting in the original RF, an improved RF with faster diagnosis speed and higher classification accuracy is constructed.
- A method to construct a decision tree similarity matrix quickly and parallelly based on Spark is proposed. This method not only is suitable for large-scale data, but also effectively reduces the time spent on sub-forest optimization.
- A rolling bearing fault diagnosis method based on Spark and IRF algorithm is designed and implemented. By executing IRF algorithm efficiently and parallelly on the Spark platform, the training speed and diagnosis speed of fault diagnosis model are significantly improved.

- A series of experiments are conducted to evaluate the effectiveness of the proposed method. The experimental results show that the proposed method can not only obtain good fault diagnosis accuracy, but also has fast model training speed and fault diagnosis speed for large-scale rolling bearing datasets.

The rest of this paper is organized as follows. Section II introduces the basic theory used in this paper. Section III describes the proposed IRF algorithm. Section IV discusses the rolling bearing fault diagnosis method based on Spark and IRF algorithm. Section V presents the experimental results and analysis. Section VI gives the conclusion of this paper.

II. BASIC THEORY

This section introduces the principle of RF algorithm and that of Spark parallel computing.

A. THE PRINCIPLE OF RF ALGORITHM

RF [39] is a classical algorithm in ensemble learning, which trains multiple decision trees to build an ensemble classifier, as shown in Fig. 1. Several different training subsets are obtained by sampling with replacement from the training set. Each training subset is used to train a decision tree, and multiple trained decision trees can form an ensemble classifier. When one sample is input into the ensemble classifier, each decision tree will output a classification result, and the final classification result can be obtained by the majority voting. The basic steps of RF algorithm are as follows.

Step 1. k training subsets $D = \{D_1, D_2, \ldots, D_k\}$ with the same size as the training set are obtained by sampling with replacement from the training set.

Step 2. Each sample of a training subset contains n features. First, m ($m \leq n$) features are randomly selected from n features to construct a feature subspace S. Second, the best splitting point of decision tree nodes is calculated to generate a node according to S. The above process is repeated until the stop criterion is satisfied, and then the training of a decision tree is completed. After completing the training of k training subsets in this way, k decision trees $DT = \{DT_1, DT_2, \ldots, DT_k\}$ can be obtained.

Step 3. Each decision tree is tested by each sample of the test set and k classification results $R = \{R_1, R_2, \ldots, R_k\}$ are obtained.

Step 4. The final classification result is obtained by voting on k classification results according to the majority voting.

The classification accuracy of RF algorithm is determined by the classification accuracy of each base classifier and the similarity between base classifiers, which is positively correlated with the classification accuracy of each base classifier and negatively correlated with the similarity between base classifiers [40]. The higher the classification accuracy of each base classifier is, the higher the confidence degree of voting is, and the higher the classification accuracy of RF algorithm is. The higher the similarity between base classifiers, the higher the probability of repeated voting, and the lower

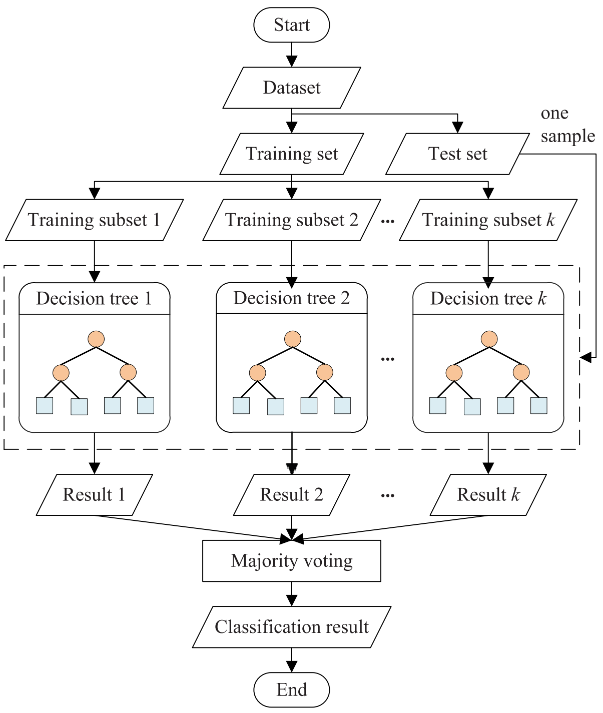


FIGURE 1. The flowchart of RF algorithm.

the classification accuracy of RF algorithm. The similarity between base classifiers is determined by m, and the base classifiers with lower similarity can be obtained by selecting the appropriate m.

B. THE PRINCIPLE OF SPARK PARALLEL COMPUTING

Spark [33] is a big data parallel computing framework based on memory computing developed by the AMP Laboratory of the University of California, Berkeley, and the core of which is the resilient distributed dataset (RDD) [41]. An RDD has multiple partitions, and each of them is a dataset fragment. Different partitions of an RDD can be saved on different nodes of the cluster, so that the parallel computing of a Spark application can be realized by calculating each partition of RDD. The RDD provides abundant transformation operators, and a Spark application can be expressed as a series of RDD transformation operations. The transformation operations between different RDDs with narrow dependencies can be pipelined to avoid the storage of intermediate results, thus significantly reducing the disk read/write, data replication, and serialization overheads.

The running process of RDD is shown in Fig. 2. First, RDD objects are created and performed a series of transformation operations. Second, SparkContext is responsible for calculating the dependency relationship between different RDD objects and building a directed acyclic graph (DAG). Third, the DAG scheduler (DAGScheduler) decomposes DAG into multiple stages. Each stage, also called a task set, contains multiple tasks. Finally, tasks are distributed to executors of each worker node through the cluster manager.

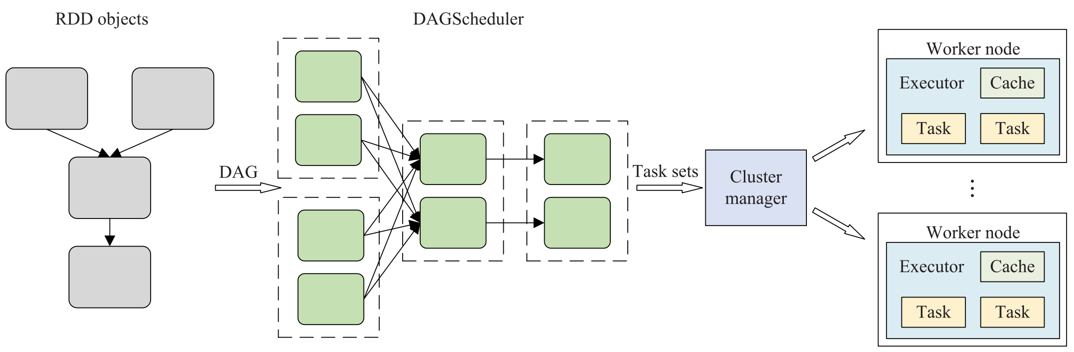


FIGURE 2. The running process of RDD.

III. THE PROPOSED IRF ALGORITHM

This section presents the IRF algorithm proposed in this paper, mainly including the sub-forest optimization and the construction of a similarity matrix.

A. THE SUB-FOREST OPTIMIZATION

Aiming at the problems of slower diagnosis speed and repeated voting of traditional RF algorithm in rolling bearing fault diagnosis under the big data environment, the IRF algorithm based on sub-forest optimization is proposed. By eliminating the decision trees with low classification accuracy and those prone to repeated voting in the original RF, an improved RF with faster diagnosis speed and higher classification accuracy is constructed. The basic process of IRF algorithm is shown in Fig. 3, including the following steps.

Step 1. Train an original RF model. First, k training subsets are obtained by sampling with replacement from the training set. Second, for each training subset, several feature subspaces are constructed by randomly selecting features, and the training is conducted according to CART decision tree algorithm. Finally, k decision trees $DT = \{DT_1, DT_2, \ldots, DT_k\}$ are obtained and combined into an original RF model.

Step 2. Filter decision trees according to the classification accuracy. The classification accuracy of each decision tree is evaluated by the validation set, and the sub-forest containing w decision trees is obtained after decision trees with low classification accuracy are eliminated.

Step 3. Traverse decision trees of the sub-forest. All path information of the sub-forest is obtained by traversing w decision trees.

Step 4. Construct a similarity matrix. The similarity between decision trees of the sub-forest is calculated according to Algorithm 1, and the following similarity matrix is obtained, where $treeSim_{i,j}$ denotes the similarity between DT_i and DT_j $(1 \leq i, j \leq w)$.

$$\begin{bmatrix} 1 & treeSim_{1,2} & \ldots & treeSim_{1,w} \\ treeSim_{2,1} & 1 & \ldots & treeSim_{2,w} \\ \vdots & \vdots & \ddots & \vdots \\ treeSim_{w,1} & treeSim_{w,2} & \ldots & 1 \end{bmatrix}$$

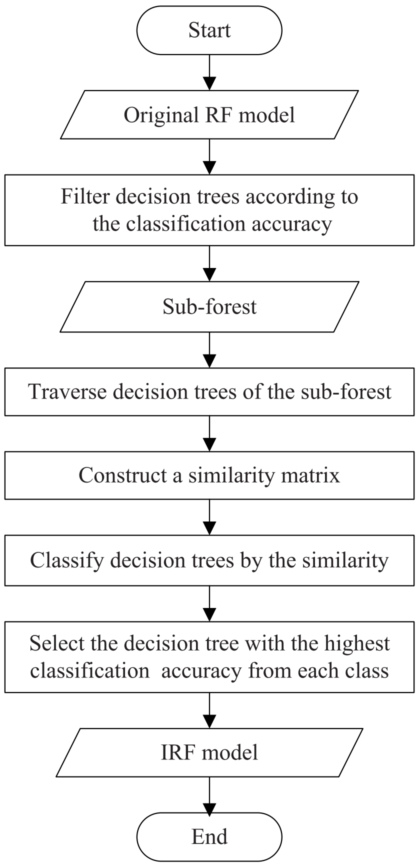

Start
Original RF model
Filter decision trees according to the classification accuracy
Sub-forest
Traverse decision trees of the sub-forest
Construct a similarity matrix
Classify decision trees by the similarity
Select the decision tree with the highest classification accuracy from each class
IRF model
End

FIGURE 3. The flowchart of IRF algorithm.

Step 5. Classify decision trees by the similarity. Firstly, the decision tree classification is performed according to the first row of the similarity matrix. These decision trees whose similarity with DT_1 exceeds the specified threshold value are classified into one class. Secondly, the decision tree classification is performed according to the i-th $(2 \leq i \leq w)$ row of the similarity matrix. Determining whether DT_i has been classified into one certain class, if so, skip the row; if not, these decision trees whose similarity with DT_i exceeds the specified threshold value are classified into one class. Finally, w decision trees can be classified into m classes according to the similarity matrix.

Step 6. Select the decision tree with the highest classification accuracy from each class. When classifying decision trees by the similarity, it is possible that one decision tree is classified into several classes and its classification accuracy is the highest among these classes, then it would be repeatedly selected, which will result in repeated voting. In order to avoid selecting duplicate decision trees and ensure the diversity of the decision trees of RF model, when selecting the decision tree with the highest classification accuracy from each class, it needs to be compared with the selected decision trees. If the selected decision trees contain a decision tree whose similarity with it exceeds the specified threshold value or a decision tree which is same as it, according to the descending order of classification accuracy, the decision trees in this class will be compared with the selected decision trees in turn until the decision tree whose similarity with each selected decision tree is lower than the specified threshold value is found.

Step 7. Combine m decision trees into an IRF model.

B. THE CONSTRUCTION OF A SIMILARITY MATRIX

This subsection describes in detail how to construct a similarity matrix. First, each path of each decision tree is traversed to obtain all path information. Second, the similarity between nodes is calculated. Third, the similarity between paths is calculated. Finally, the similarity between decision trees is calculated to construct a similarity matrix, as shown in Algorithm 1.

1) THE CALCULATION OF NODE SIMILARITY

Determining whether the features selected by two nodes are the same, if they are different, the similarity of the two nodes is 0; if they are the same, the similarity of the two nodes is calculated by

$$nodeSim_{r,s} = \begin{cases} 1, & \text{if } |n_r - n_s|/n_r \le \alpha, \\ 0, & \text{otherwise;} \end{cases} \tag{1}$$

where $nodeSim_{r,s}$ represents the similarity between node r and node s, n_r and n_s are the threshold values of node r and node s respectively, and α can be adjusted by the user and is usually set to 0.2.

2) THE CALCULATION OF PATH SIMILARITY

The prerequisite of calculating the similarity between the path of a decision tree and that of another decision tree is the root nodes of the two trees must be similar. When calculating the path similarity, determining whether the leaf nodes (i.e., fault diagnosis results) of the two paths are the same, if they are different, the similarity of the two paths is 0; if they are the same, the weighted similarity of the two paths is calculated by

$$pathSim_{i,j} = \frac{1 + \sum_{m=2}^{n} nodeSim_{i.m,j.m} + 1}{\min(pathLength_i, pathLength_j)} \times pathLength_i, \tag{2}$$

Algorithm 1 The Construction of a Similarity Matrix

```
Require: The number of decision trees $w$ and all path information of the sub-forest
Ensure: A similarity matrix
1:  for $a = 1$ to $w$ do
2:    for $b = 1$ to $w$ do
3:      if $a = b$ then
4:        $treeSim_{a,b} = 1$;
5:        continue;
6:      else
7:        if the root node of $DT_a$ and that of $DT_b$ are similar then
8:          $\gamma = 0$;
9:          $\delta = 0$;
10:         for $i = 1$ to the number of paths of $DT_a$ do
11:           $pathMaxSim_i = 0$;
12:           for $j = 1$ to the number of paths of $DT_b$ do
13:             if the leaf node of the $i$-th path and that of the $j$-th path are the same then
14:               $\xi = 0$;
15:               $\varphi = \min(pathLength_i, pathLength_j)$;
16:               for $m = 1$ to $\varphi$ do
17:                 Calculate the node similarity $nodeSim_{i.m,j.m}$ by (1);
18:                 $\xi += nodeSim_{i.m,j.m}$;
19:               end for
20:               $pathSim_{i,j} = \xi / \varphi \times pathLength_i$;
21:             else
22:               $pathSim_{i,j} = 0$;
23:             end if
24:             $pathMaxSim_i = \max(pathMaxSim_i, pathSim_{i,j})$;
25:           end for
26:           $\gamma += pathMaxSim_i$;
27:           $\delta += pathLength_i$;
28:         end for
29:         $treeSim_{a,b} = \gamma / \delta$;
30:       else
31:         $treeSim_{a,b} = 0$;
32:       end if
33:     end if
34:   end for
35: end for
```

where $pathSim_{i,j}$ denotes the similarity between path i of a decision tree and path j of another decision tree, $pathLength_i$ and $pathLength_j$ represent the number of nodes in path i and path j respectively, and $nodeSim_{i.m,j.m}$ ($2 \le m \le n$) is the similarity between the m-th node in path i and the m-th node in path j, where $n = \min(pathLength_i, pathLength_j) - 1$. When calculating the path similarity according to (2), it is required that the root nodes of the two paths are similar and the leaf nodes of the two paths are the same, thus the sum of node similarity is at least 2. When calculating the similarity $pathSim_{i,j}$ between path i and path j, the number of nodes $pathLength_i$ in path i is taken as the weight.

3) THE CALCULATION OF DECISION TREE SIMILARITY

For brevity, the calculation of the similarity between DT_a and DT_b is taken as an example. Firstly, determining whether the root nodes of DT_a and DT_b are similar according to (1), if not, the similarity of the two trees is 0. Secondly, the similarity between the i-th path of DT_a and each path of DT_b is calculated according to (2), and the maximum value of them is taken as the maximum similarity $pathMaxSim_i$ of path i. Finally, the similarity $treeSim_{a,b}$ between DT_a and DT_b is calculated by

$$treeSim_{a,b} = \frac{\sum_{i=1}^{l} pathMaxSim_i}{\sum_{i=1}^{l} pathLength_i}, \tag{3}$$

where l represents the number of paths of DT_a.

4) TIME COMPLEXITY ANALYSIS OF SUB-FOREST OPTIMIZATION

The most time consuming step in the sub-forest optimization is the construction of a similarity matrix. As can be seen from Algorithm 1, the outermost for-loop is repeated w times (line 1). The second for-loop is repeated w times (line 2). The third for-loop is repeated l times (line 10), where l is the number of paths of DT_a. The fourth for-loop is repeated u times (line 12), where u is the number of paths of DT_b. The innermost for-loop is repeated φ times (line 16). The loop body in lines 17-18 takes constant time. Therefore, the overall time complexity of the sub-forest optimization is $O(w^2lu\varphi)$.

IV. THE FAULT DIAGNOSIS METHOD BASED ON SPARK AND IRF ALGORITHM

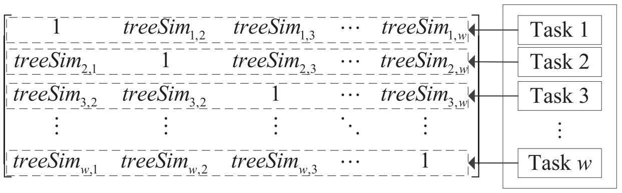
This section first discusses the parallel design and implementation of IRF algorithm based on Spark platform, and then gives the overall process of rolling bearing fault diagnosis based on Spark and IRF algorithm.

A. THE PARALLEL DESIGN OF IRF ALGORITHM BASED ON SPARK PLATFORM

The training of IRF model can be divided into two stages: the training of original RF model and the optimization of sub-forest. Due to the huge computational costs of the two stages, if they can be parallelized using Spark, which will greatly improve the training speed of IRF model.

It can be seen from Fig. 1 that the construction of different decision trees in the random forest are independent of each other, and therefore the training of original RF model can be parallelized. The tasks of constructing decision trees in the original RF can be reasonably assigned to each worker node of a Spark platform, and the construction of decision trees can be executed in parallel on different worker nodes.

The sub-forest optimization mainly includes the following two procedures. One is that each path of each decision tree in the sub-forest is traversed to get all path information. Another is that the similarity between decision trees in the sub-forest is calculated to construct a similarity matrix, which has a huge computational cost. Therefore, the sub-forest optimization can be parallelized from the following two aspects. The first aspect is that the traversals of decision trees are parallelized. Suppose that the sub-forest includes w decision trees, these w decision trees can be evenly distributed to each worker node of a Spark platform. The traversals of w decision trees are executed in parallel on different worker nodes, and the obtained path information is collected to the master node. The second aspect is that the construction of a similarity matrix is parallelized. As shown in Algorithm 1, the construction of a similarity matrix contains a five-layer nested for-loop, and each layer can be parallelized. Therefore, the similarity matrix can be constructed in parallel using the following four different ways.



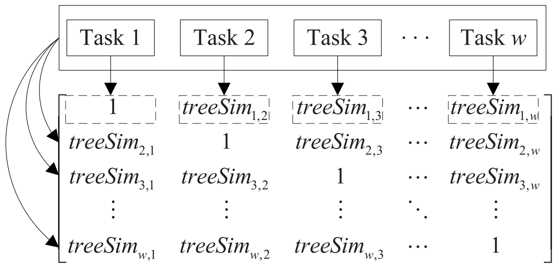
$$\begin{array}{ccccc|c} 1 & treeSim_{1,2} & treeSim_{1,3} & \cdots & treeSim_{1,w} & \text{Task 1} \\ treeSim_{2,1} & 1 & treeSim_{2,3} & \cdots & treeSim_{2,w} & \text{Task 2} \\ treeSim_{3,2} & treeSim_{3,2} & 1 & \cdots & treeSim_{3,w} & \text{Task 3} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ treeSim_{w,1} & treeSim_{w,2} & treeSim_{w,3} & \cdots & 1 & \text{Task } w \end{array}$$

FIGURE 4. The schematic diagram of method A for constructing a similarity matrix in parallel.

The basic idea of method A for constructing a similarity matrix in parallel is as follows: the outermost for-loop (see line 1 in Algorithm 1) is parallelized, i.e., the construction of a $w \times w$ similarity matrix is divided into w computational tasks which can be executed in parallel, and one row of the similarity matrix will be constructed after finishing each computational task, as shown in Fig. 4. The time complexity and space complexity of each computational task in method A are high. When there is a large amount of data to be processed, a lot of computing resources are required to execute each computational task, which results in that method A is not suitable for processing large-scale datasets. It is found that when the size of a dataset is more than 5 GB, an out-of-memory exception will occur if the similarity matrix is constructed according to method A on the experimental platform adopted in this paper.

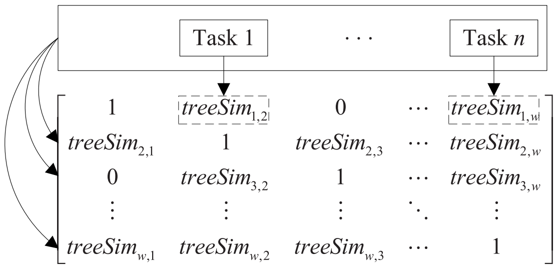
The basic idea of method B for constructing a similarity matrix in parallel is as follows: the fourth for-loop (see line 12 in Algorithm 1) is parallelized, i.e., the calculations of the similarity between one path of DT_a and each path of DT_b are divided into several computational tasks which can be executed in parallel, where $1 \leq a \leq w$ and $1 \leq b \leq w$. If DT_a has l_a paths, each computational task needs to be executed $w \sum_{a=1}^{w} l_a$ iteratively to complete the construction of a similarity matrix. Although the time complexity of each computational task of method B is low, $w \sum_{a=1}^{w} l_a$ RDDs are needed to be created to complete the construction of a similarity matrix. The creations of a large number of RDDs will bring huge extra overheads, thus method B is not suitable to construct a similarity matrix.

The basic idea of method C for constructing a similarity matrix in parallel is as follows: the second for-loop (see line 2 in Algorithm 1) is parallelized, i.e., the construction of a $1 \times w$ similarity matrix is divided into w independent



$$\begin{bmatrix} 1 & treeSim_{1,2} & treeSim_{1,3} & \cdots & treeSim_{1,w} \\ treeSim_{2,1} & 1 & treeSim_{2,3} & \cdots & treeSim_{2,w} \\ treeSim_{3,1} & treeSim_{3,2} & 1 & \cdots & treeSim_{3,w} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ treeSim_{w,1} & treeSim_{w,2} & treeSim_{w,3} & \cdots & 1 \end{bmatrix}$$

FIGURE 5. **The schematic diagram of method C for constructing a similarity matrix in parallel.**



$$\begin{bmatrix} 1 & treeSim_{1,2} & 0 & \cdots & treeSim_{1,w} \\ treeSim_{2,1} & 1 & treeSim_{2,3} & \cdots & treeSim_{2,w} \\ 0 & treeSim_{3,2} & 1 & \cdots & treeSim_{3,w} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ treeSim_{w,1} & treeSim_{w,2} & treeSim_{w,3} & \cdots & 1 \end{bmatrix}$$

FIGURE 6. **The schematic diagram of method D for constructing a similarity matrix in parallel.**

decision tree similarity calculation tasks, and one element of the $1 \times w$ similarity matrix will be obtained after finishing each calculation task, as shown in Fig. 5. The w decision tree similarity calculation tasks are evenly distributed to each worker node of a Spark platform, and one row of the $w \times w$ similarity matrix will be constructed by executing these w tasks in parallel on different worker nodes. Through w iterations, a $w \times w$ similarity matrix can be constructed. The method C overcomes the shortcomings of method A and method B. It not only is suitable for processing large-scale datasets, but also can avoid excessive extra overheads caused by creating a large number of RDDs.

According to the principles that the value of each diagonal element in the similarity matrix is 1 and the similarity of two decision trees whose root node similarity is lower than the specified threshold value is 0, the method D for constructing a similarity matrix in parallel makes some improvements to method C, as shown in Fig. 6. At first, the similarity between the root node of one decision tree and that of another decision tree is calculated quickly, and the elements whose values are 0 in the $1 \times w$ similarity matrix are obtained preliminarily. Then, the construction of a $1 \times w$ similarity matrix without elements whose values are 1 and 0 is divided into n $(n \leq w - 1)$ independent decision tree similarity calculation tasks, and the value and position of one element in the $1 \times w$ similarity matrix will be obtained after finishing each calculation task. Through w iterations, a $w \times w$ similarity matrix can be constructed. Compared with method C, method D not only reduces the unnecessary communication overhead, but also avoids the load imbalance between worker nodes. Therefore, method D is adopted to construct a similarity matrix.

The time complexity of method D for constructing a similarity matrix in parallel is $O(w(w/(pvt))lu\varphi)$, where p, v, and t are the number of worker nodes of a Spark platform, the number of executors in each worker node, and the number of tasks assigned to each executor, respectively. It is easy to see that the parallel sub-forest optimization can bring a significant performance improvement than the serial sub-forest optimization.

B. THE PARALLEL IMPLEMENTATION OF IRF ALGORITHM BASED ON SPARK PLATFORM

The process of parallel implementation of IRF algorithm based on Spark platform is shown in Fig. 7, and the specific steps are as follows.

Step 1. Initialize the Spark environment, and the eigenvectors are read from Hadoop Distributed File System (HDFS) to create an initial RDD. According to a certain proportion, the RDD is divided into three RDDs: *trainRDD*, *validationRDD*, and *testRDD*.

Step 2. Train an original RF model in parallel. The sampling with replacement is performed in parallel from all eigenvectors of *trainRDD* to generate k training subsets, and k decision trees are trained in parallel to obtain the original RF model.

Step 3. Traverse decision trees of the sub-forest in parallel to get all path information. First, *validationRDD* is used to evaluate all decision trees of the original RF model, and the classification accuracy of each decision tree is obtained. Second, k decision trees are filtered according to the classification accuracy, and w decision trees are obtained. Third, a sub-forest RDD *subforestRDD* with w decision trees is created. Fourth, all decision trees of *subforestRDD* are evenly distributed to each worker node, and the path information of each decision tree is obtained by traversing w decision trees in parallel using the *map* operator. Finally, all path information of the sub-forest is obtained by collecting the path information of each decision tree from each worker node using the *collect* operator.

Step 4. Construct a similarity matrix in parallel. First, the a-th $(1 \leq a \leq w)$ row of the similarity matrix is preliminarily constructed as follows. The value of the element in the a-th row and a-th column is set to 1. The similarity between the root node of DT_a and that of DT_b is calculated according to (1), if the similarity is 0, then $treeSim_{a,b} = 0$, where $1 \leq b \leq w$ and $b \neq a$. Second, a decision tree similarity calculation RDD *treeSimCalRDD* with n $(n \leq w - 1)$ tuples is constructed according to the obtained path information. Each tuple includes the path information of DT_a and DT_b and the serial number b of DT_b, which will be used to calculate the similarity between DT_a and DT_b. Third, n tuples of *treeSimCalRDD* are evenly distributed to each worker node, and n decision tree similarity calculation tasks are executed in parallel. Each calculation task processes one tuple, and the value and position of one element in the a-th row of the similarity matrix will be obtained after finishing each calculation task. Finally, the calculation results from each worker node are collected by the *collect* operator to complete the a-th row of the similarity matrix. A $w \times w$ similarity matrix can be constructed by repeating the above process w times.

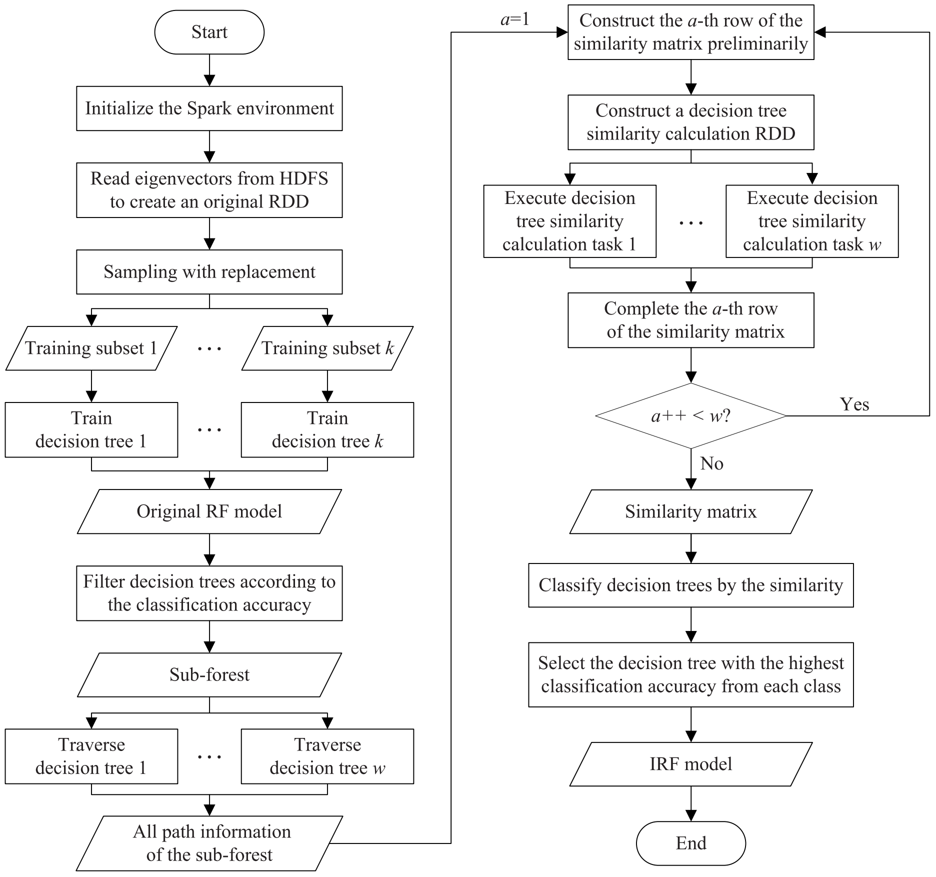

Start
Initialize the Spark environment
Read eigenvectors from HDFS to create an original RDD
Sampling with replacement
Training subset 1
Training subset k
Train decision tree 1
Train decision tree k
Original RF model
Filter decision trees according to the classification accuracy
Sub-forest
Traverse decision tree 1
Traverse decision tree w
All path information of the sub-forest
a=1
Construct the a-th row of the similarity matrix preliminarily
Construct a decision tree similarity calculation RDD
Execute decision tree similarity calculation task 1
Execute decision tree similarity calculation task w
Complete the a-th row of the similarity matrix
a++ < w?
Yes
No
Similarity matrix
Classify decision trees by the similarity
Select the decision tree with the highest classification accuracy from each class
IRF model
End

FIGURE 7. The process of parallel implementation of IRF algorithm based on Spark platform.

Step 5. Classify w decision trees by the similarity, and the decision tree with the highest classification accuracy is selected from each class to combine an IRF model. The effectiveness of the model is evaluated by *testRDD*.

C. THE OVERALL PROCESS OF FAULT DIAGNOSIS

The overall process of rolling bearing fault diagnosis based on Spark and IRF algorithm is depicted in Fig. 8. Firstly, multiple group sensors are used to collect the vibration signals of rolling bearings in real time. Secondly, during the data preprocessing stage, the original vibration signals are divided into many samples, and these standardized samples are decomposed by wavelet packet transform [42] to get eigenvectors, which are stored in HDFS. Thirdly, a part of eigenvectors are randomly chosen as the training sample data, which are divided into the training set, validation set, and test set according to a certain proportion. Fourthly, the original RF model is built by executing RF algorithm in parallel with the training set on a Spark platform. Fifthly, an IRF model (i.e., the rolling bearing fault diagnosis model) is obtained from the original RF model by executing the sub-forest optimization procedure in parallel with the validation set, and the effectiveness of IRF model is evaluated with the test set. Finally, the rolling bearing fault diagnosis model is executed in parallel to diagnose the data to be diagnosed on a Spark platform, and the fault diagnosis results are obtained.

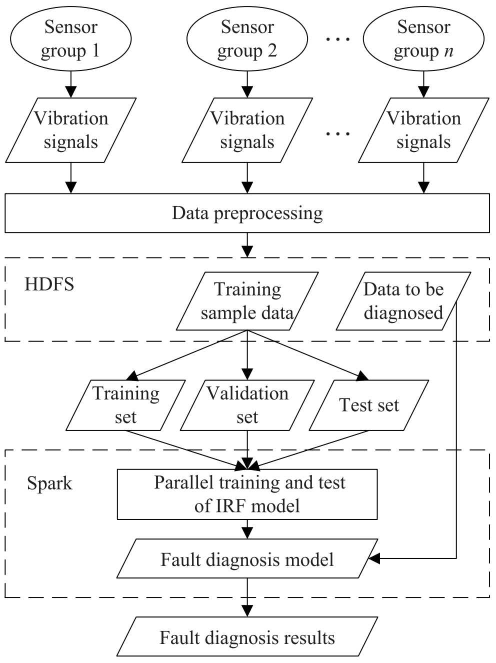


FIGURE 8. The overall process of rolling bearing fault diagnosis.

TABLE 1. Description of working conditions.

Fault Type	Fault Diameter (inch)	Motor Load (HP)
Normal	0	0 / 1 / 2 / 3
Inner race fault	0.007 / 0.014 / 0.021	0 / 1 / 2 / 3
Ball fault	0.007 / 0.014 / 0.021	0 / 1 / 2 / 3
Outer race fault	0.007 / 0.014 / 0.021	0 / 1 / 2 / 3

V. EXPERIMENTAL RESULTS AND ANALYSIS

A. EXPERIMENTAL DATASET

In order to verify the effectiveness of the proposed rolling bearing fault diagnosis method based on Spark and IRF algorithm, a series of experiments are carried out on the rolling bearing dataset [43] provided by Case Western Reserve University bearing data center. The dataset includes normal state data and fault state data, and all of them are collected by sensors deployed at the base end, drive end, and fan end at 12 kHz and 48 kHz sampling frequencies under different working conditions listed in Table 1. The test rig of rolling bearing is displayed in Fig. 9.

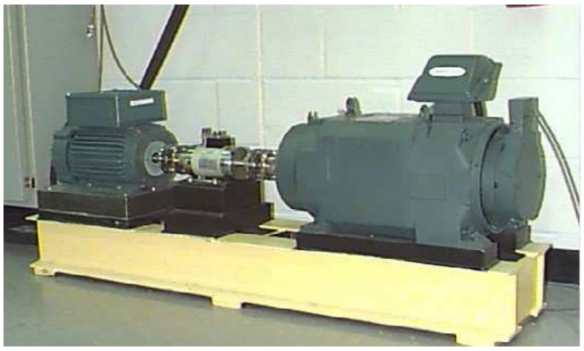


FIGURE 9. The test rig of rolling bearing.

Since the original rolling bearing dataset is small, it is difficult to effectively evaluate the parallel performance of the proposed rolling bearing fault diagnosis method. Therefore, the overlapping sampling method [44] is exploited to enhance the original vibration data, as shown in Fig. 10. After preprocessing the enhanced vibration data, three different size datasets are obtained, as listed in Table 2.

TABLE 2. Experimental datasets.

Dataset Name	Dataset Size (GB)
Dataset 1	5.6
Dataset 2	11.2
Dataset 3	20.4

Considering the training efficiency of the model and the limitation of hardware resources of the experimental platform used in this paper, the size of the dataset that can be effectively supported can reach up to 20.4 GB. To support a larger dataset, the hardware resources of the experimental platform need to be increased, or a large dataset is sampled to get a small similar subset for model training.

B. EXPERIMENTAL SETTINGS

The experimental platform includes one master node and four worker nodes. The master node consists of a quad-core Intel Xeon E3-1225 v5 CPU at 3.3 GHz and 32 GB main memory, and each worker node consists of an eight-core Intel Core i7-9700K CPU at 3.6 GHz and 32 GB main memory. The software configurations of the experimental platform are as follows: CentOS 8.1 operating system, Hadoop 3.2, Spark 3.0, and Python 3.7.

In the training of the rolling bearing fault diagnosis model based on Spark and IRF algorithm, the parameter settings of IRF algorithm are listed in Table 3.

TABLE 3. Parameter settings of IRF algorithm.

Parameter Name	Parameter Value
maxDepth	17
maxBins	45
impurity	entropy
similarityFilter	0.6
minInstancePerNode	200

The parameter of *maxDepth* represents the maximum depth of a decision tree. The larger the value of *maxDepth* is, the stronger the diagnosis ability of the fault diagnosis model is, but the longer the model training time is. Since rolling bearings usually operate in complicated and various working conditions, it is difficult to accurately distinguish different types of rolling bearing faults, thus a larger value of *maxDepth* should be adopted to get a fault diagnosis model with stronger diagnosis ability.

The parameter of *maxBins* denotes the maximum number of boxes when feature boxing is used for decision tree training. For the massive data or high-dimensional categorical features, if the value of *maxBins* is too large, the computation and communication overheads of model training will be high.

The parameter of *impurity* is used to specify the impurity function adopted in decision tree training. The commonly used impurity functions include entropy and Gini index. Different decision tree algorithms use different impurity functions: ID3 and C4.5 generally use entropy, whereas CART generally uses Gini index or entropy. The selection of impurity functions should be done by analyzing the actual data and the underlying decision tree algorithm. In this experiment, entropy is chosen as the impurity function, which can achieve a better classification effect.

The parameter of *similarityFilter* is used to specify the similarity threshold value, which generally ranges from 0.5 to 0.9. The value of *similarityFilter* can be determined according to the decision tree similarity matrix, if the similarities of decision trees are high, a larger value of *similarityFilter* can be selected; otherwise, a smaller value is more suitable. Since the vibration signals of rolling bearing collected under complex working conditions usually are complicated, the similarities of decision trees are relatively low, and therefore a smaller value of *similarityFilter* should be set to eliminate the decision trees which are prone to repeated voting.

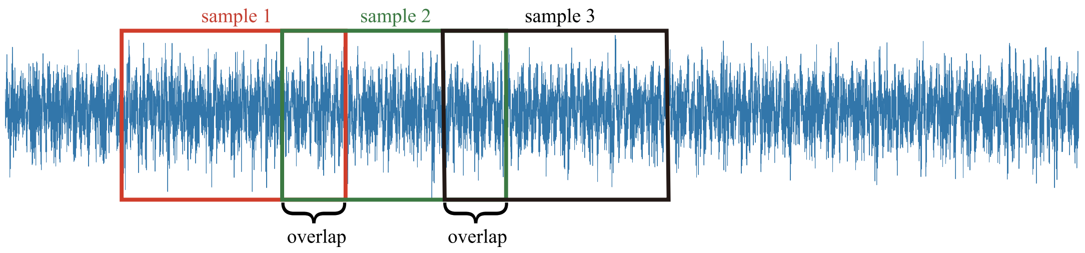


FIGURE 10. The schematic diagram of overlapping sampling.

The parameter of *minInstancesPerNode* represents the minimum number of instances each child node must have after the decision tree is split. If the number of training samples is large, a larger value of *minInstancesPerNode* should be set to avoid over-fitting and reduce the model training time. If the number of training samples is small, a smaller value of *minInstancesPerNode* should be set to avoid underfitting. Since a large-scale vibration data would usually be produced during the long-term operation of rolling bearings, it is necessary to set this parameter reasonably to avoid too much computational overhead for model training and ensure a good generalization ability and high diagnosis accuracy.

C. MODEL TRAINING AND VERIFICATION

In this experiment, firstly, the proposed rolling bearing fault diagnosis model based on Spark and IRF algorithm (Spark-IRFA) is trained and tested. Secondly, the rolling bearing fault diagnosis model implemented by RF algorithm provided by Spark MLlib [45] (Spark-RFA) is trained and tested. Finally, the rolling bearing fault diagnosis model implemented by serial RF algorithm provided by Python sklearn library [46] (Python-RFA) is trained and tested. Dataset 1 is used in the training and test of three models, and it is divided into the training set, validation set, and test set according to the ratio of 6:2:2.

Fig. 11 presents the comparison of fault diagnosis accuracies obtained by different methods under different scale forests. As can be seen from Fig. 11, with the number of decision trees in the random forest increasing from 20 to 200, the fault diagnosis accuracies obtained by Python-RFA, Spark-RFA, and Spark-IRFA are increased from 95.85%, 95.69%, and 95.66% to 96.43%, 96.51%, and 97.22%, respectively. The results demonstrate that the increase of forest scale plays an important role in improving the fault diagnosis accuracy. However, when the number of decision trees in the random forest is increased to 100, the fault diagnosis accuracies obtained by the three methods tend to be stable, thus an appropriate increase in the number of trees is helpful to improve the fault diagnosis accuracy.

As shown in Fig. 11, Spark-IRFA achieves higher fault diagnosis accuracy when the forest scale is large. For example, when the number of decision trees is 200, compared with Python-RFA and Spark-RFA, the fault diagnosis accuracy of Spark-IRFA is increased by 0.79% and 0.71%, respectively. This is because the proposed IRF algorithm based on

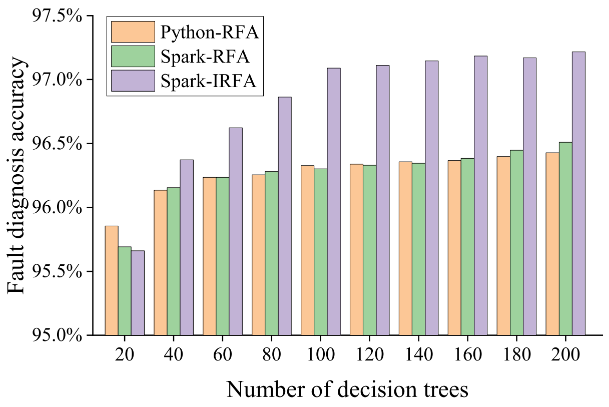


FIGURE 11. Comparison of fault diagnosis accuracies obtained by different methods under different scale forests.

sub-forest optimization can eliminate the decision trees with low classification accuracy and those prone to repeated voting in the original RF, which effectively reduces the probabilities of wrong classification and repeated voting of decision trees in the random forest. It can also be seen from Fig. 11, when the number of decision trees is 20, the fault diagnosis accuracy of Spark-IRFA is lower than that of Python-RFA and Spark-RFA. This is because the sub-forest optimization has a certain impact on the generalization of the model when the forest scale is small, thus the recommended number of decision trees is set to more than 60 when Spark-IRFA is adopted.

In order to evaluate the effect of different size datasets on the diagnosis accuracy of the proposed rolling bearing fault diagnosis method, the three datasets listed in Table 2 are utilized to test Spark-IRFA respectively, and the number of decision trees is set to 100. As depicted in Fig. 12, for Dataset 1, Dataset 2, and Dataset 3, the fault diagnosis accuracies obtained by Spark-IRFA are 97.09%, 97.38%, and 98.12%, respectively. With the increase of dataset size, the fault diagnosis accuracy is also increased. Specifically, the diagnosis accuracy obtained with Dataset 3 of 20.4 GB is 1.03% and 0.74% higher than that obtained with Dataset 1 of 5.6 GB and that obtained with Dataset 2 of 11.2 GB, respectively. This is because the larger-scale dataset contains more different kinds of training samples, and Spark-IRFA can utilize more abundant features to construct feature subspaces, so as to train a model with higher fault diagnosis accuracy.

D. PERFORMANCE ANALYSIS OF MODEL TRAINING

In order to effectively evaluate the performance of parallel training of the proposed rolling bearing fault diagnosis model,

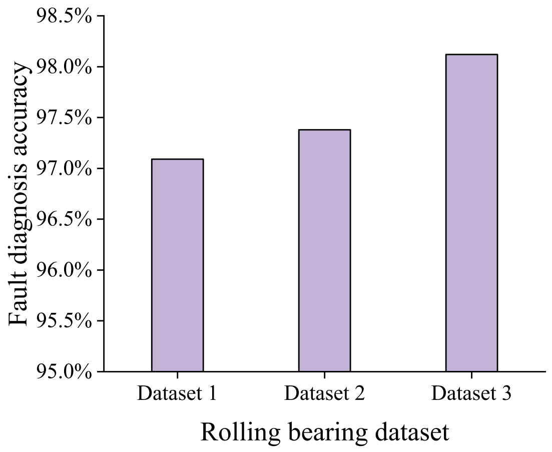


FIGURE 12. The fault diagnosis accuracies obtained by Spark-IRFA and different size datasets.

TABLE 4. The time spent on training the fault diagnosis model using Spark-IRFA.

Dataset	Model Training Time (minutes)			
	1 Worker Node	2 Worker Nodes	3 Worker Nodes	4 Worker Nodes
Dataset 1	195.3	101.1	71.6	57.1
Dataset 2	413.2	178.2	136.6	103.6
Dataset 3	751.3	384.3	240.7	194.6

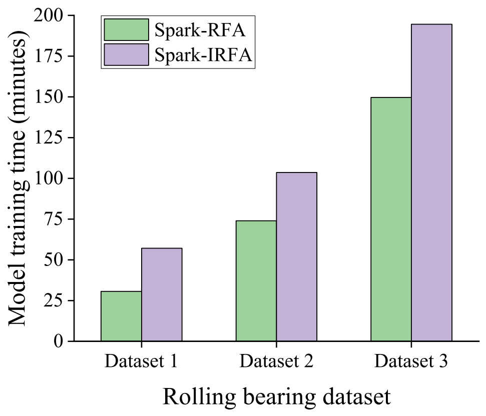


FIGURE 13. Comparison of the time spent on training fault diagnosis models using Spark-RFA and Spark-IRFA under different size datasets.

TABLE 5. The time spent on sub-forest optimization in the process of training the fault diagnosis model using Spark-IRFA.

Dataset	Sub-Forest Optimization Time (minutes)			
	1 Worker Node	2 Worker Nodes	3 Worker Nodes	4 Worker Nodes
Dataset 1	91.7	47.2	32.3	26.4
Dataset 2	106.9	54.3	36.9	29.6
Dataset 3	169.8	85.8	57.8	44.9

Spark-RFA and Spark-IRFA are used to train the rolling bearing fault diagnosis models for Dataset 1, Dataset 2, and Dataset 3 on the Spark platforms with different number of worker nodes, respectively, where the number of decision trees is set to 100 and the training set accounts for 60% of the dataset.

Table 4 presents the time spent on training fault diagnosis models with Spark-IRFA and different size datasets on different scale clusters. For three different size datasets, the training time of fault diagnosis models obtained with 2, 3, and 4 worker nodes is 51.48%, 66.08%, and 73.26% lower than that obtained with a single worker node, respectively. The results suggest that the performance of fault diagnosis model training is gradually improved with the expansion of Spark cluster scale. This is because the increase of the number of worker nodes can not only provide more computing resources to train the model in parallel, but also provide more memory resources to store intermediate results to speed up the computation. Therefore, when processing large-scale rolling bearing data, the speed of parallel training of fault diagnosis model can be effectively improved by increasing the number of worker nodes.

When 4 worker nodes are used on the Spark platform, the time spent on training fault diagnosis models using Spark-RFA and Spark-IRFA under three different size datasets is shown in Fig. 13. For Dataset 1, Dataset 2, and Dataset 3, the time spent on training fault diagnosis models using Spark-IRFA is 85.98%, 40.01%, and 29.99% higher than that using Spark-RFA, respectively. For these three different datasets, Spark-IRFA has a longer model training time than Spark-RFA. The reason is that Spark-IRFA needs to train an original RF model and then conducts the sub-forest optimization on this model. In other words, compared with Spark-RFA, Spark-IRFA adds the extra sub-forest optimization time.

Table 5 shows the time spent on sub-forest optimization in the process of training fault diagnosis models with Spark-IRFA and different size datasets on different scale clusters. For three different size datasets, the time consumed by sub-forest optimization when using 2, 3, and 4 worker nodes is reduced by 49.23%, 65.41%, and 72.36% on average compared with that when using a single worker node, respectively. The results demonstrate that the more worker nodes are used, the less time is needed for sub-forest optimization, because the Spark platform can provide more computing resources for parallel execution of sub-forest optimization.

Fig. 14 presents the percentage of the time spent on original RF model training and sub-forest optimization in the process of training fault diagnosis models with Spark-IRFA and three different size datasets on the Spark platform with 4 worker nodes. Obviously, with the increase of dataset size, the percentage of sub-forest optimization in the total model training time decreases gradually, which means that the impact of sub-forest optimization on the performance of model training decreases gradually. Therefore, the larger the dataset size, the smaller the performance gap of model training between Spark-IRFA and Spark-RFA.

When a large-scale dataset is used for fault diagnosis model training, the memory space requirement of sub-forest optimization is small. In the process of executing the sub-forest optimization, the intermediate results are stored in memory, and there is no disk I/O overhead. However, the training of

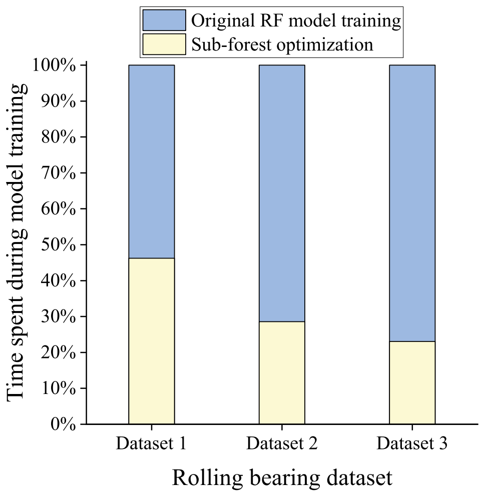


FIGURE 14. The percentage of the time spent on original RF model training and sub-forest optimization in the process of training the fault diagnosis model using Spark-IRFA.

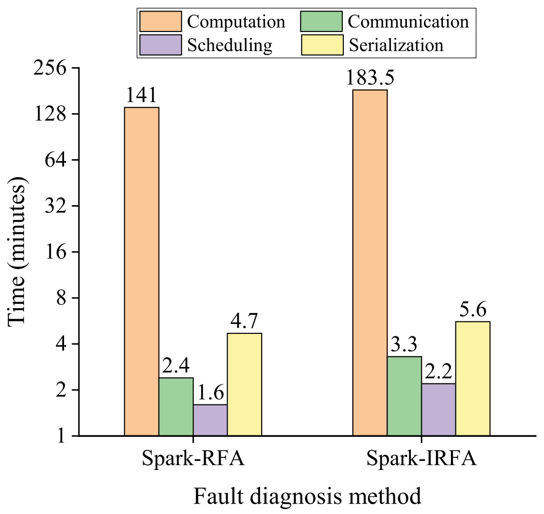


FIGURE 15. The computation time and runtime overheads of Spark-RFA and Spark-IRFA for training the fault diagnosis models using Dataset 3 on the Spark platform with 4 worker nodes.

original RF model requires more memory space. If the cluster scale is small, most of the intermediate results might be spilled onto disk, which will greatly reduce the training speed of original RF model. For example, most of the intermediate results produced during the training of original RF model have been spilled onto disk for Dataset 2 and Dataset 3 on the Spark platform with a single worker node. Therefore, for large-scale datasets, the scale of the Spark cluster can be expanded to increase the available memory space, so as to reduce the impact of a large amount of disk I/O on the speed of model training.

In this paper, the model training time consists of the computation time and runtime overhead. In order to make a more comprehensive performance evaluation of model training, the runtime overhead of model training is analyzed. In this experiment, the runtime overhead mainly includes the communication overhead, scheduling overhead, and serialization overhead. Fig. 15 presents the computation time and runtime overhead of Spark-IRFA for training the fault diagnosis model using Dataset 3 on the Spark platform with 4 worker nodes. It can be seen from Fig. 15 that the runtime overhead accounts for 5.70% of the model training time of Spark-IRFA. Specifically, the communication overhead, scheduling overhead, and serialization overhead account for 1.69%, 1.13%, and 2.88% of the model training time respectively. The results show that the runtime overhead of Spark-IRFA is only a tiny part of the model training time. The low runtime overhead is mainly attributed to the following two points. The first point, the communication optimization problem and the number and size of RDDs are considered in the design of Spark-IRFA, thus the communication overhead and serialization overhead are greatly reduced. The second point, Spark provides the efficient task scheduling and resource management mechanism, thus the scheduling overhead is very low.

As shown in Fig. 15, the runtime overheads of Spark-RFA and Spark-IRFA are 8.7 minutes and 11.1 minutes respectively. Compared with Spark-RFA, the runtime overhead of Spark-IRFA is increased by 27.59%. The reason is that Spark-RFA only needs to perform the RF model training, but Spark-IRFA needs to conduct the sub-forest optimization in addition to the original RF model training. For Spark-IRFA, the runtime overhead generated from the original RF model training and that generated from the sub-forest optimization account for 4.47% and 1.23% of the model training time respectively. The results show that the runtime overhead brought by the sub-forest optimization has only a marginal effect on the performance of model training.

E. PERFORMANCE ANALYSIS OF FAULT DIAGNOSIS

In order to effectively analyze the diagnosis performance of the proposed rolling bearing fault diagnosis method, Python-RFA, Spark-RFA, and Spark-IRFA are adopted to train rolling bearing fault diagnosis models on the Spark platform for three different size datasets, and then the trained models are used for fault diagnosis, where Python-RFA only runs on a single worker node of the Spark platform. In addition, to better analyze the impact of large-scale datasets on the performance of fault diagnosis, all the data in each dataset will be diagnosed.

Table 6 presents the time spent on diagnosing three different size datasets using the fault diagnosis model based on Spark-IRFA on the Spark platforms with different number of worker nodes. As shown in Table 6, the more the worker nodes employed on a Spark platform, the less the diagnosis time of Spark-IRFA. For example, for Dataset 3, the fault diagnosis time obtained with 2, 3, and 4 worker nodes is reduced by 44.07%, 70.34%, and 73.73% than that obtained with a single worker node, respectively. The results reveal that Spark-IRFA can effectively improve the diagnosis performance by employing multiple worker nodes to perform fault diagnosis in parallel.

TABLE 6. The fault diagnosis time of Spark-IRFA.

Dataset	Fault Diagnosis Time (minutes)			
	1 Worker Node	2 Worker Nodes	3 Worker Nodes	4 Worker Nodes
Dataset 1	2.5	1.5	1.1	0.9
Dataset 2	5.2	3.0	2.1	1.8
Dataset 3	11.8	6.6	3.5	3.1

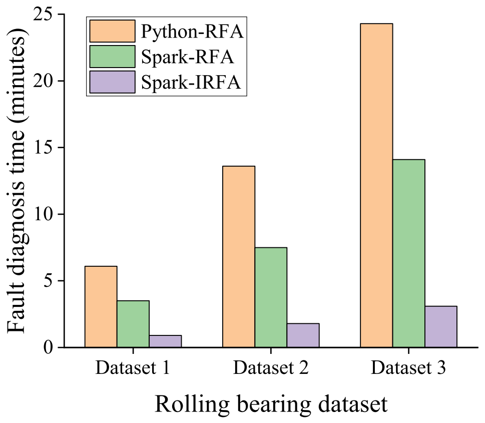


FIGURE 16. Comparison of the fault diagnosis time of Python-RFA, Spark-RFA, and Spark-IRFA under different size datasets.

Fig. 16 illustrates the fault diagnosis time of Python-RFA, Spark-RFA, and Spark-IRFA under three different size datasets. Compared with Python-RFA, both Spark-RFA and Spark-IRFA achieve better performance. For example, when the size of the data to be diagnosed reaches up to 20.4 GB (i.e., the size of Dataset 3), the diagnosis time of Python-RFA is 24.3 minutes while that of Spark-IRFA is only 3.1 minutes. This is because Python-RFA can only use a single CPU core to perform fault diagnosis serially, while both Spark-RFA and Spark-IRFA can utilize multiple CPU cores of multiple worker nodes to perform fault diagnosis in parallel. As shown in Fig. 16, the fault diagnosis time of Spark-IRFA is significantly shorter than that of Spark-RFA. For Dataset 1, Dataset 2, and Dataset 3, the fault diagnosis time of Spark-IRFA is reduced by 74.29%, 76.00%, and 78.01% than that of Spark-RFA, respectively. This is mainly due to the sub-forest optimization is performed on the original RF model in Spark-IRFA, and the computational complexity of the final obtained RF model (i.e., IRF model) is far less than that of the original RF model. Thus, the proposed fault diagnosis method can quickly diagnose a large-scale rolling bearing vibration data.

To further analyze the effect of the number of decision trees on the performance of fault diagnosis, Spark-RFA and Spark-IRFA are adopted to train rolling bearing fault diagnosis models for Dataset 3 under different scale forests, and then the trained models are used to diagnose 20.4 GB of data. Fig. 17 shows the fault diagnosis time of Spark-RFA and Spark-IRFA under five different scale forests. As the number of decision trees increases from 20 to 100, the diagnosis time of Spark-RFA increases from 1.3 minutes to 14.1 minutes, while the diagnosis time of Spark-IRFA increases from 1.0

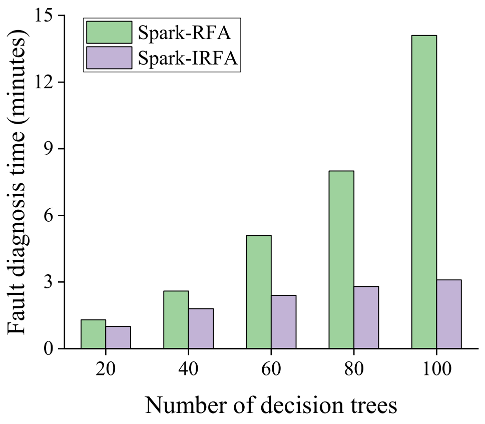


FIGURE 17. Comparison of the fault diagnosis time of Spark-RFA and Spark-IRFA under different scale forests.

minutes to 3.1 minutes, which reveals that the number of decision trees has a great impact on the performance of fault diagnosis. The more the decision trees used in model training are, the higher the diagnosis accuracy is, and the longer the training time and diagnosis time are. Therefore, it is necessary to set the number of decision trees reasonably.

As shown in Fig. 17, with the increase of the number of decision trees, the diagnosis performance gap between Spark-RFA and Spark-IRFA becomes larger and larger. When the number of decision trees is set to 20, 40, 60, 80, and 100 respectively, the fault diagnosis time of Spark-IRFA is reduced by 23.08%, 30.77%, 52.94%, 65.00%, and 78.01% than that of Spark-RFA respectively, which is mainly due to the sub-forest optimization significantly reduces the number of decision trees in the original RF model. Generally speaking, the more the decision trees used in the training of original RF model are, the larger the reduction ratio of forest scale obtained by sub-forest optimization is.

F. ANALYSIS OF THE IMPACT OF NOISE ON THE FAULT DIAGNOSIS ACCURACY

In order to analyze the impact of noise on the fault diagnosis accuracy, firstly, 20%, 40%, 60%, 80%, and 100% additive white Gaussian noise are added to the vibration signals of rolling bearing respectively; secondly, the vibration signals with different noise intensity are decomposed by wavelet packet transform to get eigenvectors, and five different datasets with the same size as Dataset 1 are obtained; finally, Spark-RFA and Spark-IRFA are used to train and test the fault diagnosis models for the five different datasets.

Fig. 18 presents the comparison of fault diagnosis accuracies obtained under different noise environments. As seen in Fig. 18, the fault diagnosis accuracy of Spark-IRFA can reach over 91.15% when the percentage of added noise is less than or equal to 20%. However, the fault diagnosis accuracy of Spark-IRFA decreases gradually along with the increase of the percentage of added noise. When the percentage of added noise increases from 40% to 100%, the fault diagnosis accuracy of Spark-IRFA decreases from 85.13% to 79.56%. The

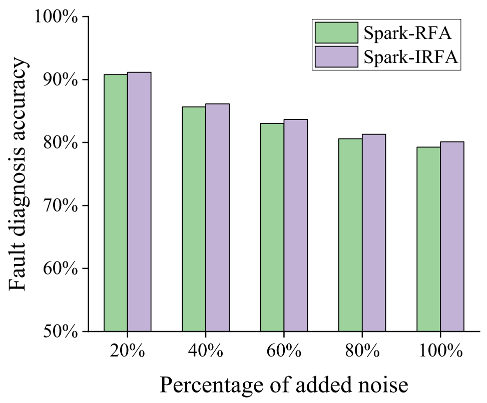


FIGURE 18. Comparison of fault diagnosis accuracies obtained under different noise environments.

results demonstrate that Spark-IRFA is still effective when the vibration signals contain a small amount of noise, but the fault diagnosis accuracy of Spark-IRFA will be greatly affected when the vibration signals contain a lot of noise. To improve the anti-noise ability of Spark-IRFA, some feature extraction methods [18], [47] can be adopted to more accurately extract fault features under noise environment, and the features and labels can be treated as probability distribution functions [48] to improve the robustness of RF.

Compared with Spark-RFA, the fault diagnosis accuracy of Spark-IRFA is increased by 0.76% on average for five datasets with different percentages of added noise. The reason is that the classification accuracies of some decision trees are lower due to noise interference, and the sub-forest optimization can effectively eliminate these decision trees with lower classification accuracy.

G. DIAGNOSIS EFFECT ANALYSIS FOR IMBALANCED DATASETS

In order to better analyze the diagnosis effect of the proposed fault diagnosis method for imbalanced datasets, Spark-RFA and Spark-IRFA are used to train and test the fault diagnosis models for four different imbalanced datasets described in Table 7, and each imbalanced dataset is divided into the training set, validation set, and test set according to the ratio of 6:2:2. To make a more comprehensive analysis, two different evaluation indexes are adopted, including the accuracy and macro AUC.

Table 8 presents the comparison of fault diagnosis effect of Spark-RFA and Spark-IRFA for four different imbalanced datasets. As shown in Table 8, Spark-RFA and Spark-IRFA achieve good diagnosis effect for the dataset with a smaller imbalanced ratio, but they perform poorly for the dataset with a larger imbalanced ratio. The results show that the imbalance degree of a dataset will have a great impact on the fault diagnosis effect of the proposed fault diagnosis method. For the dataset with a very few fault samples, it is realistic to generate the training data being compatible with intrinsic natural fault features [49], therefore some sampling methods [50], [51] and generative adversarial networks [52], [53] can

TABLE 7. Four different imbalanced datasets.

Imbalanced Dataset	Imbalanced Ratio	The Number of Samples			
		Normal State	Inner Race Fault	Ball Fault	Outer Race Fault
Dataset A	100:1	5000	50	50	50
Dataset B	50:1	5000	100	100	100
Dataset C	20:1	5000	250	250	250
Dataset D	10:1	5000	500	500	500

be adopted to generate more fault samples to improve the fault diagnosis effect. It can also be seen from Table 8 that the fault diagnosis effect of Spark-IRFA is slightly better than that of Spark-RFA, which benefits from the sub-forest optimization adopted in Spark-IRFA.

TABLE 8. Comparison of fault diagnosis effect of Spark-RFA and Spark-IRFA for different imbalanced datasets.

Imbalanced Dataset	Fault Diagnosis Accuracy		Macro AUC	
	Spark-RFA	Spark-IRFA	Spark-RFA	Spark-IRFA
Dataset A	69.42%	70.55%	0.8479	0.8534
Dataset B	82.43%	83.21%	0.9166	0.9217
Dataset C	92.96%	93.67%	0.9642	0.9668
Dataset D	96.23%	96.88%	0.9832	0.9845

H. COMPARISON WITH OTHER FAULT DIAGNOSIS METHODS

In order to further verify the effectiveness of the proposed rolling bearing fault diagnosis method, which is compared with QPSO-BPNN [6], LeNet-5 [8], VGG-16 [12], ResNet-18 [13], and AlexNet [14]. For the sake of fairness, QPSO-BPNN, LeNet-5, VGG-16, ResNet-18, and AlexNet are implemented on the Spark platform using SparkTorch. In this experiment, Spark-IRFA, Spark-QPSO-BPNN, Spark-LeNet-5, Spark-VGG-16, Spark-ResNet-18, and Spark-AlexNet are used to train and test six different rolling bearing fault diagnosis models on the Spark platform with 4 worker nodes. For Spark-IRFA and Spark-QPSO-BPNN, Dataset 3 is used as the experimental dataset and is divided into training set, validation set, and test set according to the ratio of 6:2:2. For Spark-LeNet-5, Spark-VGG-16, and Spark-ResNet-18, the vibration data with the same size as Dataset 3 are converted into 64 $\times$ 64 pixel gray images. For Spark-AlexNet, the vibration data with the same size as Dataset 3 are converted into 224 $\times$ 224 $\times$ 3 pixel time-frequency images by the continuous wavelet transform. The dataset composed of gray images or time-frequency images is also divided into training set, validation set, and test set according to the ratio of 6:2:2.

During the stage of model training, the network structures and super-parameter settings of Spark-QPSO-BPNN, Spark-LeNet-5, Spark-VGG-16, Spark-ResNet-18, and Spark-AlexNet come from [6], [8], [12], [13], and [14], respectively. The key super-parameters of neural networks mainly include batch size, learning rate, and momentum. The settings of three key super-parameters for different neural networks used in this experiment are listed in Table 9. The batch

TABLE 9. The settings of key super-parameters for different neural networks.

Neural Network	Batch Size	Learning Rate	Momentum
Spark-QPSO-BPNN	4096	0.003	0.9
Spark-LeNet-5	256	0.002	0.8
Spark-VGG-16	1024	0.003	0.8
Spark-ResNet-18	1024	0.002	0.9
Spark-AlexNet	128	0.001	0.9

TABLE 10. Comparison of different fault diagnosis methods based on Spark platform.

Fault Diagnosis Method	Fault Diagnosis Accuracy	Model Training Time (minutes)	Fault Diagnosis Time (minutes)
Spark-QPSO-BPNN	98.65%	327.3	5.6
Spark-LeNet-5	99.71%	584.9	16.7
Spark-VGG-16	99.87%	5637.7	43.5
Spark-ResNet-18	99.92%	1921.3	28.4
Spark-AlexNet	99.81%	688.7	20.9
Spark-IRFA	98.12%	194.6	3.1

size is sensitive to the computational time and convergence speed. With the increases of batch size, the computational time of each epoch is reduced and the convergence speed is increased, but the number of epochs needed to achieve the same accuracy is increased. The learning rate is sensitive to the convergence performance, if it is too large, the model may not converge to a global minimum; if it is too small, the model may converge to a local minimum and the convergence speed will become very slow. The momentum is sensitive to the convergence speed, and a larger value of momentum is helpful to improve the convergence speed. The number of epochs is set to 70, 60, 50, 50, and 40 for Spark-QPSO-BPNN, Spark-LeNet-5, Spark-VGG-16, Spark-ResNet-18, and Spark-AlexNet respectively, which can ensure a higher fault diagnosis accuracy with a fewer training time. Moreover, the key parameters of QPSO algorithm adopted in Spark-QPSO-BPNN include the shrinkage factor and the number of particles, which are set to 0.8 and 100 respectively.

During the stage of fault diagnosis, Spark-IRFA and Spark-QPSO-BPNN diagnose all the data of Dataset 3, Spark-LeNet-5, Spark-VGG-16, and Spark-ResNet-18 diagnose all the data of the gray image dataset, and Spark-AlexNet diagnoses all the data of the time-frequency image dataset.

Table 10 presents the diagnosis accuracies, model training time, and fault diagnosis time of six different fault diagnosis methods based on Spark platform. From the perspective of fault diagnosis accuracy, Spark-QPSO-BPNN, Spark-LeNet-5, Spark-VGG-16, Spark-ResNet-18, and Spark-AlexNet are better than Spark-IRFA. Specifically, the diagnosis accuracy of Spark-QPSO-BPNN is 0.53% higher than that of Spark-IRFA, this is because the initial weights and thresholds of BPNN are optimized by QPSO algorithm and multiple well-trained BPNN models are fused by DS evidence theory. The diagnosis accuracy of Spark-LetNet-5 is 1.59% higher than that of Spark-IRFA, this is because the modified LetNet-5 with a deep network structure and zero-padding can effectively extract the features of 2-D gray images. Spark-VGG-16, Spark-ResNet-18, and Spark-AlexNet also achieve higher diagnosis accuracies than Spark-IRFA, this is because they have more powerful feature extraction abilities by constructing a deeper network structure, thus they can automatically mine more useful features from the massive training data, which is helpful to improve the fault diagnosis accuracy.

As seen in Table 10, Spark-IRFA has the fastest model training speed and fault diagnosis speed among six different fault diagnosis methods. Compared with Spark-QPSO-BPNN, Spark-LeNet-5, Spark-VGG-16, Spark-ResNet-18, and Spark-AlexNet, the model training speed of Spark-IRFA is increased by 0.68×, 2.01×, 27.97×, 8.87×, and 2.54× respectively, and the fault diagnosis speed of Spark-IRFA is increased by 0.81×, 4.39×, 13.03×, 8.16×, and 5.74× respectively. Facing the large-scale rolling bearing vibration data, the performance of Spark-QPSO-BPNN, Spark-LeNet-5, Spark-VGG-16, Spark-ResNet-18, and Spark-AlexNet can be greatly improved by using GPU with strong parallel computing ability (such as Tesla V100), but GPU is a kind of expensive computing resource. Therefore, the proposed fault diagnosis method not only has good fault diagnosis accuracy but also has fast model training speed and fault diagnosis speed on the Spark cluster composed of cheap computing resources.

VI. CONCLUSION

In this paper, an efficient rolling bearing fault diagnosis method based on Spark and IRF algorithm is proposed. The decision trees with low classification accuracy and those prone to repeated voting in the original RF are effectively eliminated by sub-forest optimization, and an improved RF with faster classification speed and higher classification accuracy is obtained. The sub-forest optimization significantly reduces the diagnosis time of rolling bearing fault diagnosis model based on IRF algorithm, and improves its diagnosis accuracy to a certain extent. The parallelization of the proposed IRF algorithm is realized on the Spark platform, which mainly includes the parallelizations of the following three procedures: the training of original RF model, the traversal of the sub-forest, and the construction of a decision tree similarity matrix. In the face of large-scale rolling bearing datasets, the training speed and diagnosis speed of the fault diagnosis model are significantly improved by executing IRF algorithm efficiently and parallelly on the Spark platform. The effectiveness of the proposed rolling bearing fault diagnosis method is verified through a large number of experiments, and the results show that the proposed method not only achieves a higher model training speed and fault diagnosis speed, but also can diagnose multiple different rolling bearing faults and achieve a better fault diagnosis accuracy. For a 20.4 GB of rolling bearing dataset, compared with the non-parallel RF algorithm (i.e., Python-RFA) and the parallel RF algorithm provided by Spark (i.e., Spark-RFA), the fault diagnosis speed of the proposed Spark-IRFA is increased by 6.84 times and 3.55 times, and the fault diagnosis accuracy of Spark-IRFA reaches up to 98.12%.

In real-life industrial environments, the collected rolling bearing vibration data usually contain noise and may be imbalanced. In order to improve the anti-noise ability of the proposed fault diagnosis method, the future work will explore how to effectively combine Spark-IRFA with a new feature extraction method which can more accurately extract fault features under noise environment. Facing the imbalanced dataset with a very few fault samples, how to generate more fault samples to improve the fault diagnosis accuracy needs to be explored. In addition, during the training of the fault diagnosis model, the sub-forest optimization will inevitably increase the model training time, and it is difficult to find the optimal sub-forest. Therefore, another important work is to explore how to find the near-optimal sub-forest and minimize the time spent on sub-forest optimization by using a suitable meta-heuristic algorithm in the future.

REFERENCES

[1] I. El-Thalji and E. Jantunen, "A summary of fault modelling and predictive health monitoring of rolling element bearings," *Mech. Syst. Signal Process.*, vols. 60–61, pp. 252–272, Aug. 2015.

[2] Y. Qin, "A new family of model-based impulsive wavelets and their sparse representation for rolling bearing fault diagnosis," *IEEE Trans. Ind. Electron.*, vol. 65, no. 3, pp. 2716–2726, Mar. 2018.

[3] Y. Li, Y. Yang, X. Wang, B. Liu, and X. Liang, "Early fault diagnosis of rolling bearings based on hierarchical symbol dynamic entropy and binary tree support vector machine," *J. Sound Vib.*, vol. 428, pp. 72–86, Dec. 2018.

[4] Q. Zhou, Y. Lv, L. Tu, M. Wang, and S. Li, "Study of fault diagnosis for rolling bearing based on clustering algorithms," in *Proc. 5th Int. Conf. Control Robot. Eng. (ICCRE)*, Osaka, Japan, Apr. 2020, pp. 58–62.

[5] X. Chen, Z. Yang, and W. Lou, "Fault diagnosis of rolling bearing based on the permutation entropy of VMD and decision tree," in *Proc. 3rd Int. Conf. Electron. Inf. Technol. Comput. Eng. (EITCE)*, Xiamen, China, Oct. 2019, pp. 1911–1915.

[6] L. Wan, H. Li, Y. Chen, and C. Li, "Rolling bearing fault prediction method based on QPSO-BP neural network and Dempster–Shafer evidence theory," *Energies*, vol. 13, no. 5, p. 1094, Mar. 2020.

[7] J. Xie, G. Du, C. Shen, N. Chen, L. Chen, and Z. Zhu, "An end-to-end model based on improved adaptive deep belief network and its application to bearing fault diagnosis," *IEEE Access*, vol. 6, pp. 63584–63596, 2018.

[8] L. Wen, X. Li, L. Gao, and Y. Zhang, "A new convolutional neural network-based data-driven fault diagnosis method," *IEEE Trans. Ind. Electron.*, vol. 65, no. 7, pp. 5990–5998, Jul. 2018.

[9] L. Wen, X. Li, X. Li, and L. Gao, "A new transfer learning based on VGG-19 network for fault diagnosis," in *Proc. IEEE 23rd Int. Conf. Comput. Supported Cooperat. Work Design (CSCWD)*, Porto, Portugal, May 2019, pp. 205–209.

[10] L. Wen, X. Li, and L. Gao, "A transfer convolutional neural network for fault diagnosis based on ResNet-50," *Neural Comput. Appl.*, vol. 32, no. 10, pp. 6111–6124, May 2020.

[11] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Dec. 1998.

[12] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent. (ICLR)*. San Diego, CA, USA, 2015, pp. 521–534.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 770–778.

[14] J. Wang, Z. Mo, H. Zhang, and Q. Miao, "A deep learning method for bearing fault diagnosis based on time-frequency image," *IEEE Access*, vol. 7, pp. 42373–42383, 2019.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[16] H. Liu, J. Zhou, Y. Zheng, W. Jiang, and Y. Zhang, "Fault diagnosis of rolling bearings with recurrent neural network-based autoencoders," *ISA Trans.*, vol. 77, pp. 167–178, Jun. 2018.

[17] O. Sagi and L. Rokach, "Ensemble learning: A survey," *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, vol. 8, no. 4, p. e1249, 2018.

[18] Z. Wang, Q. Zhang, J. Xiong, M. Xiao, G. Sun, and J. He, "Fault diagnosis of a rolling bearing using wavelet packet denoising and random forests," *IEEE Sensors J.*, vol. 17, no. 17, pp. 5581–5588, Sep. 2017.

[19] G. Xu, M. Liu, Z. Jiang, D. Söffker, and W. Shen, "Bearing fault diagnosis method based on deep convolutional neural network and random forest ensemble learning," *Sensors*, vol. 19, no. 5, p. 1088, Mar. 2019.

[20] G. Tang, B. Pang, T. Tian, and C. Zhou, "Fault diagnosis of rolling bearings based on improved fast spectral correlation and optimized random forest," *Appl. Sci.*, vol. 8, no. 10, p. 1859, 2018.

[21] T. Han and D. Jiang, "Rolling bearing fault diagnostic method based on VMD-AR model and random forest classifier," *Shock Vib.*, vol. 2016, Sep. Jun. 2016, Art. no. 5132046.

[22] P. Kundu, A. K. Darpe, and M. S. Kulkarni, "An ensemble decision tree methodology for remaining useful life prediction of spur gears under natural pitting progression," *Structural Health Monitor.*, vol. 19, no. 3, pp. 854–872, May 2020.

[23] B. Pang, T. Tian, and G.-J. Tang, "Fault state recognition of wind turbine gearbox based on generalized multi-scale dynamic time warping," *Struct. Health Monitor.*, vol. 2020, Dec. 2020, Art. no. 147592172097862, doi: 10.1177/1475921720978622.

[24] Y. Xu, Y. Sun, J. Wan, X. Liu, and Z. Song, "Industrial big data for fault diagnosis: Taxonomy, review, and applications," *IEEE Access*, vol. 5, pp. 17368–17380, 2017.

[25] H. Yan, J. Wan, C. Zhang, S. Tang, Q. Hua, and Z. Wang, "Industrial big data analytics for prediction of remaining useful life based on deep learning," *IEEE Access*, vol. 6, pp. 17190–17197, 2018.

[26] A. Alkasem, H. Liu, and M. Shafiq, "Improving fault diagnosis performance using Hadoop MapReduce for efficient classification and analysis of large data sets," *J. Comput.*, vol. 29, no. 4, pp. 185–202, 2018.

[27] Y. Cao, P. Li, and Y. Zhang, "Parallel processing algorithm for railway signal fault diagnosis data based on cloud computing," *Future Gener. Comput. Syst.*, vol. 88, pp. 279–283, Nov. 2018.

[28] H. Miao, H. Zhang, M. Chen, B. Qi, and J. Li, "Two-level fault diagnosis of SF6 electrical equipment based on big data analysis," *Big Data Cognit. Comput.*, vol. 3, no. 1, p. 4, Jan. 2019.

[29] H. B. Deng, J. Hu, X. S. Wang, and M. Yu, "Transformer fault diagnosis based on massive vibration data," in *Proc. Chin. Control Decis. Conf. (CCDC)*, Nanchang, China, Jun. 2019, pp. 4586–4591.

[30] M. B. Imani, M. Heydarzadeh, L. Khan, and M. Nourani, "A scalable spark-based fault diagnosis platform for gearbox fault diagnosis in wind farms," in *Proc. IEEE Int. Conf. Inf. Reuse Integr. (IRI)*, San Diego, CA, USA, Aug. 2017, pp. 100–107.

[31] H. Lei, J. Liu, and C. Xian, "Application of distributed machine learning model in fault diagnosis of air preheater," in *Proc. 4th Int. Conf. Syst. Rel. Saf. (ICSRS)*, Rome, Italy, Nov. 2019, pp. 312–317.

[32] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[33] M. Zaharia, R. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. Franklinm, and A. Ghodsi, "Apache spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[34] J. Chen, K. Li, Z. Tang, K. Bilal, S. Yu, C. Weng, and K. Li, "A parallel random forest algorithm for big data in a spark cloud computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 919–933, Apr. 2017.

[35] T. T. Swe, P. Phyu, and S. P. P. Thein, "Weather prediction model using random forest algorithm and Apache Spark," *Int. J. Trend Sci. Res. Dev.*, vol. 3, no. 6, pp. 549–552, 2019.

[36] H. Chen, P. Chang, Z. Hu, H. Fu, and L. Yan, "A spark-based ant lion algorithm for parameters optimization of random forest in credit classification," in *Proc. IEEE Inf. Technol. Netw. Electron. Autom. Control Conf. (ITNEC)*. Chengdu, China, Dec. 2019, pp. 992–996.

[37] W. Lin, Z. Wu, L. Lin, A. Wen, and J. Li, "An ensemble random forest algorithm for insurance big data analysis," *IEEE Access*, vol. 5, pp. 16568–16575, 2017.

[38] B. Ait Hammou, A. Ait Lahcen, and S. Mouline, "An effective distributed predictive model with matrix factorization and random forest for big data recommendation systems," *Expert Syst. Appl.*, vol. 137, pp. 253–265, Dec. 2019.

[39] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[40] S. Bharathidason and C. Jothi Venkataeswaran, "Improving classification accuracy based on random forest model with uncorrelated high performing trees," *Int. J. Comput. Appl.*, vol. 101, no. 13, pp. 26–30, Sep. 2014.

[41] 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 *Proc. USENIX Symp. Netw. Syst. Design Implement.* San Jose, CA, USA, 2012, pp. 15–28.

[42] O. P. Yadav and G. Pahuja, "Bearing fault detection using logarithmic wavelet packet transform and support vector machine," *Int. J. Image Graph. Signal Process.*, vol. 11, no. 5, pp. 21–23, 2019.

[43] CWRU Bearing Data Center. *CWRU Rolling Bearing Dataset*. Accessed: Jan. 20, 2020. [Online]. Available: http://csegroups.case.edu/bearingdatacenter/home

[44] S. Ma, W. Cai, W. Liu, Z. Shang, and G. Liu, "A lighted deep convolutional neural network based fault diagnosis of rotating machinery," *Sensors*, vol. 19, no. 10, p. 2381, May 2019.

[45] X. Meng, J. Bradley, B. Yavuz, and E. Sparks, "MLLIB: Machine learning in Apache Spark," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1235–1241, 2016.

[46] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[47] L. Zhan, F. Ma, J. Zhang, C. Li, Z. Li, and T. Wang, "Fault feature extraction and diagnosis of rolling bearings based on enhanced complementary empirical mode decomposition with adaptive noise and statistical time-domain features," *Sensors*, vol. 19, no. 18, p. 4047, Sep. 2019.

[48] I. Reis, D. Baron, and S. Shahaf, "Probabilistic random forest: A machine learning algorithm for noisy data sets," *Astronomical J.*, vol. 157, no. 1, p. 16, Dec. 2018.

[49] S. Gao, X. Wang, X. Miao, C. Su, and Y. Li, "ASM1D-GAN: An intelligent fault diagnosis method based on assembled 1D convolutional neural network and generative adversarial networks," *J. Signal Process. Syst.*, vol. 91, no. 10, pp. 1237–1247, Oct. 2019.

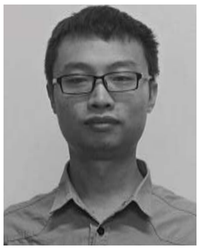
[50] Z. Qu, H. Li, Y. Wang, J. Zhang, A. Abu-Siada, and Y. Yao, "Detection of electricity theft behavior based on improved synthetic minority oversampling technique and random forest classifier," *Energies*, vol. 13, no. 8, p. 2039, 2020.

[51] Z. Xu, D. Shen, T. Nie, and Y. Kou, "A hybrid sampling algorithm combining M-SMOTE and ENN based on random forest for medical imbalanced data," *J. Biomed. Informat.*, vol. 107, Jul. 2020, Art. no. 103465.

[52] W. Mao, Y. Liu, L. Ding, and Y. Li, "Imbalanced fault diagnosis of rolling bearing based on generative adversarial network: A comparative study," *IEEE Access*, vol. 7, pp. 9515–9530, 2019.

[53] S. Shao, P. Wang, and R. Yan, "Generative adversarial networks for data augmentation in machine fault diagnosis," *Comput. Ind.*, vol. 106, pp. 85–93, Apr. 2019.

LANJUN WAN was born in Hunan, China, in 1982. He received the B.S. and M.S. degrees in computer science and technology from the Hunan University of Technology, Zhuzhou, China, in 2005 and 2009, respectively, and the Ph.D. degree in circuits and systems from Hunan University, Changsha, China, in 2016. He is currently an Assistant Professor with the School of Computer Science, Hunan University of Technology. His research interests include industrial big data analysis, industrial equipment fault diagnosis, high-performance computing, and parallel computing. He has published many research articles in international conferences and journals, such as *Journal of Parallel and Distributed Computing (JPDC)*, *Concurrency and Computation: Practice and Experience (CCPE)*, *Parallel Computing*, and *Sensors*. He serves as a reviewer for the *Journal of Parallel and Distributed Computing (JPDC)*, *Concurrency and Computation: Practice and Experience (CCPE)*, and IEEE Access.



KUN GONG was born in Hunan, China, in 1996. He received the B.S. degree in mechanical engineering from the Hunan University of Technology, Zhuzhou, China, in 2019, where he is currently pursuing the M.S. degree in computer science and technology. His research interests include industrial big data analysis and industrial equipment fault diagnosis.

GEN ZHANG was born in Anhui, China, in 1995. He received the B.S. degree in network engineering from West Anhui University, Luan, China, in 2019. He is currently pursuing the M.S. degree in computer science and technology with the Hunan University of Technology, Zhuzhou, China. His research interests include industrial big data analysis and industrial equipment fault diagnosis.

XINPAN YUAN was born in Hunan, China, in 1982. He received the B.S., M.S., and Ph.D. degrees in computer science and technology from Central South University, Changsha, China, in 2005, 2008, and 2012, respectively. He is currently an Associate Professor with the School of Computer Science, Hunan University of Technology, Zhuzhou, China. His research interests include industrial big data analysis, industrial equipment fault diagnosis, information retrieval, data mining, and natural language processing. He has published many research articles in international conferences and journals, such as *International Journal of Nursing Studies (IJNS)*, *Journal of Information Processing Systems (JIPS)*, and *Information*.

CHANGYUN LI was born in Hunan, China, in 1972. He received the Ph.D. degree in computer science and technology from Zhejiang University, Hangzhou, China, in 2007. He is currently a Full Professor of Computer Science and the Dean of the Graduate School, Hunan University of Technology, Zhuzhou, China. His research interests include industrial big data analysis, industrial equipment fault diagnosis, intelligent information perception and processing technology, the Internet of Things, and software methodology. He has published many research articles in international conferences and journals, such as *Journal of Information and Communication Technology (JICT)*, *Journal of Software (JSW)*, and *The Journal of Chemical Physics (JCP)*.

XIAOJUN DENG was born in Hunan, China, in 1974. He received the M.S. degree in computer science and technology from the National University of Defense Technology, Changsha, China, in 2004. He is currently a Full Professor with the School of Computer Science, Hunan University of Technology, Zhuzhou, China. His research interests include industrial big data analysis, industry equipment health management, the Internet of Things, and image processing. He has published many research articles in international conferences and journals, such as *International Journal of Studies in Nursing (IJSN)*, *Journal of Computer Science and Engineering (JCSE)*, and IEEE Access.