IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 11 July 2022, accepted 29 July 2022, date of publication 1 August 2022, date of current version 8 August 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3195872

RESEARCH ARTICLE

KNN Normalized Optimization and Platform Tuning Based on Hadoop

CHEN MA AND YUHONG CHI
School of Computer Science, Xijing University, Xi'an 710123, China
Corresponding author: Chen Ma (997528107@qq.com)

ABSTRACT Big data has become part of the life for many people. The data about people's life are being continously collected, analysized and applied as our society progresses into the big data era. Behind the scene, the computer server clusters need to process hundres of millions pieces of data every day. It is very important to choose the right big data processing platform and algorithm to deal with different kinds of datasets. Therefore, in order to be fully familiar with the related work of driving big data processing, it is necessary to master the classification algorithm of data. It aims to help us carry out a classification model or operation analysis of classification function by screening and classifying the current data in data mining. In addition, the given data can be mapped to the specified category area, and the development trend of future data can be predicted through classification models. So this kind of algorithm helps to reduce the difficulty of work operation and improve people's work efficiency. This paper optimizes the classical classification algorithm—KNN, and designs a new normalized algorithm called PEWM_G KNN. From the perspective of distance measurement, we use Pearson correlation coefficient to replace the traditional Euclidean Metric, then we further refine the study for attribute values of datasets and introduce the entropy weight method, combined with Pearson's measure to optimize the distance calculation equation. After the K value is fixed, we added Gaussian Function to carry out the selection of classification. In this study, we compared the effects of every step, and tested datasets with different data types and sizes, in order to test the performance of the algorithm under different scenarios. The datasets we used include Iris, Breast Cancer, Dry Bean and HTRU2 (All datasets are from The University of California, Irvine). Finally, we further analyze the performance of different system configuration parameters on the prediction rate and time. The experimental results show that PEWM_G KNN algorithm has better optimization effect for datasets with more complex attribute values and more records than the original KNN algorithm. Moreover, the optimization of platform parameters improves prediction rates of algorithms and reduces the time. We tested PEWM_G KNN on the Hadoop platform, confiured with HDFS, Hadoop-YARN, ZooKeeper, Hadoop-HA and MapReduce.

INDEX TERMS KNN, classification algorithm, HDFS, Hadoop-YARN, ZooKeeper, Hadoop-HA, Hadoop, MapReduce.

I. INTRODUCTION

With social progress and technological development, big data has gradually become an indispensable part of people's daily life. So, choosing the right big data processing platform and big data processing algorithms for different datasets is also one of the challenges today. Among many big data processing algorithms, classification algorithm has always been a hot topic and one of data mining application technologies, and tasks it handles generally fall into two broad catrgories:

(1) predictable tasks, which predicts the value of a particular attribute based on the values of other attributes.

(2) describable tasks and outline patterns (correlations, trends, clusters, trajectories, and anomalies) of potential connections in data.

Classification is a supervised learning process. It learns the attributes of an existing dataset. The categories in the target dataset are known, and what the classification process

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

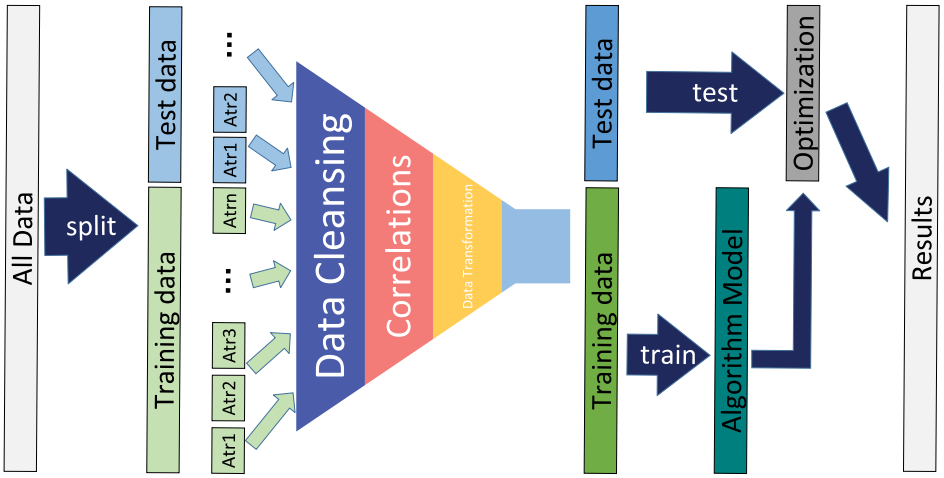


FIGURE 1. Classification algorithm workflow.

needs to do is to classify each record into the corresponding categories. The classification algorithm work flow is as follows:

It can be seen from Fig 1 that the data in datasets is divided before classification, and programmers can divide it into Test data and Training data in a customized ratio. Each record in both has the same set of attributes $\{Atr_1, Atr_2, \ldots, Atr_{n-1}, Atr_n\}$. Firstly, data cleaning operations (Data Cleansing) are required to detect and correct (or delete) the recordset. The inaccurate or corrupted records in a dataset are purged if all or part of their data are identified as incorrect, incomplete, irrelevant, inaccurate or otherwise problematic (dirty) before the data can be formally classified. Secondly, the correlation analysis of datasets(Correlation) is carried out. When there are too many attributes in a dataset, several attributes that can not only determine the characteristics of the dataset but also be representative can be selected by analyzing the relationship between different attributes or data, in order to simplify the training steps and obtain most representative training results. Data conversion(Data Transformation) is then performed, that is, data types can be converted, cleaned up, enriched, or aggregation can be performed by removing null or duplicate data. It can also make values that are strings or more complex data types into a simple type to facilitate calculation of the rest of data characteristics. After the above three steps of data filtering, we can get relatively clean Training data and Test data. Then Training data is first trained by the Algorithm Model, which is used to fit the model and train it by setting its parameters. After training, an optimal fitting model (Optimization) will be generated. The Test data can then be tested against it to get results, including accuracy, recall and other predicted information.

In general, there are a wide varieties of classification algorithms that can be applied to real world use cases. For example, e.g. [1] proposed two new incremental support vector machine approaches to improve the classification. They added several sensor networks to the smart home, to monitor residents and the environment, interpret the current situation and respond immediately. The paper also proposed and evaluated an extended sensor window approach to perform activity recognition in a streaming manner, that is, identify activity when a new sensor event is recorded. The experiment shows that the introduced incremental learning based on similarity has been shown to be 5 to 9 times faster than other methods in terms of training performance. Similarly, the final state sensor feature approach introduced improved f1 scores by at least 5% when a baseline SVM classifier was used. The e.g. [2] proposed four different PD detection decision tree integration methods based on the ForEx++ rule framework. In this paper, SysFor and ForestPA decision forest algorithms are used to control the number of trees in forest during training, which has significant detection accuracy. The Forex++ framework is used to improve the establishment process of decision forest via improving detection capability. The experiment validated these methods separately through tenfold cross-validation, LOSO cross-validation, and an ideal training–testing split of instances. Compared with other feature sorting and selection schemes, four methods are proved to be a robust one. The e.g. [3] examines the use of artificial neural networks using GPS-based large datasets and additional generated data to predict the choice of mode of transport for a new metro line in the Amsterdam metropolitan area along the north-south axis of the city. The study found that if the data on which the model was trained and tested were collected over a very similar underlying transport network, the model performed better on improving the prediction rate. And the results also seem to imply that behavioral patterns have changed significantly with the opening of new subway lines and the reorganization of the network, so it would certainly be interesting to study these in more detail. In order to deal with the uncertainty of human behavior, an improved fuzzy finite state machine (FFSM) model is proposed in e.g. [4] by

combining it with LSTM neural network and convolutional neural network (CNN). In this study, LSTM is added to enhance the learning ability of FFSM model and accurately generate fuzzy rules that control the transition between system states. CNN is added to better simulate daily human activities and collect digital and temporal information from sensory data. The experiment also proves the effectiveness of the proposed method, especially when it is applied to large data sets, the performance will be more steady and reliable. The e.g. [5] proposed a method combining fuzziness and semantic knowledge (called semFBnet) to optimize the poor performance of discrete Bayes due to the strict boundary values of discretization data, the uncertainty of parameter values and the lack of domain semantic knowledge. The method reduces uncertainty in the parametric school process and evaluates multiple daily time series predictions of temperature, wind speed, relative humidity and precipitation rates for different climatic regions in India. The study shows that compared with other methods, the semFBnet analysis results are encouraging, and the prediction performance is improved with less uncertainty.

The algorithm discussed and optimized in this paper is K-Nearst Neighbors (KNN for short). The basic idea is: the labels of categories are known for the training data. when the test data is processed, the algorithm compares the test data's features with those of the training data, and find the top K data points that are the most similar to the test data. Then the category of the test data is the category that appears most frequently among the top K data. Because KNN mainly relies on the surrounding limited adjacent samples, not the method of discriminating the class domain to determine the category, this algorithm is more suitable than others for samples to be divided with more overlapping class domains. KNN can be used not only for classification, but also for regression. The attributes of a sample can be obtained by finding the k nearest neighbors of it and assigning the average value of attributes of these neighbors to it. A more useful way is to assign different weights to neighbors at different distances on the sample, like the weight is inversely proportional to the distance. In this paper, based on the traditional KNN algorithm, the following three operations are done:

(1) In terms of distance measurement, Pearson correlation coefficient is used to replace Euclidean Metric to check and calculate the similarity between each attribute of test and training data from the perspective of correlation.

(2) In terms of refining attribute values, entropy weight method is adopted to determine attribute weights according to the variation degree of each attribute of data, which makes the result of distance measurement more objective.

(3) In terms of determining the type of samples, Gaussian optimization is used to optimize weights of samples with different distances to prevent the error of prediction types due to the imbalance of numbers of different samples, and the purpose is to improve prediction rates.

In this paper, the new algorithm is called PEWM_G KNN.

The main contributions are summarized as follows:

1. This paper presents a new algorithm——PEWM_G KNN, based on Pearson measure, entropy weight method and Gaussian function optimization.

2. It can be seen from the above optimization steps (1) and (2) that Pearson measure is used to replace the traditional Euclidean distance to solve the low prediction rate caused by the dimensional problem. In addition, entropy weight method is added to bring the weight of features into distance measurement, so as to refine the calculation with more comprehensive research.

3. According to the optimization step(3), this algorithm uses Gaussian function to solve the problem of low prediction accuracy caused by different numbers of instances in datasets.

4. The model is superior to other classifiers, such as KNN, Decision Tree, Adaboost and Random Forest. And the more large number of instances is, the more significant prediction effect of the model is.

5. The optimization algorithm is really applied to a big data platform, and related parameters are adjusted and optimized by combining platform parameters and operation process. Experimental results show that the parameter optimization has little influence on the prediction rate, but can effectively reduce the platform running time.

The remaining sections of this paper are as follows. In Sect. II, the relevant work of Hadoop and MapReduce are presented. In Sect. III, the relevant background knowledge and model of Hadoop are presented. In Sect. IV, the algorithm knowledge and model of PEWM_G are illustrated. In Sect. V, the comparison results of prediction rates of PEWM_G for different datasets are given. In Sect. VI, the process of tuning Hadoop configuration parameters is given, and after that, the prediction rate and processing time of PEWM_G for different datasets are compared. And the Sect. VII is the summary.

II. RELATED WORK OF PLATFORM AND COMPUTING FRAMEWORK

In addition to the optimization of KNN, this paper also optimized parameters of big data management platform —— Hadoop, in order to better simulate the real big data processing environment. Hadoop is a distributed system infrastructure developed by the Apache Foundation, which mainly solves massive data storage, analysis and calculation problems. It has the high reliability of bit-based storage and data processing capabilities, When data is allocated through available clusters to complete storage and computing tasks, these clusters can be easily extended to thousands of nodes with high scalability. Moreover, it can dynamically move data between nodes and ensure the dynamic balance of each node. As such, the processing speed is very fast and highly efficient. It can also automatically save multiple copies of data and reassign failed tasks with high fault tolerance. Therefore, this paper chooses this platform for data training and testing. Of course, the optimization of Hadoop platform parameters involves lots of conceptual model principles, and the analysis of them has also been mentioned in many studies. For example: e.g. [6] proposes a method named KNN-DP to

alleviate the load imbalance caused by data skewness. The primary goal is to evenly divide data to be processed in the MapReduce framework into a large number of partitions, which are then used to process small samples representing large datasets supported by a sampling technique. It also uses local nearest neighbors to obtain global nearest neighbors, and adds a small amount of local redundant data to each node, so as to improve the accuracy of approximate solution and the speed of data processing. In this paper, when optimizing platform operation parameters, the underlying principles involved will be analyzed on the basis of conceptual model principles. For example, when refining MapReduce, the ring buffer at the Map end is systematically described, and the relationship between the modulation and its operation logic is analyzed to ensure optimization effect is consistent with the parameter logic.

Besides studies on Hadoop parameters mentioned above, there are also many literatures on the derivation of KNN and big data computing framework MapReduce. In e.g. [7], a new optimized hybrid research method is proposed, which combines optimized genetic algorithm with principal and independent component analysis to identify optimal subsets and potential related features, respectively. Finally, KNN algorithm is used to classify the anopheles gambiae dataset, which contains 7 features and 2457 samples. Although the actual instance size and features are not very large, there are 708 significant features selected by the the optimized genetic algorithm. After that, the Principal Component Analysis (PCA) was used to select 10 potential features in a short period of time, and KNN classification algorithm was used for classification. However, although the dataset can specifically solve the question of a certain type, the effect that a single dataset may achieve is not significant. If there are more or fewer features and samples, will it have an impact on the prediction rate and precision of results? In addition, configuration parameters of the experimental environment may also affect experimental results, which are all factors to be considered in the experiment. Therefore, this paper selects datasets with different attributes, numbers of samples and adjusts configuration parameters of the experimental environment to compare effects of optimization algorithms, and strive to objectively analyze algorithm optimization and processing conditions from different perspectives.

The e.g. [8] proposes a three-partition active learning (TALK) through k-nearest neighbors, whose objective is to minimize total training cost and misclassification cost. The experiment focuses on reducing the total cost and aims to create a new cost sensitive classification and active learning algorithm, and it is representative in the selection of datasets. Eight different types of data sets are selected, with different eigenvalues, classification types and number of instances, and they are used to compare classification costs with other active learning algorithms so that the experimental results are representative. This paper analyzes whether the complexity of dataset has a certain degree of influence on PEWM G, and also studies turning parameters of Hadoop to check its applicability for various types of datasets. The four datasets in this paper are all from the University of California, Irvine [9], which are consistent with the datasets in e.g. [8], including iris [10], breast cancer diagnosis in Wisconsin [11], dry bean [12] and pulsar [13]. The properties of pulsars can be explained in more detail in e.g. [14], and dry bean dataset is also often used to discuss classification problems, as can be seen from e.g. [15].

The e.g. [16] integrates the MapReduce parallel programming framework [17] with heuristics to overcome scalability challenges and provide much-needed privacy protection, while producing efficient analysis results in limited execution time. Features such as simplicity and high fault tolerance are key to making MapReduce a promising framework, and this is one reason why this paper chose to use it on Hadoop. The study carries out multiple sets of experiments on the number of compute nodes, size of data sets, and data hiding methods, then it is concluded that the combination of MapReduce with these adopted heuristic algorithms can achieve scalable and multifold speed, thus producing efficient analysis results. It is worth mentioning the continuous refinement of configuration parameters in e.g. [16], from various aspects to analyze the impact of platform architecture on the efficiency of algorithms, experimental speed and so on. So, this paper will also analyze the influence of various platform configuration parameters on the prediction rate and time of PEWM G based on underlying principles and experimental results of MapReduce.

Based on MapReduce programming, e.g. [18] proposes a meta-learning applied on MapReduce [19] to avoid parallelization of machine learning algorithms and improve their scalability to large datasets. Authors's main point was that the lack of iteration support in MapReduce may result in the high overhead of creating circular methods in Hadoop ([20], [21]), and it is easier to realize the iterative algorithm in models using meta-learning mode than directly using MapReduce to modify its own internal algorithm. For example, in literature [22]–[25]. It can be seen that MapReduce is still a popular parallel programming framework so far, and the discussion and research on it are still being enriched.

III. PLATFORM RELATED TECHNICAL BACKGROUND

In this section, we will introduce the related technical background of Hadoop, the big data management platform used by the experiment in this paper. And this includes Hadoop Distributed File System(HDFS), Yet Another Resource Negotiator(YARN), ZooKeeper, hadoop high availability(HA) and the computing framework——MapReduce. Some details of them related to the experiment will be mentioned in subsequent experiments.

A. HDFS

The full name of HDFS is Hadoop Distributed File System, which is a distributed file system designed to run on general-purpose hardwares. In modern enterprise environments,

single machine capacity is often insufficient to store large amounts of data, and cross-machine storage is required. And once the network is introduced into the system, it inevitably adds programming complexity, such as ensuring that data is not lost when nodes are not available. Traditional network file systems (NFS) is a type of distributed file system with limitations. Since files are stored on a single server, when many clients access the server at the same time, the server may be overloaded. In addition, operations on NFS files need to be synchronized to the local PC, and the modification is invisible to clients until the modification is synchronized to servers. NFS is also a file system that shares data through the Remote Procedure Call(RPC) communication protocol, so it must run while ensuring that works. When files are read and written, servers actually operates on a shared file address. Once they fail, other machines cannot read, written files and data cannot be retrieved. But HDFS can be integrated with local systems, Amazon S3([26]–[29], [30], [31]) and other systems, and can even be operated through Web protocols. Its files are distributed on clustered machines and copies are provided for fault tolerance and reliability. For example, the client's direct operations of writing and reading files are distributed across machines in the cluster, with no performance pressure of single point. Compared with NFS, HDFS has system backup and managed nodes can perform backup processing, which is more secure and reliable. After data is stored in multiple copies, it can resist various disasters. As long as one copy is not lost, data will not be lost. Therefore, the data security is high.

The storage architecture of HDFS is as follows:

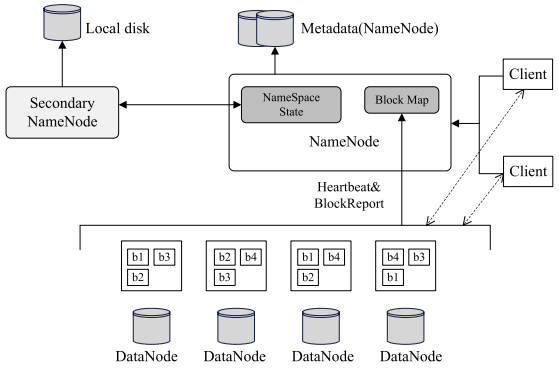


FIGURE 2. The storage architecture of HDFS.

It uses a master-slave architecture to store data, which mainly consists of four parts in Fig 2: HDFS Client, NameNode, DataNode and Secondary NameNode. Let's take a look at what each part does and how it works.

(1) HDFS Client is the client and user of HDFS. It can manage HDFS through commands, such as starting or stopping it, and access it through commands. When accessing the HDFS, it first needs to upload files, which are divided into small blocks $\{b_1, b_2, b_3, b_4\}$. These blocks are then stored in DataNodes, with which clients can interact to read or write data. However, before doing that, the HDFS Client needs to interact with the NameNode to obtain the metadata information stored in it.

(2) NameNode is the master in master-slave scheduling structure, a supervisor manager. It keeps the directory tree of all files in the file system, and tracks where across the cluster the file data is kept. It does not store the data of these files itself. One of its functions is to save and manage HDFS metadata, which is the system data used to describe the characteristics of a file, such as access rights, file owners, and the distribution of data blocks. In a clustered file system, distribution information includes the location of the file on disk or cluster. Clients who need to operate a file must firstly get its metadata before they can locate the location and get the content or related attributes of the file. In HDFS, the files that manage metadata are edits(operation log files) and fsimage(namespace image files). When NameNode is started, operations performed by HDFS are recorded in edits, and fsimage contains metadata information in HDFS (such as modified time, access time, data block information). Therefore, NameNode can manages blocks on DataNodes.

(3) DataNode is the slave in the master-slave scheduling structure, when NameNode gives the commands to it, the DataNode will performs the actual operations. HDFS firstly divides a large file into several small blocks, and then writes them to different nodes(called DataNodes), which are responsible for storing file data. First, DataNode's most obvious function is to store data, store blocks as Linux files on disk, and read and write them according to block id and byte range. One block is backed up on multiple Datanodes. In addition, DataNode will communicates with NameNode periodically and receive instructions from it. If NameNode does not receive the heartbeat message from one DataNode within a specified period of time, the DataNode is marked dead and the NameNode won't forward further data operation to it.

(4) HDFS has a node that periodically creates namespace checkpoint operations, which is Secondary NameNode. NameNode first records write operations such as creating, moving and deleting files submitted by HDFS Clients in edit logs(edits). The fsimage is a snapshot of metadata in NameNode, which contains all metadata information of DataNodes and their blocks. After NameNode starts, changes to the file system are saved in edits. On the next reboot, edits will be merged into fsimage to get the latest snapshot. Secondary NameNode's duty is to merge edit logs(NameNode) into fsimage(NameNode). First, it periodically fetches edit logs(edits) from NameNode and updates them to fsimage(Secondary NameNode). It loads the downloaded fsimage into local disk(Fig 2), and then executes update operations in edits, keeping fsimage in memory up to date. After the update succeeds, it will copy fsimage back to NameNode, which will use this new fsimage the next time it restarts, reducing the restart time.

B. YARN

Apache Hadoop YARN(Yet Another Resource Negotiator) is a new Hadoop explorer responsible for providing server resources for computing applications, which is equivalent to a distributed operating system platform. It can provide unified resource management and scheduling for upper-layer applications, and its introduction brings great benefits to clusters in unified resource utilization management and data sharing. YARN does not know the operating mechanism of programs submitted by clients, but only provides the scheduling of computing resources. That is, programs apply for resources to YARN, and YARN allocates resources. It is completely decoupled from running user programs, which means that various types of the distributed computing architecture can be run on it, like MapReduce [17], Storm [32], DataMPI [33] and any framework of the resource request mechanism that complies with YARN specifications. so it is a universal resource scheduling platform. All existing computing clusters in an enterprise can be integrated into a physical cluster to improve resource utilization and facilitate data sharing. Its architecture and workflow are as follows:

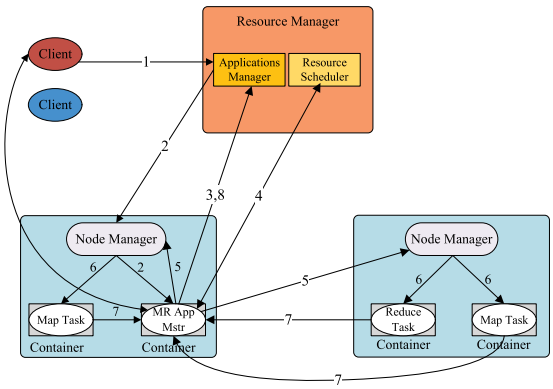


FIGURE 3. The architecture and workflow of YARN.

First, let's take a look at four major components of the YARN architecture: ResourceManager(RM), ApplicationMaster(AM), NodeManager(NM) and Container.

(1) As a core component of YARN, Resource Manager is the main control node of a YARN cluster that schedules resources based on application requirements. It coordinates and manages resources of the entire cluster (all NodeManagers) and responds to resolution, scheduling, and monitoring of applications submitted by clients. It is mainly composed of Resource Scheduler and Application Manager. Resource Scheduler allocates resources to running applications according to certain policies based on system resource capacity and queue constraints imposed by administrators. The Resource Scheduler is not responsible for monitoring or tracking application execution status, or for restarting failed tasks caused by execution or hardware failure. These tasks are the responsibility of the ApplicationMasters. Applications Manager is responsible for managing all applications throughout the system, including application submissions requesting resources from the task scheduler to start ApplicationMaster, monitoring the running state, and restart it if it fails.

(2) Each time a client submits an application, a corresponding ApplicationMaster is generated for that, and this single process is running on Node Manager. Its effect is to apply for resources (represented by Containers) from ResourceManager for applications, communicates with NodeManager to start or stop tasks, and monitors the running status of all tasks. If one task fails, it will apply for resources again and restart this task. And after registering with ResourceManager, ApplicationMaster will provides clients with job progress information.

(3) NodeManager is the real resource provider in YARN and one provider of the Container that actually executes the application and monitors the resource usage of applications. It is the resource and task manager on each child node, the agent that manages the current node. Each node in the YARN cluster runs a NodeManager. On the one hand, NodeManager reports resource usage on the current node and the running status of each Container to RM through heartbeat information. On the other hand, it receives and processes various requests from AM to start or stop Containers, monitors the lifecycle management and resource usage of Containers, tracks node health, and manages logs and ancillary services used by different applications.

(4) Container is an abstraction of system resources in YARN and a basic unit of resource allocation. It encapsulates multi-dimensional resources on nodes, including CPU, memory, disk, and networks. When an AM applies for resources from RM, resources returned by RM are represented by Containers. It assigns a Container to each task, which can only use the resources described in the Container. It is also a dynamic resource division unit that is automatically generated based on the resources required by submitted applications.

The mentioned above are the four major components involved in YARN. On this basis, let's learn about the working process and procedure of YARN in Fig 3:

1. The client submits an application program to Yarn, including files related to it, ApplicationMaster(AM) commands, ApplicationMaster(AM) programs, and etc.

2. ResourceManager(RM) assigns a Container to the application, and communicates with NodeManager(NM) where the Container is located. And this NM is required to start the application's corresponding ApplicationMaster(denoted by MR APP Mstr) in the Container.

3. MR APP Mstr first registers with RM so that the client can see the running status of the application directly from RM, then requests resources for various tasks of the current application and monitors their running status until the end of the run(performed by Applications Manager).

4. AM applies for and obtains resources from RM through RPC(Remote Procedure Call Protocol) protocol in polling mode.

5. Once the AM requests resources, it communicates with NM corresponding to the requested Container and asks it to

start tasks. It is important to note that both NM and Container are not unique, that is, there are multiple NM nodes and Containers within the cluster to share the computation of current application. In addition, there may be different types of computing tasks, such as Map Task and Reduce Task in Fig. 3.

6. NM configures the environment for tasks to be started and writes the startup commands in a script that runs them.

7. Each task reports its running status and progress to the corresponding MR APP Mstr through RPC protocol, so that MR APP Mstr can master the running status and restart tasks when they fail.

8. After application mentioned by the client is running, MR APP Mstr logs out to RM and shuts itself down.

YARN allows multiple distributed systems to be well integrated, greatly reducing the operation and maintenance cost of the whole cluster. YARN monitors the status of each distributed task. For the mutable part —— Application-Master, clients can write their own App Mstr for different programming models to allow more types of programming models run in the Hadoop cluster. YARN can use cluster resources greatly to avoid wasting them, and also realizes the data sharing of clusters so that different distributed computing framework can complete data interaction.

C. ZooKeeper

ZooKeeper is an open-source distributed application service, an implementation of Google's Chubby and a key component of Hadoop. It provides consistency services for distributed applications, so that they can implement functions such as data publishing or subscribing, load balancing, naming services, and etc based on ZooKeeper. It is designed to encapsulate complex and error-prone services and provide clients with easy-to-use interfaces, efficient and functional systems.

A distributed system is a system which is composed of multiple softwares running independently across multiple physical hosts, and the performance of each server is limited. But in a distributed system, many servers can be consolidated into a cluster, whose capacity can be expanded indefinitely. The advantages of this type of system are fast data processing, high data security and mutual backup, but then there's a problem: how is information synchronized and shared between nodes on a distributed system? The method ZooKeeper uses is shared storage, and the process is as follows:

In fact, the principle of distributed collaboration in ZooKeeper is the same as project team synchronizing tasks in Subversion(SVN). ZooKeeper is like SVN, which stores shared information about the completion of tasks and so on. Each distributed application node is a member subscribs to this shared information. When the master node (leader) changes the labor division information of a slave node, this slave that subscribles to the ZooKeeper will be notified and get its latest task assignment. When finishing tasks, the slave will store the completion to master node that subscribed to

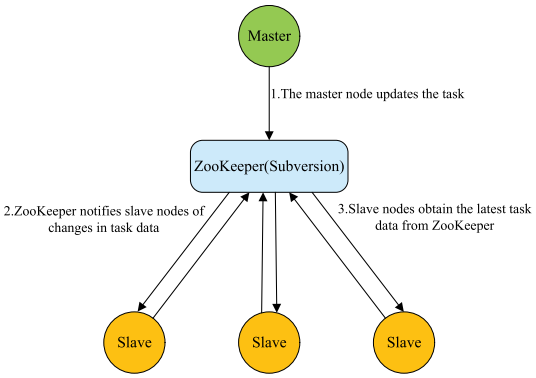


FIGURE 4. The process of shared storage.

this task's completion information, so it will be notified that ZooKeeper is complete. Problems in most distributed systems are caused by information sharing. If information between nodes can't be shared and synchronized in a timely manner, various problems will occur in the collaboration process. ZooKeeper solves the problem of collaboration by ensuring the consistency of information in distributed systems.

The ZooKeeper application mentioned in this paper is only used in Hadoop high availability(HA). Other areas of ZooKeeper such as main flow selection, synchronization flow, workflow-leader, workflow-follower, etc. are not described here.

D. HA

The most critical strategy of HA, or high availability, is to eliminate single points of failure. The single point of failure (SPOF) means that a failure in a system makes the entire system unable to function. In other words, a single point of failure causes an overall failure. In cluster environment, pressure is common, especially when there is only one NameNode. Once a failure occurs, it takes a long time to restart the system immediately, and the system cannot work during this period. Also, a single Namenode has limited memory, making DateNodes unscalable. The above problems are solved by configuring Active and Standby NameNodes to implement hot backup in HA. This is illustrated in the following HA architecture diagram:

Inside the blue box at the top, there is a new component called FailoverController. Its full name is ZKFailover-Controller(ZKFC), and it is a single ZooKeeper client. In Hadoop HA, it monitors and manages states of NameNode, with which it has a one-to-one relationship. Fig. 5 is a workflow diagram of Hadoop high availability, and NN represents NameNode, DN represents DataNode, ZK represents ZooKeeper. In this diagram, there are two NameNodes, one is Active and other is Standby. And they are not controlled directly by ZooKeeper, but by their respective ZKFailoverController(ZKFC)-specific processes. On the one hand, ZKFC keeps the heartbeat with ZooKeeper and sends its connection information, status information and

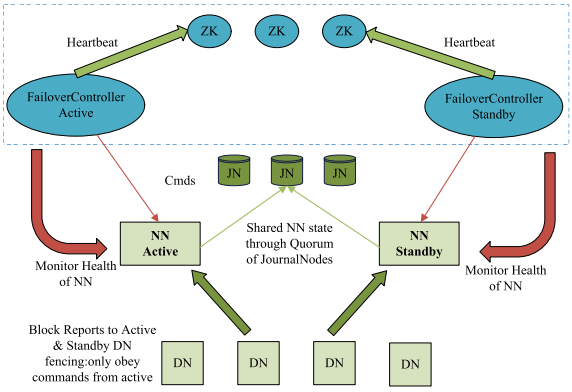


FIGURE 5. The HA architecture diagram.

task information back to ZooKeeper. On the other hand, the health information of NameNode is also monitored. When a NameNode goes down, it reports to its ZooKeeper, and the ZooKeeper of the other NameNode also gets information about its outage, because the information within the ZooKeeper cluster is synchronized. Once it gets this information, it sends an instruction to the NameNode it controls to switch from Standby to Active. Real-time information synchronization of NameNodes is achieved through JournalNodes(JN). The NameNode initially in Active periodically writes edits to JournalNode, and another Standby NameNode periodically takes edits out of JournalNode to itself.The above operation ensures not only the high availability of the NameNode, but also that the DataNode can communicate with the NameNode of Active, Standby. Once the Active one goes down, DataNode also automatically communicates with the new Active.

As you can see from the above description and Fig. 5, HA in this figure is not so much for Hadoop clearly, but for NameNode. Its most important function is active/standby switchover. So far, the background of experimental platform of this paper has been introduced, and the final cluster architecture built in this paper is Hadoop HA. A more detailed introduction of the hardware configuration involved in this paper will be introduced in the subsequent experiment section.

E. MapReduce

MapReduce is a programming model for parallel operations on large-scale datasets. Concepts "Map (mapping)" and "Reduce (contracting)" are their main ideas. The current software implementation is to specify a Map function to map a set of key-value pairs into a new set of key-value pairs, and to specify a concurrent Reduce function to ensure that each of all mapped pairs shares the same key group. Mapper is responsible for Map, and Reducer is responsible for Reduce. MapReduce's biggest characteristic is divide and conquer:

(1) Mapper is responsible for "divide", which divides complex tasks into several simple tasks: 1) the tasks whose sizes of data or calculations are greatly reduced compared to the original task, 2) the tasks are assigned to the node where the required data is stored for the calculation, 3) the tasks can be computed in parallel with little dependencies on others.

(2) Reducer is responsible for summarizing the results from the map stage. The user can set the parameter values in the relevant configuration file to determine the number of Reducers needed for a specific problem.

This is the concept of MapReduce, which works as follows:

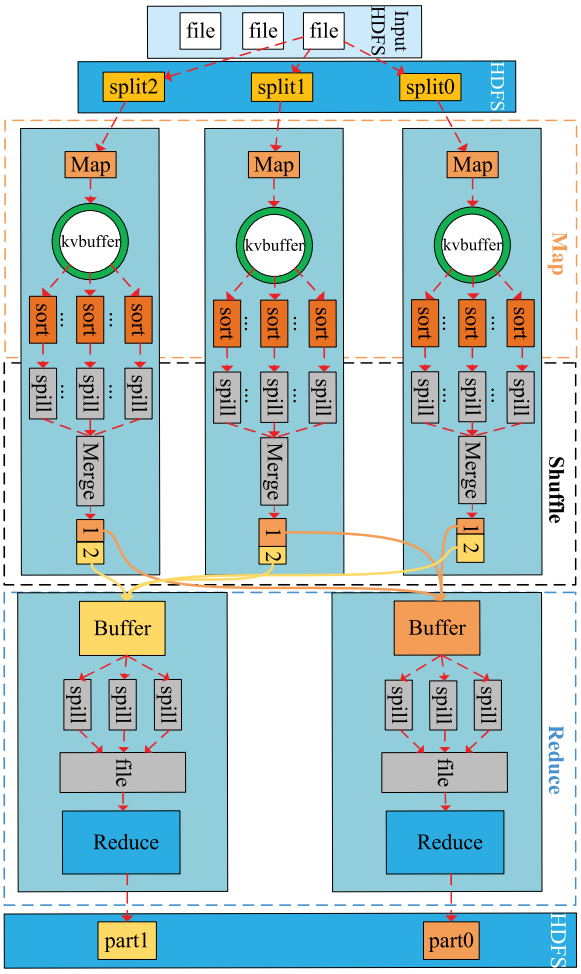


FIGURE 6. Workflow of MapReduce.

1. First, clients need to upload some source files to HDFS for processing and classification. These files will be split into split blocks after being uploaded. The number of split blocks and the size of each block can be set in Hadoop's configuration file.

2. On the Map end, each split block has a corresponding Map responsible for the partitioning of the data. A buffer pool, kvbuffer, then reads the input split blocks as key-value pairs. When the current read data reaches a certain threshold, kvbuffer outputs it and sort each key-value pair, exporting them in turn as $< k, v >$. The buffer pool size and threshold can be set in Hadoop configuration file.

3. Then on the shuffle end, key-value pairs output by the Map end is spilled to the disk first. Of course, if tasks

uploaded by client are computational and involve more computation, we can reduce the number of spilling using the corresponding settings in the Hadoop configuration file, thus reducing disk I/O. MapReduce then merges the spill blocks, combining values of the same key. However, it is important to note that the calculation of values of the same key is not performed at this stage. For example, for one key aaa, one map contains value 5 and another map contains value 6. The data stored on disk after the merge is [“*aaa*”, [5, 6, . . .]].

4. Finally, on the Reduce end, merged files on the disk from the shuffle phase is first exported to Buffers. The purpose is also to prevent reading merged files with large size and thus causing excessive disk I/O times. Of course, the Buffer works in the same way as the kvbuffer on the Map side. When a certain threshold is reached, the data in the Buffer is spilled and merged into a file, and each Reduce corresponds to an output file to be processed. Then Reduce computes these files, that is, merges values of the same key and outputs them as $< k, v >$. Eventually, the Reduce will merge all key-value pairs in the file into parts and upload them to HDFS to complete the entire MapReduce process. And the number of Reduce and the size of Buffer can also be specified in the Hadoop configuration file. At this point, the MapReduce workflow ends. Of course, there are some unmentioned workflow details and principles that are related to the platform parameters outlined below. They will be covered and analyzed in subsequent experiments.

Apache Hadoop MapReduce is a distributed computational framework developed by Doug Cutting based on Google’s MapReduce paper in e.g. [34]. It has many characteristics: 1) Easy to program. It simply implements some interfaces, and you can complete a distributed program, which can be distributed to a large number of cheap PC machines. 2) Good scalability. When your computing resources are not satisfied, you can expand its computing power by simply adding more machines. 3) High fault tolerance. When a machine in the cluster is down, it can transfer the above computational task to another node without making the task fail. And this process does not require human intervention and is completely done internally by Hadoop. 4) Suitable for offline processing. The MapReduce is difficult to do if the processed task is asked to return a millisecond-level result. 5) Wide application range. Many technical frameworks in the Hadoop ecosystem, such as Hive, Flume, Sqoop, Azkaban, and other underlying computing engines, are all using Apache Hadoop MapReduce.

IV. ALGORITHM MODEL

Because the algorithm PEWM_G KNN in this paper is optimized on the basis of KNN, algorithm models of its optimization will be introduced one by one in this section. Including k-nearest neighbor(KNN), Pearson correlation coefficient, entropy weight method(EWM) and Gaussian function.

A. KNN

K-nearest neighbor (KNN) algorithm is a famous pattern recognition statistical method, used broadly in machine learning classification. It is a relatively mature method in theory. It is both one of the simplest machine learning algorithms, the most basic of instance-based learning methods, and one of the best text classification algorithms. The k means k nearest neighbors. If most of the k most similar samples in a feature space (the closest neighbor in it) belong to a certain category, the sample also belongs to this category. KNN’s concept diagram is as follows:

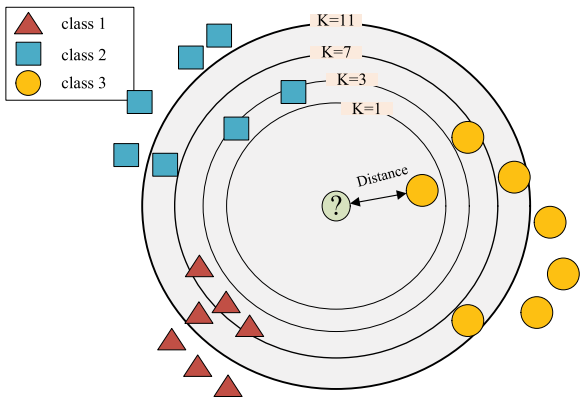


FIGURE 7. The concept diagram of KNN.

As shown in Fig. 7, for a point of unknown categories, the value of k directly affects the determination of its category. For example, when k = 1, there’s only 1 neighbor, and its class is 3. So the category of the unknown point is also class 3. When k = 3, the number of class 2 is 2, and the number of class 3 is 1, so the category of the point is 2. By definition, we can know that different values of k will have an impact on the determination of unknown point’s categories. Of course, excessive k values can affect the results. If a smaller k value is chosen, it means training instances in smaller neighborhoods. Only those instances close to the unknown point will affect the prediction results. The disadvantage of a smaller k value is that learning estimation errors will increase, and results will be sensitive to the instances of close neighbors. If the adjacent instance points happen to be noisy, the prediction will be wrong. In other words, the reduced k means that overall model becomes complex, which is prone to overfitting. If a large k is chosen, it means training instances in larger neighborhoods. The advantage is that estimation errors of training can be reduced. But the disadvantage of a larger K value is that approximate errors will increase, because the training instances far from the input one will now affect the prediction results. Increasing the k means that overall model become simple. And if it is deployed on Hadoop for prediction, the larger k increases the computational pressure on a single node. For a dataset with multiple number of records, both the computation speed and load pressure on the service node will increase more. Therefore, choosing a suitable k can yield a more reliable prediction result.

However, its value involved in this paper is fixed after many experiments, so it is not within the optimization range of this paper.

As we can see from Fig. 7, when determining whether the unknown point and known points are of the same class, the distance between the two is also determined. The distance calculation is one of optimizations we made in this experiment. Let's first introduce the default distance calculation formula of native KNN compared with PEWM_G KNN in this experiment, the Euclidean Metric:

$$\rho = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{1}$$

In mathematics, Euclidean Metric is a distance or an ordinary distance (straight line) between two points in Euclidean space. It is a commonly adopted distance definition, meaning the true distance between two points in an m-dimensional space, or the natural length of vectors. In two and three dimensions, it also means the actual distance between two points. Suppose there are two points (x_1, y_1) and (x_2, y_2) in a 2-dimensional space, ρ is the Euclidean Metric between them. So the formulas for three and n-dimensional Spaces can be derived as follows:

$$\begin{aligned} \rho &= \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \\ \rho &= \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \end{aligned} \tag{2}$$

As can be seen from above formulas that the Euclidean Metric is actually the simplest direct calculation formula of the distance between two points. Although it is better understood, this does not mean it's applicable to all statistical problems. The real issue is that in most statistical cases, its distance is unsatisfactory. Because it doesn't take into account differences between different variables. For example, the value range of the variable A is 0 to 1, and the value range of variable B is 0 to 10,000, so that a small difference in B will outweigh the differences in A, making a higher impact on the calculated Euclidean distance. That is, when the coordinates represent measurements, they tend to have random fluctuations of varying magnitude. When each variable is a quantity of different properties, the ''distance'' is sensitive to units of variables. Euclidean Metric doesn't consider the difference between different attributes (indicators or variables) of a point, thus fails to meet the practical requirements of some use cases. Therefore, based on this finding, this paper will use Pearson metric instead of Euclidean Metric distance to better calculate the classification distance in terms of similarity.

B. PEARSON CORRELATION COEFFICIENT

In statistics, Pearson correlation coefficient, also known as Pearson's product moment correlation coefficient (PPMCC or PCCs), is used to measure the correlation (linear correlation) between two variables X and Y. It is defined as follows:

$$\rho_{XY} = \frac{Cov(X, Y)}{\sqrt{D(X)}\sqrt{D(Y)}} = \frac{E((X - E(X))(Y - E(Y)))}{\sqrt{D(X)}\sqrt{D(Y)}} \tag{3}$$

E is the mathematical expectation or mean, D is the variance, $\sqrt{D}$ is the standard deviation, E((X-E(X)(Y-E(Y)) is called the covariance of random variables X and Y, as Cov(X,Y). And the quotient of the covariance and standard deviation between two variables is called the correlation coefficient of the random variables X and Y, as ρ_{XY}. Therefore, Pearson correlation coefficient can also be expressed by the following formula:

$$r = \frac{\sum_{i=1}^{n}((X_i - \overline{X})(Y_i - \overline{Y}))}{\sum_{i=1}^{n}\sqrt{(X_i - \overline{X})^2}\sum_{i=1}^{n}\sqrt{(Y_i - \overline{Y})^2}} \tag{4}$$

And r can also be estimated by the standard score mean of the sample point to obtain an expression equivalent to the above equation:

$$r = \frac{1}{n-1}\sum_{i=1}^{n}(\frac{X_i - \overline{X}}{\sigma_X})(\frac{Y_i - \overline{Y}}{\sigma_Y}) \tag{5}$$

$\frac{X_i - \overline{X}}{\sigma_X}$, $\overline{X}$ and σ_X are the standard score, sample mean and sample standard deviation, respectively.

We can try to understand Pearson correlation coefficients in term of the matrix of covariance. We assume that two groups of vectors X and Y, both contain n elements, then calculating the covariance of both can be recorded as the following formula:

$$Cov(X, Y) = \frac{\sum_{i=1}^{n}((X_i - \overline{X})(Y_i - \overline{Y}))}{n} \tag{6}$$

Let's analyze the numerator first. The numerator is positive when either both X and Y are greater than the average X or Y respectively, or when both X and Y are less than their average respectively. Otherwise, the numerator is negative. If the data is more complex and the positive and negative cancel out, the covariance is closer to 0. Then we can't tell whether X and Y are related. However, if the two variables are correlated, then the covariance is a large negative or positive value. There is also a dimensional problem: assuming that X is around 0.0001, but Y is around 10000, then the covariance is excessively affected by Y (this is the defect of European Metric). To eliminate the dimensional problem, in Equation 4, X and Y are normalized by subtracting their respective means and dividing their variances. The net result is that the difference in different dimensions is removed from the calculation. This operation is also called z-score normalization, which means by transforming data of different magnitude into scores of uniform measures, the data comparability is improved, and the results are not affected by the different magnitude.

The range of values Pearson correlation coefficient takes is from -1 to 1. The larger the absolute value, the higher

the correlation between X and Y. When X and Y are linearly correlated, the correlation coefficient is 1 (positive linear correlation) or −1(negative linear correlation). When the correlation coefficient is 0, X and Y variables are unrelated. Therefore, our first step of optimization in KNN is to replace the Euclidean Metric with the absolute value of Pearson correlation coefficient, which solves the dimensional problem of distance from the similarity and simplifies the measure comparison. The corresponding relationship between the range of absolute value and similarity degree is as follows:

C. EWM

Entropy weight method(EWM) is an objective assignment method. Entropy is a physical concept of thermodynamics, which is a measure of system disorder. The larger the entropy is, the more chaotic the system is (and the less information it carries). The smaller entropy means the system is more orderly (and it carries more information). Therefore, information entropy draws from the concept of entropy to describe the size of event information on average. Mathematically, information entropy is the expectation of the amount of information contained by an event(or mean, is the probability of each possible outcome in a trial multiplied by the sum of its outcome). In the process of specific use, EWM calculates the entropy right of each index according to the degree of variation, and then corrects weights through the entropy right, in order to obtain the more objective weight. The implementation steps of EWM are as follows:

(1) First, it is necessary to check if there are negative numbers in the input matrix, and if so, to re-standardize to the non-negative interval (each element is non-negative when calculating the probability later). Assuming that there are n objects to be evaluated and m evaluation indicators (which have been positively normalized), the positively normalized matrix formed by them is as follows:

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \tag{7}$$

Assuming the normalized matrix be Z, and the element in Z can be represented by z_{ij}:

$$z_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=1}^{n} x_{ij}^2}} \tag{8}$$

where i is the position of the evaluation object, and j is the position of the evaluation indicator. After this, EWM will also check whether there is a negative number in the Z matrix. If there is, it is necessary to use another standardization method for X to obtain the Z matrix. The standardized formula is:

$$z_{ij} = \frac{x_{ij} - \min\{x_{1j}, x_{2j}, \ldots, x_{nj}\}}{\max\{x_{1j}, x_{2j}, \ldots, x_{nj}\} - \min\{x_{1j}, x_{2j}, \ldots, x_{nj}\}} \tag{9}$$

TABLE 1. The corresponding relationship between the range of absolute value and similarity degree.

Value range	Similarity degree
0.8-1.0	Highly
0.6-0.8	Strong
0.4-0.6	Moderate
0.2-0.4	Weak
0.0-0.2	Very weak or no

At this point, the normalization of the matrix X is over. The purpose of this step is to compress the data within the range of [0,1], without negative numbers, and to keep the mathematical units between the data consistent. This is also one of the effective methods to solve the dimensional problem.

(2) Calculating the proportion of the ith sample of the jth indicator and regarding it as the probability used in relative entropy calculation. Assuming that there are n objects to be evaluated and m evaluation indicators, the non-negative matrix obtained after the previous step is as follows:

$$Z = \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1m} \\ z_{21} & z_{22} & \cdots & z_{2m} \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ z_{n1} & z_{n2} & \cdots & z_{nm} \end{bmatrix} \tag{10}$$

The probability matrix P is then calculated, and the formula of each element pij in it is as follows:

$$p_{ij} = \frac{z_{ij}}{\sum_{i=1}^{n} z_{ij}} \tag{11}$$

The sum of each column is guaranteed to be 1, that is, the corresponding probability sum of each index is 1.

(3) Calculating the information entropy of each indicator, the information utility value and normalizing it to obtain the entropy weight of each indicator. For the jth indicator, the information entropy is calculated as follows:

$$e_j = -\frac{1}{\ln n} \sum_{i=1}^{n} p_{ij} \ln(p_{ij}), \quad (j = 1, 2, \ldots, m) \tag{12}$$

The information utility value is as follows:

$$d_j = 1 - e_j \tag{13}$$

Then the larger information utility value is, the more information it corresponds to. By normalizing these information utility values, we can get the entropy weight for each indicator:

$$w_j = \frac{d_j}{\sum_{j=1}^{n} d_j} \tag{14}$$

Generally speaking, the smaller an indicator's information entropy is, the more valuable the indicator is, and the more

information it can provide. Such an indicator can play greater role in the comprehensive evaluation, and get assigned more weight. Entropy was first introduced by Shannon, and has been widely used in engineering technology, social economy and other fields. In the specific use, according to the dispersion degree of data of each indicator, the information entropy calculates the entropy right, and then corrects it, so as to get a more objective weight.

D. GAUSSIAN FUNCTION

Before introducing the Gaussian function, we need to understand a drawback of the original KNN: if the number of samples in the original dataset is unbalanced, it is easy to conclude that the participation probability of each category is always higher for the category with the larger number of samples. For example, the dataset has 1 letter 'a' and 99 letters 'b', assuming a sample of 'a' and with a k value set to 5. So no matter how similar the test array is to the 'a', the proportion of category 'b' is always the largest of the final k neighbors. If so, the algorithm failed. To avoid this, we can weight KNN. The idea is: neighbors with smaller distances from the sample have higher weights. So the Gaussian is introduced:

$$f(x) = ae^{-\frac{(x-b)^2}{2c^2}} \tag{15}$$

where, a, b, and c are real constants, and a > 0. A is the height of curved spike, b is the coordinate of the center of spike, and c is called the standard variance, characterizing the width of bell curve. The Gaussian function is a normally distributed density function with the following figure:

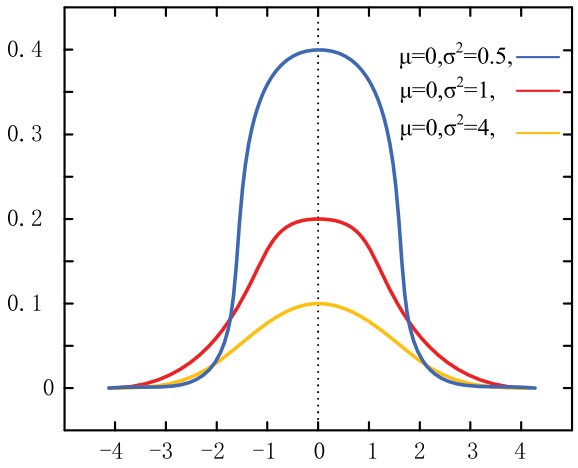


FIGURE 8. The gaussian function.

These corresponding parameters are b = μ, c = σ, and a = $\frac{1}{\sigma\sqrt{2\pi}}$. We can see from Fig. 8 that if the closer the known categories are to the central point, namely the test point in the real sample, the larger their corresponding Gaussian function is. Conversely, the smaller Gaussian function indicates that the known categories point is farther away from the test point.

If we know the k nearest neighbors, we can then calculate their weight values, and thus determine the category of true unknown points. For example, if k = 5, the categories of k close neighbors around the unknown point are $\{A, A, A, B, B\}$. There are 3 A and 2 B. Generally speaking, the discrimination result is A. However, in the weighted case, the weights of five nearest neighbors are $\{A(0.2), A(0.3), A(0.1), B(0.4), B(0.5)\}$. The weight equivalent to A is 0.6, B is 0.9, so B is selected as the category of unknown point. Therefore, Gaussian function well solves the problem caused by different classes and number imbalance of the original KNN algorithm.

E. PEWM_G KNN

From the above four subsections in this section, the optimization steps of the KNN algorithm are as follows:

(1) Replacing Euclidean Metric with Pearson correlation coefficient. The purpose is to eliminate dimensional problems brought by the units of dataset properties, so that data of different orders of magnitude are converted into scores of uniform measures for comparison so that results are unaffected by the calculated magnitude.

(2) On the basis of Pearson correlation coefficient, entropy weight method is introduced. When the two values X and Y belong to the same attribute (same column) in the test set and the training set for the distance calculation, multiply the weight calculated by their entropy weight method. Based on Equations 4 and 14, the resulting new attribute model is as follows:

$$r_j = \frac{\sum_{j=1}^{n}((w_jX_j - w_j\overline{X})(w_jY_j - w_j\overline{Y}))}{\sum_{j=1}^{n}\sqrt{(w_jX_j - w_j\overline{X})^2}\sum_{j=1}^{n}\sqrt{(w_jY_j - w_j\overline{Y})^2}} \tag{16}$$

where,

$$\sum_{j=1}^{n} w_j = 1 \tag{17}$$

In Equation 16 and 17, j is the jth indicator or the jth attribute (or column j) in the dataset, and the sum of weights for all attributes (all columns) of dataset is 1. On the basis of eliminating dimension, the entropy weight method calculates the weight of each attribute according to its degree of variation. Then each attribute of dataset gets a more objective distance indicator multiplied by the weight. Finally, they will be put into Pearson's formula to calculate the true distance in Fig. 7. So far, above two steps are the optimization of distance calculation before selecting k nearest neighbors in KNN.

(3) After the above two-step calculation, k neighbors are weighted using the Gaussian function. For example, by Equation 16 we can obtain the Pearson_Entropy Weight distance $\{r_1, r_2, \ldots, r_k\}$. On the basis of Equation 15, combined with Equation 16, we can know that the last weight formula of PEWM_G KNN is:

$$f(r_k) = ae^{-\frac{(r_k)^2}{2c^2}} \tag{18}$$

For k neighbors, if multiple neighbors are of the same class, the weight of this class needs to be superimposed together. The formula is as follows:

$$f(r_{k_i}) = f(r_{k_1}) + f(r_{k_2}) + \ldots + f(r_{k_{i-1}}) + ae^{-\frac{(r_{k_i})^2}{2c^2}} \quad (19)$$

And i is equal to the number of appearances of a certain class. Comparing Equation 15 and 18 we can know that x-b is also r, the Pearson correlation coefficient of the points of known species and the point tested. After that, we can get all the weights of the known classes in k neighbors in Equation 19, so as to achieve a more accurate prediction task. This step can not only solve the problem of algorithm failure caused by the imbalance of categories of a dataset, but also further refine the concept of distance and predict from a more reasonable and objective perspective. At this point, the algorithm optimization step is completely over.

F. ALGORITHM FLOW

From the above sections of this section, we can know models of the original KNN algorithm and the optimized PEWM_G KNN model. Next, we need to understand the steps in their algorithm programs. This section will explain their algorithm flows and analyze their own algorithm complexity in detail. First, let's take a look at the notations and annotations involved in the algorithm flow, as shown in the table below:

TABLE 2. Symbols and notations used in the present paper.

Notation	Meaning
Init()	Initialize a data set or collection
train	Training set
test	Test set
total	Total dataset
k	K-th nearest neighbor point
lab	Belonging category
get()	Get the data at the specified index of a dataset or collection
put()	Put the data into the specified index of a dataset or collection
Sort()	Sort the collection in positive order
Resort()	Sort the collection in reverse order
delete()	Delete an element from the collection
containsKey()	Determine whether the key is contained in a key-value pair
key()	Get the value at the specified index in a key-value pair
getWeight()	Set of weights of a specified dataset by entropy weight method

The first is the algorithm flow of the original KNN algorithm:

From Algorithm 1, we can know that after initializations of training set and test set, the training set is traversed (lines 1-10). Firstly, set of features is obtained, and cateogry *lab* of the current instance is recorded(because cateogries of the dataset covered in this paper are in the last column, the index value is *features.size - 1*), then *features* then removes *lab*(lines 2-4). The second step is to traverse the test set and obtain curr_test, the feature set of a specified row index of test set, and then iterate it and add the corresponding Euclidean calculation formula of Equation 2 to tmp. Finally, *key_value* is recorded as a collection of tuples*(tmp,lab)*(lines 5-10). The third step is to sort *key_value* by tmp from small to large. After the *k* is determined, the *lab* of *k* positions in *key_value* can be

Algorithm 1 KNN

Input:
The set of training set, *Init*(*train*);
The set of test set, *Init*(*test*);
Output:
The set of K neighbors, *result*;

```
1:  for i in train.size:
2:      features = train.get(i)
3:      lab = features.get(features.size − 1)
4:      features.delete(lab)
5:      for j in test.size:
6:          curr_test = test.get(j)
7:          tmp = 0
8:          for k in curr_test.size:
9:              tmp += (curr_test.get(k) - features.get(k))^2
10:         key_value.put(tmp, lab)
11: Sort(key_value)
12: Init(k)
13: for n in (key_value.size & n < k):
14:     result.put(key_value.get(n))
15: return result;
```

added to *result*(lines 11-15). At the end of KNN, the result of this prediction can be obtained by counting categories with the largest number of *result*.

The second is the algorithm flow of the PEWM_G KNN algorithm:

From Algorithm 2, we can know that after the initialization, we also need to use entropy weight method to calculate the set of weights corresponding to feature values of total dataset *weights*, namely, the result of Equation 14. The first step is the same as KNN, from lines 1 to 7. In the second step, N is the number of features of the current number of lines in the test set, weight is *wj* in Equation 16, SumX, SumY, SumX_sq, SumY_sq, SumXY, a is the numerator, and b is the denominator of Equation 16(lines 8-21). If b is 0, *tmp* is set to 0. Then, *key_value* is recorded as a collection of tuples*(tmp,lab)*, and it is sorted by *tmp* from large to small(lines 22-28). The third step is to assign weights to k nearest neighbors by gaussian function. After initializing *k* and *label_dis*, we can get the category of a current k-nearest neighbor *cur_l*. Then, if the initialized *label_dis* does not contain *cur_l*, we can store (*cur_l*,0) in it. But if *label_dis* contains the current *cur_l*, we can change the value at it to a sum of superimposed Gaussian function, which is the result of Equation 19(lines 29-35). At the end of the algorithm, the type of a maximum value in *label_dis* is selected.

Algorithms 1 and 2 not only help us understand flows and details of KNN and PEWM_G KNN, but also show the algorithm complexity. If we have *N* samples, and each sample is characterized by a vector in *D* dimensions. For any target sample in Algorithm 1, its prediction needs to cycle all test samples, and the complexity is *O(N)*(N can be viewed as *test.size*). In addition, when we calculate the distance between two samples, another complexity depends on the

Algorithm 2 PEWM_G KNN

Input:

The set of training set, *Init*(*train*);
The set of test set, *Init*(*test*);
The set of total dataset, *Init*(*total*);
The set of weights by entropy weight method, *weights* = *getWeight*(*total*);

Output:

A collection of key-value pairs used to store category and distance measures, *label_dis*;

```
for i in train.size:
    features = train.get(i)
    lab = features.get(features.size − 1)
    features.delete(lab)
    for j in test.size:
        curr_test = test.get(j)
        tmp = 0
        SumX = SumY = SumX_sq = 0
        SumY_sq = SumXY = weight = 0
        N = curr_test.size
        for k in curr_test.size:
            weight = weights.get(k)
            SumX += weight * curr_test.get(k)
            SumY += weight * features.get(k)
            SumX_sq += (weight * curr_test.get(k))^2
            SumY_sq += (weight * features.get(k))^2
            SumXY += weight * curr_test.get(k) *
            weight * features.get(k)
        a = SumXY - SumX * SumY / N
        b = sqrt((SumX_sq − SumX^2) * (SumY_sq − SumY^2))
        / N
        tmp = |a/b|
        if b == 0:
            tmp = 0
        key_value.put(tmp, lab)
Resort(key_value)
Init(k)
Init(label_dis)
for n in (key_value.size & n < k):
    cur_l = key_value.get(n)
    if(!label_dis.containsKey(cur_l))
        label_dis.put(cur_l, 0)
    label_dis.put(cur_l, label_dis.get(cur_l)
    +e^(−(key_value.key(n))^2)/2)
return label_dis;
```

characteristic dimension, which is *O(D)*(D can be viewed as *curr_test.size*). So the total complexity is the product of them, which is *O(N*D)*. For convenience, we assume that N = D, and the complexity of the algorithm can be expressed as $O(N^2)$.

For Algorithm 2, the outer loop doesn't change, there's still N samples. For the inner loop, calculating the distance between samples only adds multiple variables to the memory(lines 12-18 in Algorithm 2), with no additional loop increment. The inner complexity can still be expressed as *O(D)*. In other words, for PEWM_G KNN, its algorithm complexity the same as that of KNN. However, this new algorithm adds many variables to receive weights of entropy weight method and Pearson coefficient parameters, while the original algorithm does not. Therefore, the space complexity changes from *O(1)* to *O(N)*, where N refers to the number of eigenvalue weights obtained by entropy weight method during initialization, as well as multiple variables in the memory cycle (lines 12-18 in Algorithm 2). So In order of magnitude, the space complexity is *O(N)*.

From the analysis of above two paragraphs, it can be seen that PEWM_G KNN is an algorithm to solve dimensional problems and simplify metric comparison by adding eigenvalue weights of instances, Gaussian functions of k-nearest neighbors and Pearson coefficients at the cost of spatial memory. Subsequent experiments will verify whether it has any effect on improving prediction rates of datasets.

V. COMPARISON OF ALGORITHM OPTIMIZATION

After introducing the algorithm model and optimization steps, the next experiment is the algorithm test on Hadoop.

A. EXPERIMENTAL ENVIRONMENT

As mentioned in the previous Sect. III, the experimental architecture of this study is constructed with reference to Fig. 5. And based on this, it completes the true Hadoop high availability environment combined with the YARN proposed in Fig. 3. Its environmental architecture is composed as follows:

On the premise of satisfying the perfect architecture, the hardware configuration of each node also needs to be rationalized to make the cluster environment better. To do this, setting up hardware for the four nodes in Table 3:

Table 4 shows the hardware configurations for Node01, 02, 03, 04 from top to bottom. It is important to note that Node01 does not need to be allocated much memory, because it mainly sets up configuration files and global variables for the experimental cluster on command lines in the subsequent experimental steps and has fewer controls in Table 3. Node02, 03, and 04 have management resource scheduling configuration in YARN, in order to work with datasets with different number of records. To ensure these different datasets can be processed successfully, they are assigned more memory space and CPU cores in order to achieve high performance processing capacity and I/O disk operation. The configuration of platform parameter optimization mentioned in next section is also related to these three nodes.

B. EXPERIMENTAL PROCEDURE

Now that we understand the experimental environment and hardware configuration, let's introduce the main steps of our experiment:

TABLE 3. Environmental architecture.

	HDFS[1]		YARN[2]		HA[3]		
Node	NN	DN	RM	NM	ZK	ZKFC	JN
Node01	✓					✓	✓
Node02	✓	✓		✓	✓	✓	✓
Node03		✓	✓	✓	✓		✓
Node04		✓	✓	✓	✓		

Note: In HDFS(Fig. 2), NN represents NameNode, and DN represents DateNode. In YARN(Fig. 3), RM represents ResourceManager, and NM represents NodeManager. In HA(Fig. 5), ZK represents ZooKeeper, ZKFC represents ZooKeeperFailoverController, and JN represents JournalNode.

1.Hadoop core data control.

2.Hadoop resource scheduling control.

3.Hadoop highly available control.

TABLE 4. Hardware configuration of the nodes.

IP	OS	CPU	Mem	Disk
192.168.88.36	CentOS6.5 64bit	1996.253MHZ single core	1GB	196GB
192.168.88.37	CentOS6.5 64bit	1996.253MHZ double cores	6GB	196GB
192.168.88.38	CentOS6.5 64bit	1996.253MHZ double cores	6GB	196GB
192.168.88.39	CentOS6.5 64bit	1996.253MHZ double cores	6GB	196GB

TABLE 5. Basic information about datasets.

Dataset	Type	Class	Feature	Instance
iris	Numerical	3	4	150
breast cancer diagnosis	Numerical	2	31	569
dry soybean	Numerical	7	16	4084
pulsar	Numerical	2	8	11307

(1) Prepare the datasets. All datasets involved in the algorithm are from UC Irvine [9], including iris [10], breast cancer diagnosis [11] in Wisconsin Island, USA, dry soybean [12] and pulsar [13] datasets. The following table provides some basic information about their four datasets:

All datasets we used in this test are classified datasets. Our focus is on the prediction rate the optimized algorithm has for these different datasets. The specific classes and features of these datasets are not the consideration of the control variables in this paper. In order to reflect the applicability of the optimized algorithm, four datasets with different application backgrounds are chosen.

(2) After getting the data for each dataset, it was divided into a 30 percent test set and a 70 percent training set through Pycharm. Such operations are repeated 15 times, and 15 combination of test and training sets is output, ensuring that the predicted data results are representative.

(3) Next up, we need to implement the MapReduce algorithm. We need to write three algorithm/programs to compare the algorithm optimization steps and their respective results. The first one is the original KNN algorithm, whose distance metric and the weighted optimization have not yet been modified, and the default distance measure is the Euclidean Metric. The second is the distance measure optimized by the Pearson correlation coefficient and entropy weight method added to the original KNN. On the basis of solving the dimensionality problem in theory, it is compared with the original algorithm. The third is to add gaussian function on the basis of the second, which is used to carry out weighted optimization for k nearest neighbors, so as to determine the class of unknown points more reasonably and compare with the original algorithm. We named the first program as KNN, the second as PEWM KNN, and the third as PEWM_G KNN.

(4) Lastly, we packaged the three algorithms into jar packages, and uploaded them to the Hadoop cluster, and tested them with the 15 combinations of test and training datasets in step 2. HDFS, YARN and HA were then turned on, and we tested every combination with the three algorithms respectively for each dataset, and received 45 results. Then the data visualization is used to show the change trend of the data and compare it with others.

C. INTERPRETATION OF RESULT

After the algorithm prediction tasks were completed, 45 prediction results of each of the four datasets used in this paper were visualized. Their figures are as follows:

First, we can see from Fig. 9 that for iris dataset, the prediction rate curves of the PEWM KNN is higher than the original KNN curve, and the final PEWM_G KNN curve is the highest. It shows that for this dataset, both the optimization of the distance measurement and the Gaussian

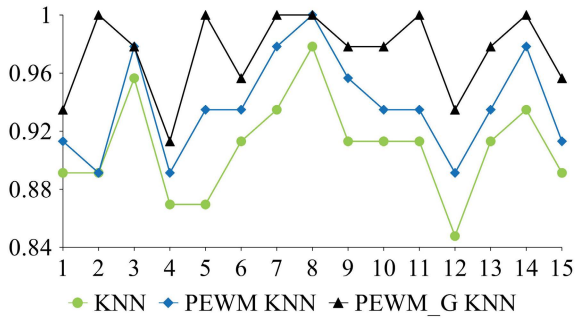


FIGURE 9. The prediction rate comparison for dataset "iris."

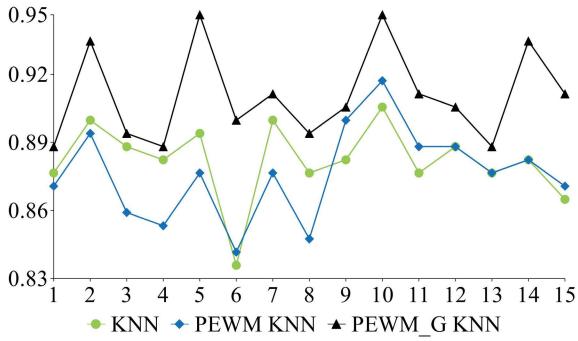


FIGURE 10. The prediction rate comparison for dataset "breast cancer diagnosis."

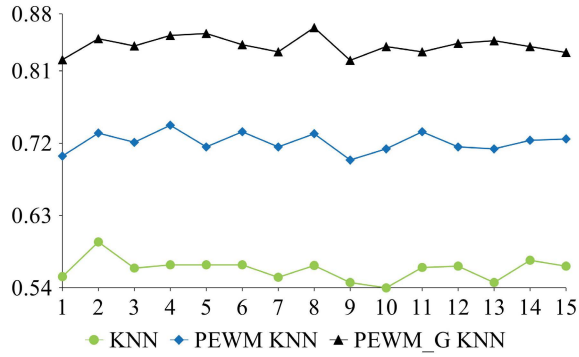


FIGURE 11. The prediction rate comparison for dataset "dry soybean."

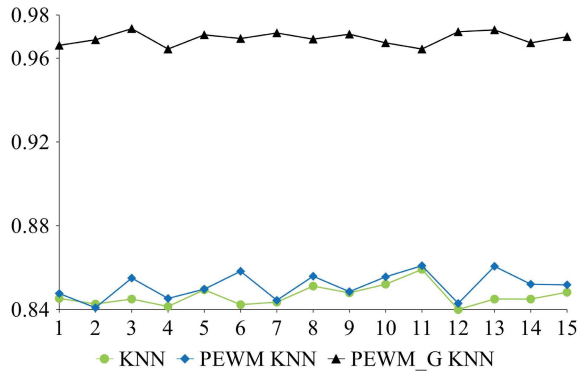


FIGURE 12. The prediction rate comparison for dataset "pulsar."

optimization contributed to the prediction rate improvement. In order to check if there are any limitations in any of the optimization steps, we further analyzed other datasets.

From Fig. 10, we can see that the original KNN algorithm outperformed the PEWM KNN algorithm in some of the tests. However, the PEWM_G KNN curve still outperforms the other two in all tests, and there is a larger difference between its performance and that of the PEWM KNN when compared to the test result shown in Fig. 9. Then we can preliminarily conclude that adding Gaussian Optimization to PEWM_G KNN contributed to its better performance. By comparing only the curves of KNN and PEWM_G KNN in the above two figures, we can conclude that PEWM_G KNN does improve the prediction rate.

As shown in the curve change diagram of Fig. 11, we can see that the advantages of the optimization algorithm are gradually emerging. The curve of PEWM KNN is significantly higher than that of KNN, and the curve of PEWM_G KNN remains at the highest position. This figure not only proves that the optimization of distance metric improves the prediction rate, but also again verifies that the overall prediction rate of the algorithm remains at the highest after the addition of Gaussian optimization. And it can also be seen that the dry soybean dataset is also a great dataset to reflect the effect of every optimization step in this study.

Finally, the curve of PEWM_G KNN remains at the highest position, as shown by the curve change in Fig. 12. Although there is little difference between PEWM KNN and KNN curves after distance measurement optimization, PEWM KNN does have a relatively better prediction rate. Moreover, by comparing amplitudes of curve changes in the above four graphs, we can see that the prediction rate is also gradually leveling off.

However, based on Fig. 9-12, we can see that KNN with entropy weight method and Pearson coefficient has low stability of prediction. In Fig. 9 and 12, the curve of PEWM KNN fluctuates greatly and its prediction is unstable. And in Fig. 10, its prediction rate is not high. Except in Fig.11, PEWM KNN has an obvious prediction effect. It shows that the optimization combination of Pearson coefficient and entropy weight method needs further research and further optimization in the future. However, for PEWM KNN with Gaussian function, its algorithmic advantage is obvious. Although the curve fluctuation of PEWM_G KNN is still large in Fig. 9 and 10, its prediction rate remains at the highest. It shows that PEWM_G KNN improves the prediction rate, although the prediction stability is not high for the dataset with small number of instances. In Fig. 11 and 12, PEWM_G KNN also maintains high predictive stability. It shows that when the number of instances increases and the environment is constantly approaching the real application one, the PEWM_G KNN prediction rate not only remains higher, but also improves the stability. Therefore, the optimization algorithm proposed in this paper is effective

in increasing the prediction rate. The prediction curves of PEWM_G KNN and KNN are drawn separately, as shown below:

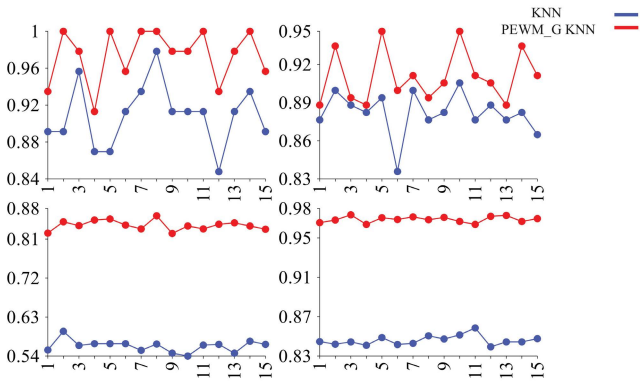


FIGURE 13. Comparison in KNN and PEWM_G KNN.

At this point, we finish the discussion of function optimization on prediction rate. As can be seen from Fig.13, PEWM_G KNN is indeed superior to the original KNN in terms of prediction rate. And with the increase of instances from dataset "Iris" to "Pulsar", the optimization effect is more and more obvious and the stability is higher and higher.

Of course, the above figures shows the comparison between the original KNN and PEWM_G KNN. But what about the result if we compare the new algorithm with other common classification algorithms? Therefore, PEWM_G KNN and KNN, Decision Tree [26], Adaboost [58] and Random Forest [40] were compared. The number of predictions for each algorithm is still 45, that is, 15 for each dataset. The experimental results are shown below:

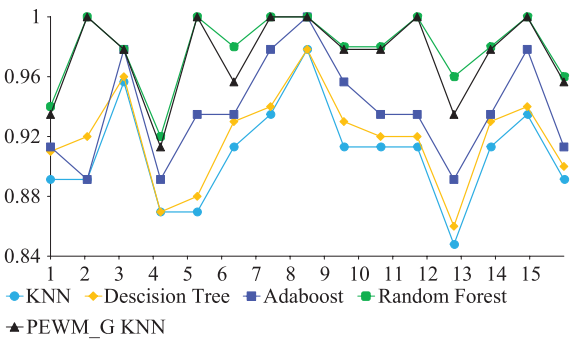


FIGURE 14. The prediction rate comparison for dataset "iris."

As shown in Fig. 14, no matter for these common classification algorithms or PEWM_G KNN, their accuracy curves fluctuate greatly, indicating that their predictions are not stable for datasets with small number of instances. Moreover, the prediction rate of PEWM_G KNN is not significantly different from that of Random Forest. It indicates that the advantage of PEWM_G KNN is not obvious for the dataset with small number of instances, and we can use another classification algorithm to replace it.

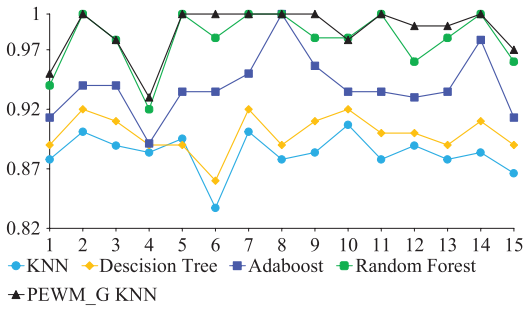


FIGURE 15. The prediction rate comparison for dataset "breast cancer diagnosis."

As can be seen from Fig. 15, when the number of instances began to increase, the amplitude of curve fluctuation gradually began to decrease. In addition, the prediction rate of PEWM_G KNN always remains the highest among several algorithms. Although the difference with Random Forest is not obvious, the advantages of its algorithm gradually become prominent. In Fig. 16 and 17, the advantage of PEWM_G KNN is obvious. Compared with other algorithms, the prediction rate of PEWM_G KNN always remains the highest, which is significantly different from that of Random Forest. And fluctuation of the curve is gradually stable with the increase of the number of instances.

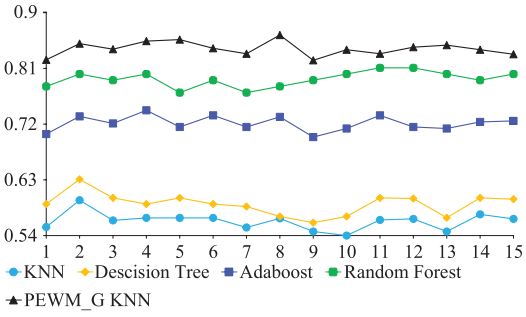


FIGURE 16. The prediction rate comparison for dataset "dry soybean."

To sum up, when the number of instances is small, the prediction rate of PEWM_G KNN is not high, and its prediction stability is low. So we can completely replace it with other classification algorithms. But when the number of instances increases and the environment is constantly approaching the real application one, the prediction rate of PEWM_G KNN reached the highest, and the stability of prediction gradually became higher. Therefore, the model of PEWM_G KNN is superior to other classifiers, such as KNN, Decision Tree, Adaboost and Random Forest. And the more large number of instances is, the more significant prediction effect of the model is.

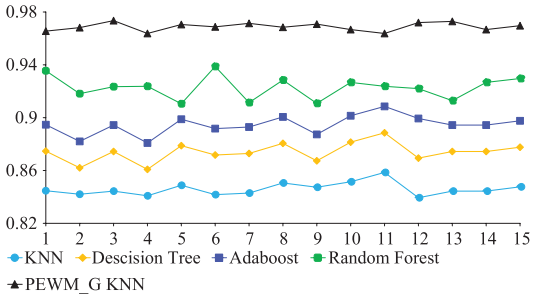


FIGURE 17. The prediction rate comparison for dataset "plusar."

VI. PLATFORM PARAMETER TUNING

Before performing platform parameter tuning and effect comparison, we need a general understanding of the algorithmic computational flow of the program ——MapReduce in Fig 6. When MapReduce runs on Hadoop, the platform parameters are closely related to its calculation process and the underlying algorithm. During parameter tuning, many exceptions and program errors are related to the MapReduce calculation process and details.

A. LIST OF PLATFORM PARAMETERS

As can be seen from e.g. [35], Hadoop platform has nearly 200 parameters, and not all of them need to optimized. In this study, whether the optimization parameters involved are necessary depends on the execution time of MapReduce and whether the execution process is smooth. Therefore, the parameters to be optimized were selected during prediction processes of four datasets in this paper. The list is as follows:

B. PARAMETERS OPTIMIZATION EXPERIMENT

The next step is to introduce and tune each parameter in Table 6. Of course, in the process of tuning, we strictly follow the rule of control variable and tune the parameters in table5 from top to bottom. The original tuning parameters of this experiment are shown in the table below:

1) mapreduce.task.io.sort.mb

This property manages the memory size of ring buffers, the kvbuffer of Map in Fig. 6. Its schematic diagram is as follows:

The map() function is an user provided function. After map() is called and the data processing is complete, the program usually calls outputCollector.collect() to output the results. Inside this function, it shards the generated $<$ *key*, *value* $>$, and writes them to a ring memory buffer in Fig. 6, 18. In kvbuffer, data input is divided into two main parts: 1) Metadata, which is used to record properties of the raw data. It contains the start position of the key, the start position of the value, and length of the value. 2) Raw data, which is the real data output by mapTask. It contains real values of key and value. In other words, the input of kvbuffer contains not only the value of key-value pair but also their address, separated by the equator. When the size of data in kvbuffer reaches or exceeds the threshold, it is written to disk. This leaves another clear area so that mapTask can be continued.

Now that we understand how kvbuffer works, we started with 200mb of memory and increased it by 200MB with each adjustment, and tested its effect on MapReduce execution time and algorithm prediction rate. The selected data set is "pulsar". On the basis of meeting the maximum number of records, parameters are tuned and the corresponding relationship between their values and measured data is as follows:

As shown in Fig. 19, the memory value of kvbuffer ranges from 800m to 1200m, and the prediction rate keeps rising until 1200m. When kvbuffer memory takes 1200m, Map, Shuffle and Reduce phase has the least time, which means that the MapReduce program has the minimum sum of execution times. So on the basis of meeting the maximum prediction rate of data set, saving the memory of ring buffer and successfully completing MapReduce execution, when

TABLE 6. List of optimized parameters.

Default	Location	Name
100m	mapred-site.xml	mapreduce.task.io.sort.mb
0.8	mapred-site.xml	mapreduce.map.sort.spill.percent
10	mapred-site.xml	mapreduce.task.io.sort.factor
Xmx200m	mapred-site.xml	mapreduce.map.java.opts
1024m	mapred-site.xml	mapreduce.map.memory.mb
Xmx200m	mapred-site.xml	mapreduce.reduce.java.opts
1024m	mapred-site.xml	mapreduce.reduce.memory.mb
False	mapred-site.xml	mapreduce.map.output.compress
8192m	yarn-site.xml	mapreduce.resource.memory.mb

TABLE 7. Original values of optimized parameters.

Default	Location	Name
800m	mapred-site.xml	mapreduce.task.io.sort.mb
0.8	mapred-site.xml	mapreduce.map.sort.spill.percent
10	mapred-site.xml	mapreduce.task.io.sort.factor
Xmx2200m	mapred-site.xml	mapreduce.map.java.opts
2750m	mapred-site.xml	mapreduce.map.memory.mb
Xmx200m	mapred-site.xml	mapreduce.reduce.java.opts
1024m	mapred-site.xml	mapreduce.reduce.memory.mb
False	mapred-site.xml	mapreduce.map.output.compress
8192m	yarn-site.xml	mapreduce.resource.memory.mb

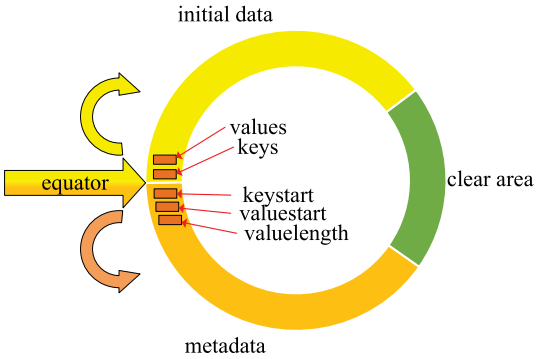


FIGURE 18. The schematic diagram of kvbuffer.

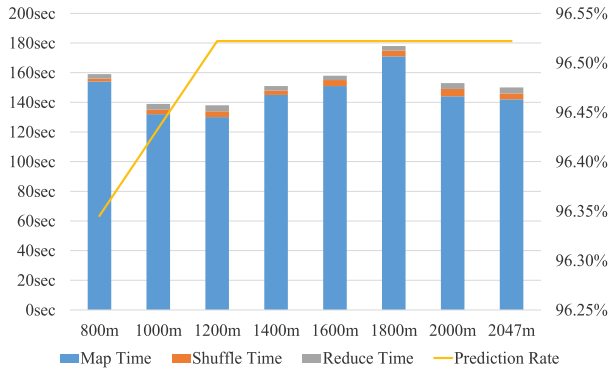


FIGURE 19. Comparison in mapreduce.task.io.sort.mb.

TABLE 8. Abnormal values.

Value	Exception
100m	Error: java.lang.ArrayIndexOutOfBoundsException: 1
200m	Error: java.lang.ArrayIndexOutOfBoundsException: 1
400m	Error: java.lang.ArrayIndexOutOfBoundsException: 1
600m	Error: java.lang.ArrayIndexOutOfBoundsException: 1
2200m	Error: java.io.IOException: Unable to initialize any output collector
2048m	Error: java.io.IOException: Unable to initialize any output collector

mapreduce.task.io.sort.mb's value is 1200 m, it can reach the best optimization effect.

However, we can also see from Table 13 that there are some abnormal values. From 100m to 600m, we can see that the system keeps reporting group index out-of-bounds exceptions, but there is nothing wrong with the program logic and code of MapReduce mentioned in this paper. And where the exception error have been locked in the reduce() method in the beginning, so we suspect that must be the scope of mapreduce.task.io.sort.mb is too small. Combined with attribute mapreduce.map.sort.spill.percent(the certain threshold of outputting data in kvbuffer)'s default value of 0.8, we made the analysis of the diagram below:

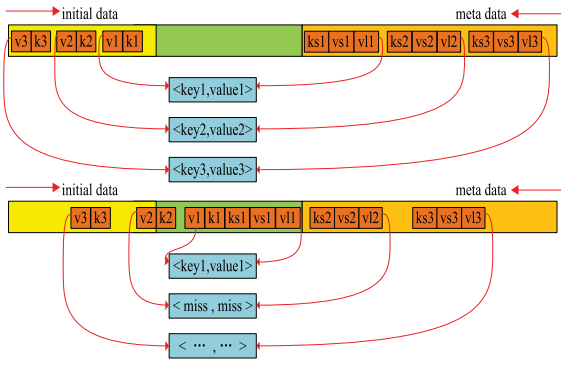


FIGURE 20. The analysis of kvbuffer exception.

In Fig. 20, when kvbuffer has sufficient memory and the threshold is reasonable, key-value pairs are sequentially output as the data reaches or exceeds the threshold. In addition, the original data and metadata will correspond one by one to form a complete key-value pair. However, if the threshold is too large, the buffer memory is too small, or the maptask is too large to handle. So when maptask starts, the data input to kvbuffer will soon reach the threshold, and the data printing will start. If the number of records in dataset is too large and there are too many maptasks, the output capacity of kvbuffer will not be able to catch up with the data processing capacity, resulting in the clear area will also be occupied or overflow. This will corrupt data, causing the output key-value pairs to be incomplete, either in the raw data or in the metadata. Therefore, index out-of-bounds exceptions occur in the reduce() method, which indirectly express data incompleteness.

The cause of the output collector initialization problem at 2200m in Table 13 is quite simple. After querying the documentation [36], we found that the maximum size of kvbuffer was 2047m. And after testing it, as shown in Figure 18, memory also reported output Collector initialization problems at 2048m. Therefore, in Fig. 19, the optimal value of this attribute is 1200m.

2) mapreduce.map.sort.spill.percent

This property, as mentioned in the previous one, specifies the threshold for the data output of kvbuffer. As can be seen from Fig. 19, the operations on the Map of datasets involved in this paper are relatively frequent and large in number. Therefore, in addition to choosing a relatively reasonable memory capacity, an appropriate threshold is also very necessary. We take the value from 0.1 to 1 at 0.1 interval to test its impact on MapReduce execution time and algorithm prediction rate. The comparison of detection results is shown below:

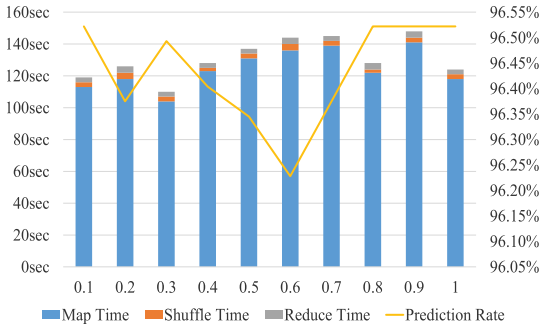


FIGURE 21. Comparison in mapreduce.map.sort.spill.percent.

As can be seen from Fig. 21, first of all, the choice of value 1 is not considered. If you wait until kvbuffer is filled with input data before output operation, there is a possibility of blocking. And if the number of dataset increases, exceptions like those in Table 13 and Fig. 20 are likely to occur again. Therefore, under the condition of ensuring highest prediction rate, a reasonable threshold value should be selected as far as possible. From 0.2 to 0.7, the prediction rate fluctuates and takes a long time,

and when the threshold is 0.8, the prediction rate is the highest and then tends to be stable. However, it can be seen from the figure that when the threshold is 0.1, its prediction rate is also the highest, and the running time tends to be low. From underlying principles of Fig. 6 and 18, if this value is too small, it will lead to too many spill times on Shuffle, resulting in increased I/O times. Therefore, in order to select the optimal configuration parameters from 0.1 and 0.8, we conducted a new monitoring of their data and ran them ten times respectively. The focus is on observation and comparison: 1) whether the prediction rate remains stable, 2) the spill time of Map and 3) the count of spill times.

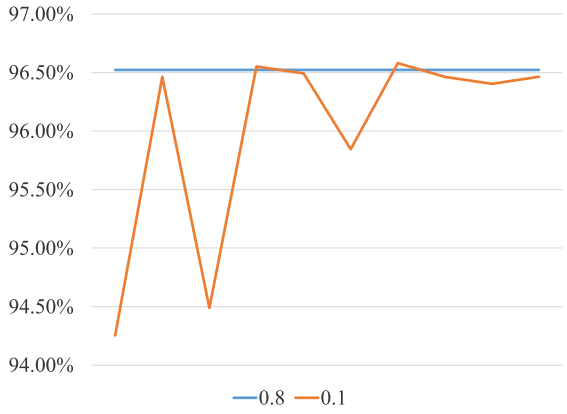


FIGURE 22. Comparison of prediction rate in mapreduce.map.sort.spill.percent.

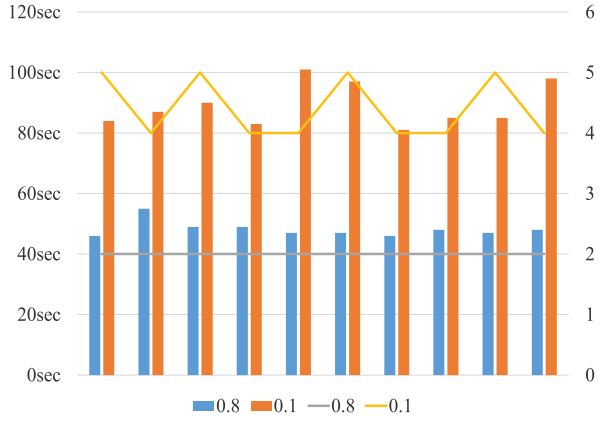


FIGURE 23. Comparison of spill time and count in mapreduce.map.sort.spill.percent.

As can be seen from Fig. 22, when the threshold value is 0.8, the prediction rate remains stable. However, when the threshold value is 0.1, the prediction rate is not consistent, and its stability is not good. Fig. 23 shows that the spill time of 0.8 is indeed much smaller than that of 0.1. And the spill count of 0.8 stays at 2, while it of 0.1 switches between 4 and 5. Therefore, the threshold value of 0.8 is the best choice to keep the prediction rate high, stable and the spill times, count short.

3) mapreduce.map.java.opts AND memory.mb

Mapreduce.map.java.opts and mapreduce.map.memory.mb are related, so we need to modify them both. The former is the startup parameter passed to Java Virtual Machine(JVM) when it starts, indicating the maximum amount of heap memory that Java program can use. Once this size is exceeded, JVM throws an out of memory exception and terminates the process. The latter, read and controlled by NodeManager, sets the memory upper limit for maptasks. When the memory required by map to execute tasks exceeds this value, NodeManager kills them. The size of the former is best set to 0.8 times of that of the latter. That is, the JVM heap setting values should be less than the memory upper limit for maptasks. Because we need to reserve memory for Java codes, it is generally recommended to reserve 20% of the memory. After testing the combination of two default values, we adjusted the values for every 400m increase in mapreduce.map.java.opts. The corresponding relationship between the two values is as follows:

TABLE 9. The corresponding relationship between mapreduce.map.java.opts and memory.mb.

map.java.opts	map.memory.mb	status
Xmx200m	1024m	java.lang.OutOfMemoryError
Xmx600m	1024m	java.lang.OutOfMemoryError
Xmx1000m	1250m	java.lang.OutOfMemoryError
Xmx1400m	1750m	java.lang.OutOfMemoryError
Xmx1800m	2250m	java.lang.OutOfMemoryError
Xmx2200m	2750m	Success
Xmx2600m	3250m	Success
Xmx3000m	2750m	Success
Xmx3400m	4250m	Success
Xmx3800m	4750m	Success
Xmx4200m	5250m	Success
Xmx4600m	5750m	Success

As can be seen from Table 9, the value of mapreduce.map.java.opts will report out of memory exceptions from 200m to 1800m, indicating that maptasks of the dataset used in this experiment has a large demand, which is more in line with the test conditions in the actual big data environment. Starting from 2200m, tasks run normally, indicating that values meet the memory requirements of the dataset. We stopped the test when mapreduce.map.memory.mb is 5750m, because Table 4 shows that the maximum memory limit for a NodeManager is 6GB, or 6144m. Then we drew the relationship diagram between different values of mapreduce.map.java.opts and the prediction rate, map time, spill time and spill count:

As can be seen from Fig. 24, the prediction rate remaines unchanged from 2200m to 4600m, but map time at 2200m and 2600m is the smallest. And in Fig. 25, the spill count remains constant and is 2, but the spill time at 2200m and 2600m is the smallest. Therefore, the map and spill time are tested again every 100m between 2200m and 2600m in order to select an appropriate value and maintain the predictive performance for the Map side (spill count and prediction rate were monitored and remained unchanged).

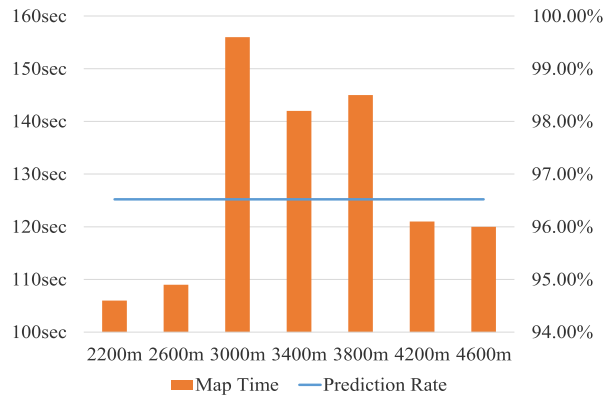


FIGURE 24. Comparison of map time and prediction rate in mapreduce.map.java.opts.

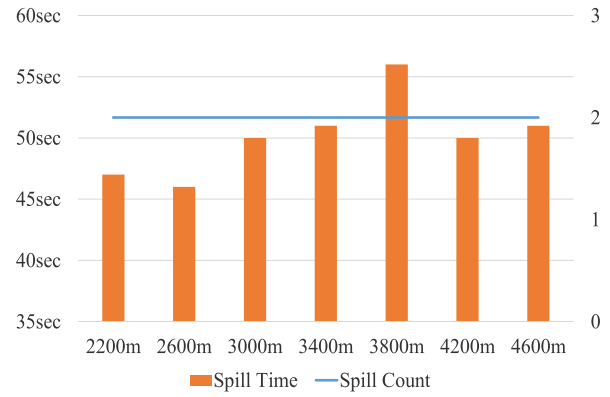


FIGURE 25. Comparison of spill time and count in mapreduce.map.java.opts.

TABLE 10. The corresponding relationship between mapreduce.map.java.opts and memory.mb.

map.java.opts	map.memory.mb	status
Xmx2200m	2750m	Success
Xmx2300m	2875m	Success
Xmx2400m	3000m	Success
Xmx2600m	3250m	Success

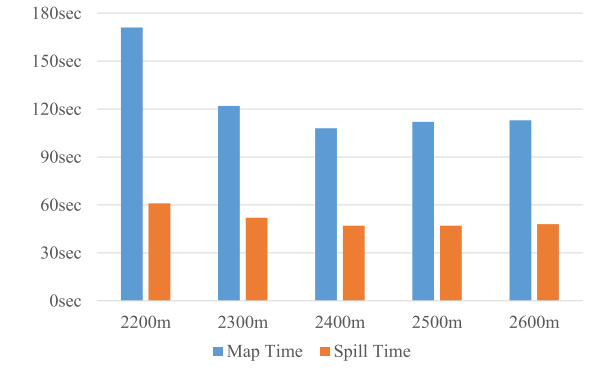


FIGURE 26. Comparison of map time and spill time.

Finally, we can learn from the Table 10 and fig. 26: when the mapreduce.map.memory.mb is 3000m and mapreduce.map.java.opts is 2400m, the current time consumption on map is minimal, and the prediction rate can be maintained at the highest level.

TABLE 11. The corresponding relationship between mapreduce.reduce.java.opts and memory.mb.

reduce.java.opts	reduce.memory.mb	status
Xmx200m	1024m	Success
Xmx600m	1024m	Success
Xmx1000m	1250m	Success
Xmx1400m	1750m	Success
Xmx1800m	2250m	Success
Xmx2200m	2750m	Success
Xmx2600m	3250m	Success
Xmx3000m	2750m	Success
Xmx3400m	4250m	Success
Xmx3800m	4750m	Success
Xmx4200m	5250m	Success
Xmx4600m	5750m	Success

4) mapreduce.reduce.java.opts AND MEMORY.MB

Mapreduce.reduce.java.opts and mapreduce.reduce.memory.mb are related, so we need to modify them both. These two attributes are essentially the same as those of the previous section, except that Mapreduce.reduce.java.opts is the JVM memory settings for the reduce phase, while mapreduce.reduce.memory.mb defines the upper limit of the memory for reducetasks. Of course, the size of the former is best set to 0.8 times of the size of the latter. In fact, from the data analysis in above three subsections and Fig. 19, 21, we can know that most of the time consumption in this experiment is not at the reduce end. However, for the sake of experimental rigor, we still need to test the value of these two attributes, and the value range relationship is as follows:

Then we stop the test when mapreduce.reduce.memory.mb is 5750m, because Table 4 shows that the maximum memory limit for a NodeManager is 6GB, or 6144m.

We then continue to monitor the same major data changes as in the above subsections. But here we need to note that one point: in Fig. 6, Reduce starts multiple fetch threads to fetch the map output when shuffle is executed to a certain proportion, stores and merges it to memory disks, so the change of fetcher time also needs to be checked. These changes are as follows:

Fig. 27 shows that changing JVM and task memory of reduce has no impact on the spill count and prediction rate. As can be seen from Fig. 28, the change of reduce time is not great. But for fetcher time, the values remain low from 3000m to 3800m. Therefore, in order to select a more appropriate value, the change of main monitoring data was tested again every 100m from 3000m to 3800m. The value range of mapreduce.reduce.java.opts and mapreduce.reduce.memory.mb, and the change of data is as follows:

As can be seen from Fig. 29, from 3000m to 3800m, the total time consumption at 3000m is the least, and the fetcher time is also the smallest. Following the workflow shown in Fig. 6, we can know that proper JVM heap size

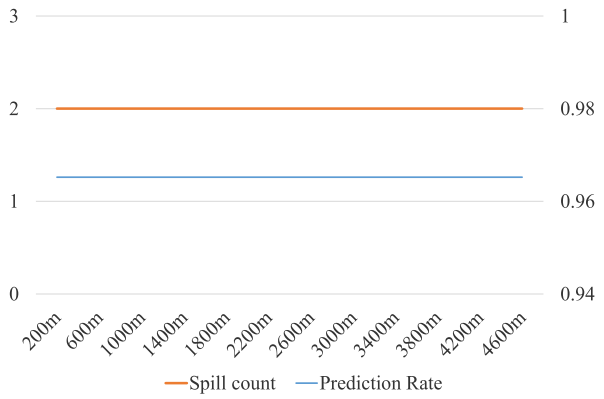


FIGURE 27. Comparison of spill count and prediction rate in mapreduce.reduce.java.opts.

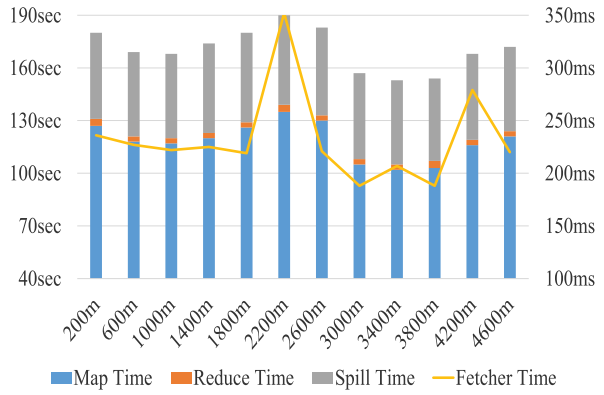


FIGURE 28. Comparison of different times in mapreduce.reduce.java.opts.

TABLE 12. The corresponding relationship between mapreduce.reduce.java.opts and memory.mb.

reduce.java.opts	reduce.memory.mb	status
Xmx3000m	3750m	Success
Xmx3100m	3857m	Success
Xmx3200m	4000m	Success
Xmx3300m	4125m	Success
Xmx3400m	4250m	Success
Xmx3500m	4375m	Success
Xmx3600m	4500m	Success
Xmx3700m	4625m	Success
Xmx3800m	4750m	Success

and reducetask memory configuration in reduce side can also greatly save the execution time of MapReduce program. Therefore, we know from the table that the most appropriate value combination is when mapreduce.reduce.java.opts is 3000m and mapreduce.reduce.memory.mb is 3750m.

5) mapreduce.map.output.compress

This attribute specifies whether compression is enabled during the execution of the MapReduce program. Using data compression is very important under Hadoop, especially when data is large and workload intensive, because I/O operations and network data transfers take a lot of time.

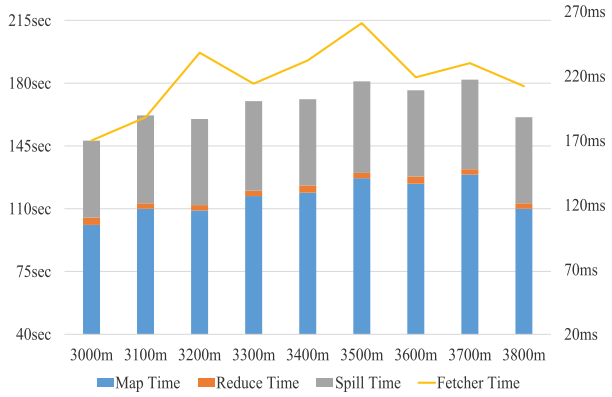


FIGURE 29. Comparison of different times in mapreduce.reduce.java.opts.

In Fig. 6, the compression takes place in the merge phase at the Shuffle end. If the Map end spill complex and large amount of data, enabling the compression technology can effectively reduce the number of bytes read and written by the underlying storage system (HDFS), improving the network bandwidth and disk space efficiency. Therefore, in addition to monitoring changes in the prediction rate, we also need to monitor shuffle time and read and write bytes of HDFS. These variation data are as follows:

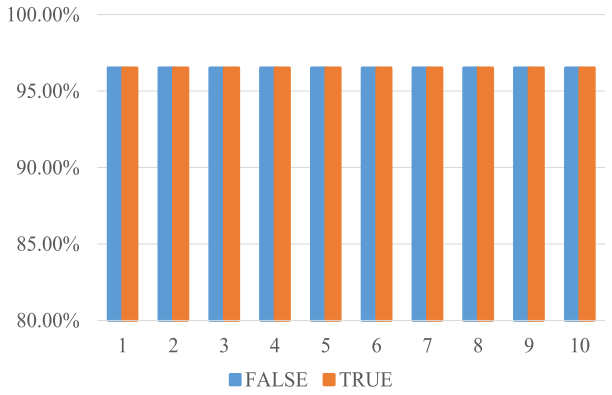


FIGURE 30. Comparison of prediction rate in mapreduce.map.output.compress.

Fig. 30 shows that the prediction rate of the dataset is not affected by the compression status. However, Fig. 31 shows that after the compression technology is turned off, shuffle time will increase significantly and the rate of change is not stable. And after it is turned on, the shuffle time was maintained at a relatively low and stable level. And as we can see from Fig. 32, the number of bytes read and written is significantly reduced when compression is enabled. This helps to reduce disk I/O, which will be helpful for network transmission. We believe the improvement will be more significant in the real world applications when hundreds of millions of megabytes of data is processed. Thus, considering all the above mentioned, the value of mapreduce.map.output.compress is selected as true.

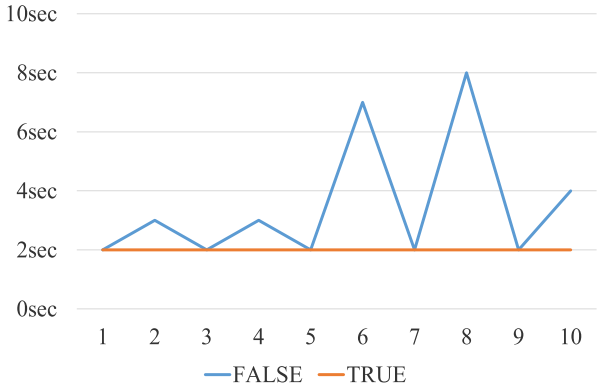


FIGURE 31. Comparison of shuffle time in mapreduce.map.output.compress.

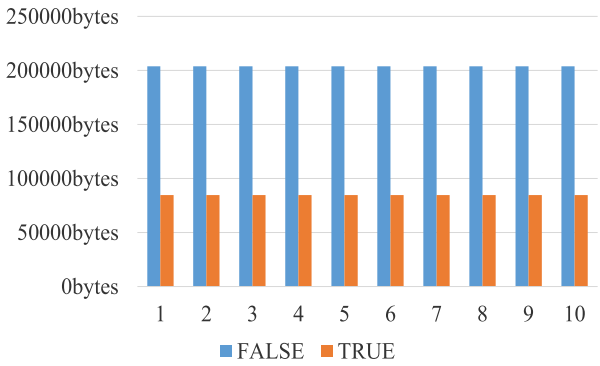


FIGURE 32. Comparison of bytes in mapreduce.map.output.compress.

6) nodemanager.resource.memory.mb

This attribute indicates the maximum amount of physical memory that YARN can use on nodes, that is, the memory of NodeManager, whose default value is 8 GB. YARN does not intelligently detect the total physical memory of nodes, so it is best to configure the NodeManager of memory before executing the MapReduce. However, it should be noted that JVM memory of map, reduce and the memory of maptask and reducetask must be less than or equal to the memory of the NodeManager. That is, their size relationship should satisfy the following formula:

$$\begin{aligned} &size(nodemanager.resource.memory.mb) \\ &\quad \geq size(mapreduce.reduce(map).memory.mb) \\ &\quad \geq size(mapreduce.reduce(map).java.opts) \end{aligned} \tag{20}$$

Next, we measure the prediction rate and map, shuffle, and reduce time at 1G interval, starting from the default of 8GB. The range of values and changes in the monitoring data are as follows:

In fact, we can see from the bottom three data in Table 2 that if the total physical memory of a single node is too small, the smoothness of MapReduce tasks will be affected, and even tasks will be stuck all the time or cannot be started. Therefore, it is necessary to choose the appropriate amount of physical memory. Of course, the value must be

TABLE 13. Abnormal values.

nodemanager.resource.memory.mb	status
8G	Success
7G	Success
6G	Success
5G	Success
4G	Success
3G	map 100%, reduce 0%
2G	map 0%, reduce stoped
1G	map stoped%, reduce stoped

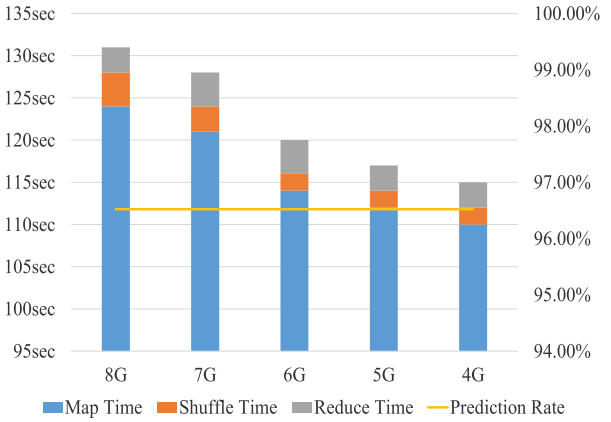


FIGURE 33. Comparison of time consumption and prediction rate in nodemanager.resource.memory.mb.

smaller than the available memory resources. Otherwise, the running efficiency of other resources on the node will be affected. In this experiment, the physical memory of a single NodeManager is 6G, so this attribute must be less than 6G if the operation performance is satisfied. However, task cannot run normally if it is less than or equal to 3G, so we need to select between 5G and 4G.

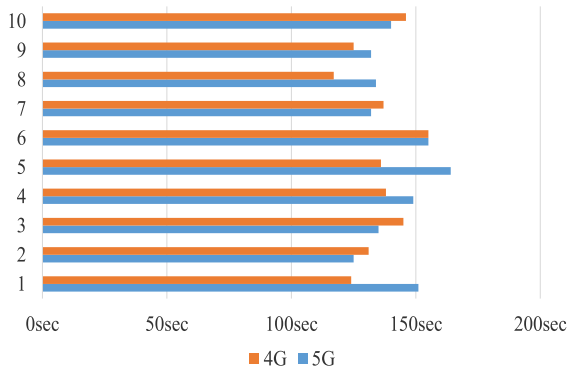


FIGURE 34. Comparison of map time in nodemanager.resource.memory.mb.

From Fig. 34, 35 and 36, we can see that more NodeManager memory doesn't always result in reduced map time, shuffle time or reduce time. For 5G, the time consumption is sometimes lower than 4G, but the difference is smaller. However, the time consumption of 5G is higher than that of 4G in a larger frequency and the difference is also

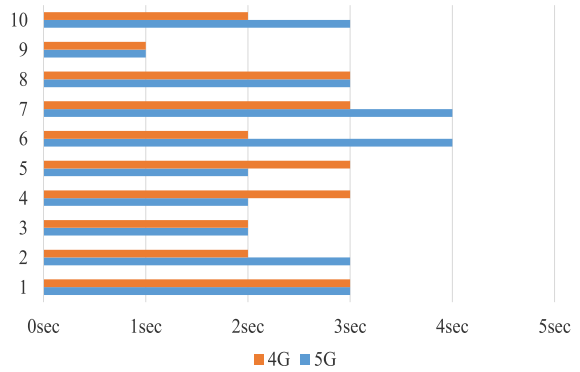


FIGURE 35. Comparison of shuffle time in nodemanager.resource.memory.mb.

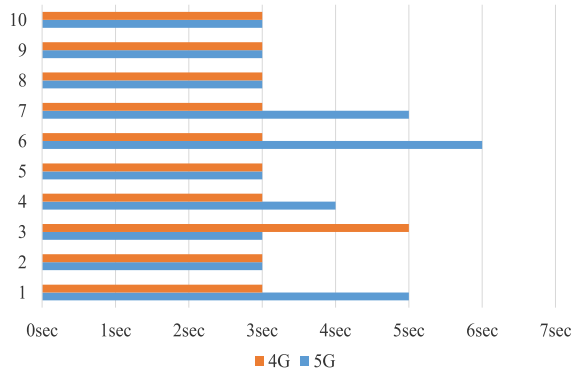


FIGURE 36. Comparison of reduce time in nodemanager.resource.memory.mb.

FIGURE 37. Comparison of map time in MultithreadedMapper.

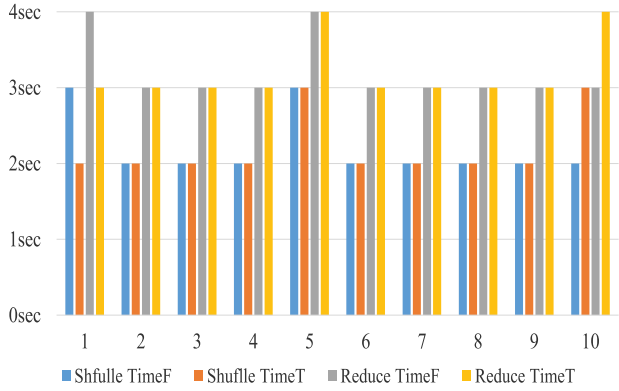


FIGURE 38. Comparison of shuffle and reduce time in MultithreadedMapper.

larger. Therefore, from the perspective of reducing resource consumption and running time of MapReduce tasks, the value of nodemanager.resource.memory.mb is selected as 4G. The hardware configuration of the three NodeManager nodes in this experiment is the same in Table 4, so it does not matter if they are uniformly set to 4G. In a real world application environment, each node in the NodeManager cluster may have a different configuration, so it is best to choose the configuration that best fits its memory for a single node.

7) MultithreadedMapper

The previous six attributes are all modified from the platform's parameter profile, which is modified from the programming code of MapReduce. This property indicates whether to enable multi-threaded execution of the map() method, which is ideal for computational jobs. In addition, the number of threads that can be opened at the same time can be specified in the code. After multiple repeated experiments, the maximum number of threads that can ensure the smooth execution of MapReduce program is 10 for the test data set of this experiment. The time consumption and prediction rate of Map, Shuffle, and Reduce are measured, and the results are as follows:

As can be seen from Fig. 37, before the multi-threading function of map is enabled, the consumption time is indeed longer than that after it is enabled. However, it can be seen from Fig. 38 and 39 that time consumption of shuffle and reduce basically does not change, and the prediction rate remains unchanged. Therefore, in order to adapt to computational tasks like the one tested in this experiment or tasks with more complex computational requirements, we choose to turn this property on.

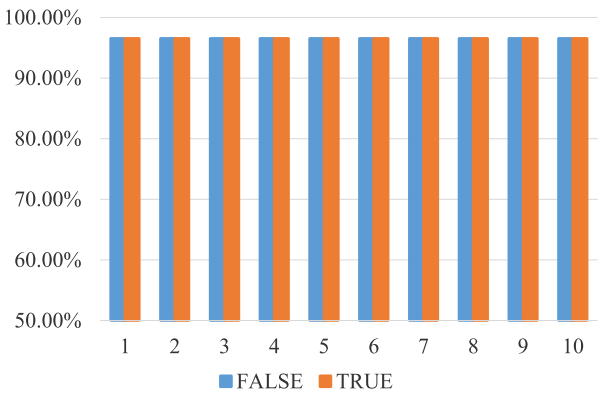


FIGURE 39. Comparison of prediction rate in MultithreadedMapper.

So far, in the platform parameter tuning, all the attributes that affect the algorithm prediction rate and the execution

time of MapReduce program involved in this experiment have been tuned. In the whole process of tuning, we can know that when the whole big data prediction algorithm is deployed in real world operating platform environments, the algorithm's writing logic and prediction rate effect is important, but platform's execution efficiency and time cost are also essential. We can adjust the hardware that runs the node, the memory capacity, and we can also adjust the code parameters of MapReduce and the data packaging methods. These optimization steps can not only improve the operation efficiency and prediction effect of big data platforms, but also enable us to better understand the underlying operation process and details of programs like Fig. 6 and 18, so as to better optimize platform parameters. After the parameter tuning, we need to do the prediction test again to observe how the prediction rate of the dataset and the time consumption of the MapReduc process will change.

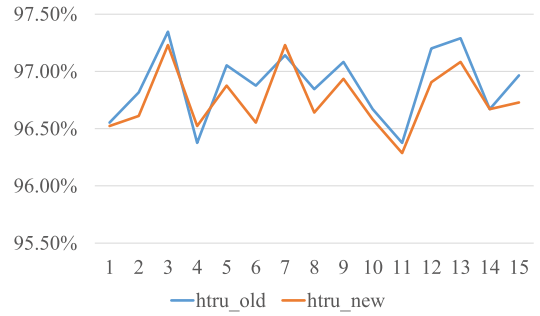


FIGURE 40. Comparison of prediction rate in different platform parameters.

As can be seen from Fig. 40, although the overall parameter optimization has an impact on the prediction rate, it is not significant. However, for individual prediction algorithms or tasks requiring strict accuracy, it is necessary to adjust platform parameters. And if in the more complex and real world big data operation environment, the requirement of accuracy will also be stricter. Therefore, we know that the optimization has little influence on the prediction rate.

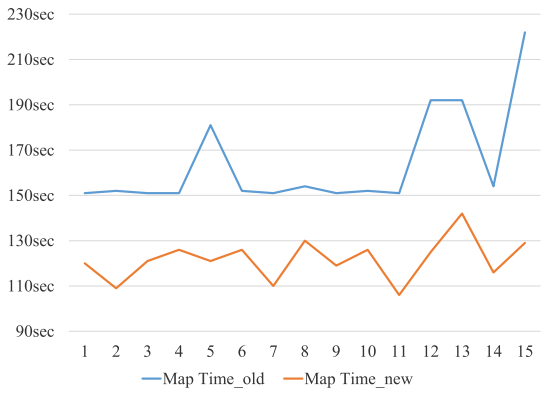


FIGURE 41. Comparison of map time in different platform parameters.

As can be seen from Fig. 41 and steps of attribute optimization in the above subsections, the most effective optimization is on map time. And in the whole running process, the most time-consuming portion is the map phase of MapReduce. The dataset involved in the experiment requires more computations. As such there are lots of calculation steps and time consumption on map side.

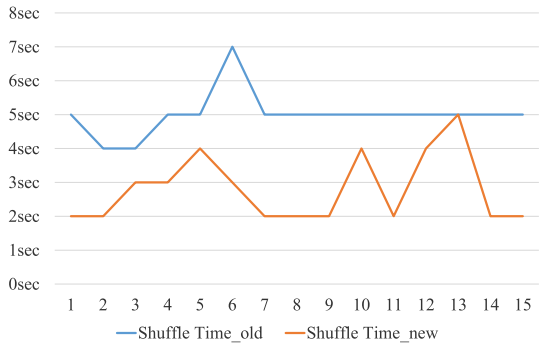


FIGURE 42. Comparison of shuffle time in different platform parameters.

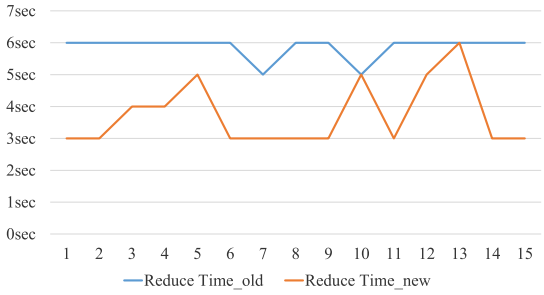


FIGURE 43. Comparison of reduce time in different platform parameters.

As can be seen from Fig. 42 and 43, for shuffle and reduce time, although effects after parameter optimization are not obvious, time consumptions are also reduced compared with that before optimization. Most of the running time of program tasks is spent on the map side. And after optimal optimization on it is completed, the following shuffle and reduce processes are automatically accelerated. Therefore, it can be seen from the above two figures that the shuffle and Reduce time of the optimized algorithm did decrease.

Fig. 44 shows that when parameters are optimized, spill time changes at a relatively stable level, although it is sometimes improved. On the contrary, the difference of spill time before optimization is sometimes large and unstable, which is mainly due to the optimization of Map side as shown in Fig. 6. When kvbuffer memory and the specified threshold reach a relatively reasonable value, the number and time of spill is correspondingly reduced (Fig. 23). In addition, after the map thread is added to perform tasks concurrently, the spill operation on Shuffle can be performed more quickly and smoothly, instead of waiting for a long time for task splitting on Map. Therefore, the spill time of the algorithm can be optimized after tuning.

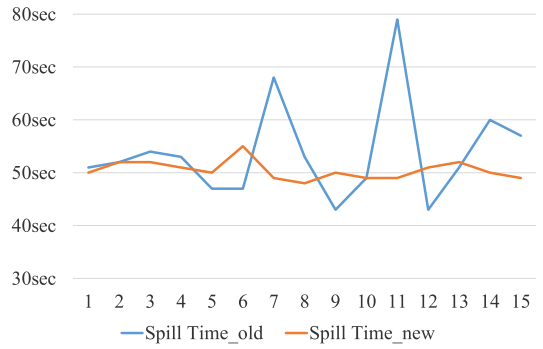


FIGURE 44. Comparison of spill time in different platform parameters.

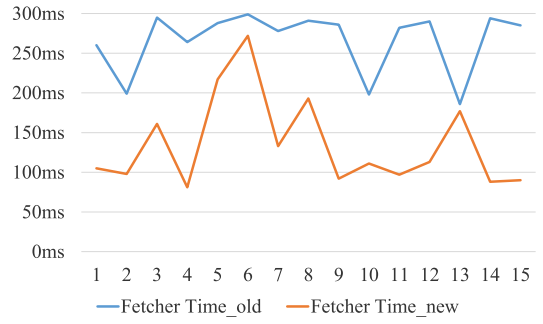


FIGURE 45. Comparison of fetcher time in different platform parameters.

Fig. 45 shows that after parameter optimization, the fetcher time consumption is also reduced compared to that before optimization. In addition to the optimization of kvbuffer and task memory at map, memory expansion at reduce and compression of output at shuffle have also been modified(Fig. 28, 29). Although datasets of this experiment have a large computational pressure on map end and small one on reduce end, these steps not only reduce the time consumption of the map itself, but also indirectly optimize the time consumption of the fetcher that finally reads data from the disk, thus reducing the pressure on Reduce end to receive data output from Map(Fig. 6). Therefore, the fetcher time is optimized after tuning.

VII. CONCLUSION

In this paper, we proposed a normalized distance measurement optimization based on Pearson, entropy weight method and KNN algorithm based on Gaussian function weighting optimization. And through experiments, we know that the more numbers of records and complex attributes of the dataset, the better prediction result this algorithm produces. In general, the result is better than the original KNN. In addition, the optimized algorithm is compared with other common classification algorithms, which proves that it is more suitable for the actual environment of large-scale data processing. Then we adjusted the parameters of Hadoop platform, combined with the process of MapReduce program, aiming to analyze the influence of different parameter adjustment on prediction rate and running time consumption. We found that for the dataset involved in this paper, most MapReduce time consumption would occur on Map, and parameter optimization did help reduce the time consumption of the program. Combined with the execution process of MapReduce, we further analyzed its relationship with parameter tuning, and concluded that platform parameter optimization has a small impact on the prediction rate, but can effectively reduce the running time of the program. Better parameter adjustment can reduce the time consumption for datasets with large number of records and should help to complete the prediction task better in real world operating environment.

Of course, future improvements should also be made from more perspectives. For example, compared with the original algorithm, the space complexity of optimized algorithm has not changed, and its space complexity has increased. This optimization improves the prediction accuracy by sacrificing spatial memory. Therefore, future research should look for some special data structures or new distance measures, which can reduce time and space complexity, simplify the flow and save memory space while maintaining high prediction rate. In terms of normalization and decentralization of KNN's algorithmic distance measurement, we can select or innovate a new measurement suitable for both large and small datasets. The new measurement should not be limited at the level of algorithm concept and design. It should reflect the logic and practical rigor of distance calculation. For the selection of k value, we can also use other training methods or other classification algorithms to predict in advance, in order to further analyze and make the final selection. For the optimization of weighted ranking, we can also select or innovate a more effective normal distribution function to replace the Gaussian function, so as to improve KNN. Future studies can apply the optimized algorithm to more types of datasets to discuss the effect of optimization under more complex conditions.

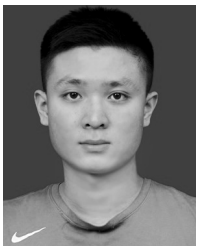
In the future, the execution process of big data platform and operation sequence of MapReduce can be further subdivided, and platform parameters can be optimized in more detail based on the underlying operation principle. Or we can divide the platform on which the algorithm program is running and compare the influence of different platforms on operation effects under the same configuration. After building hardware devices and datasets that are more in line with the scale of big data, we can better run and analyze the influence of platform parameter tuning on the algorithmic prediction process. Or different big data platforms can be applied to the experimental environment. In addition to studying the applicability of the algorithm to different platforms, we can also study the influence of different computing framework of platforms on prediction process, which can on prediction time or results.

REFERENCES

[1] Y. Nawal, M. Oussalah, B. Fergani, and A. Fleury, "New incremental SVM algorithms for human activity recognition in smart Homes," *J. Ambient Intell. Humanized Comput.*, pp. 1–18, Mar. 2022.

[2] M. Pramanik, R. Pradhan, P. Nandy, A. K. Bhoi, and P. Barsocchi, "The ForEx++ based decision tree ensemble approach for robust detection of Parkinson's disease," *J. Ambient Intell. Humanized Comput.*, pp. 1–25, Feb. 2022.
[3] R. Buijs, T. Koch, and E. Dugundji, "Using neural nets to predict transportation mode choice: Amsterdam network change analysis," *J. Ambient Intell. Humanized Comput.*, vol. 12, no. 1, pp. 121–135, Jan. 2021.
[4] G. Mohmed, A. Lotfi, and A. Pourabdollah, "Enhanced fuzzy finite state machine for human activity modelling and recognition," *J. Ambient Intell. Humanized Comput.*, vol. 11, no. 12, pp. 6077–6091, Dec. 2020.
[5] M. Das and S. K. Ghosh, "Reducing parameter value uncertainty in discrete Bayesian network learning: A semantic fuzzy Bayesian approach," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 5, no. 3, pp. 361–372, Jun. 2021.
[6] X. Zhao, J. Zhang, and X. Qin, "*k* NN-DP: Handling data skewness in *kNN* joins using MapReduce," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 3, pp. 600–613, Mar. 2018.
[7] M. O. Arowolo, M. O. Adebiyi, A. A. Adebiyi, and O. Olugbara, "Optimized hybrid investigative based dimensionality reduction methods for malaria vector using KNN classifier," *J. Big Data*, vol. 8, no. 1, pp. 1–14, Dec. 2021.
[8] F. Min, F.-L. Liu, L.-Y. Wen, and Z.-H. Zhang, "Tri-partition cost-sensitive active learning through kNN," *Soft Comput.*, vol. 23, no. 5, pp. 1557–1572, 2019.
[9] *UC Irvine Machine Learning Repository*. Accessed: Mar. 26, 2022. [Online]. Available: https://archive-beta.ics.uci.edu
[10] R. Fisher. (1988). *Iris*. UCI Machine Learning Repository. [Online]. Available: https://archive-beta.ics.uci.edu/ml/datasets/iris
[11] S. Wolberg and W. William. (1995). *Breast Cancer Wisconsin (Diagnostic)*. UCI Machine Learning Repository. [Online]. Available: https://archive-beta.ics.uci.edu/ml/datasets/breast+cancer+wisconsin
[12] (2020). *Dry Bean Dataset*. UCI Machine Learning Repository. [Online]. Available: https://archive-beta.ics.uci.edu/ml/datasets/dry+bean+dataset
[13] R. Lyon. (2017). *HTRU2*. UCI Machine Learning Repository. [Online]. Available: https://archive-beta.ics.uci.edu/ml/datasets/htru2
[14] R. J. Lyon, B. W. Stappers, S. Cooper, J. M. Brooke, and J. D. Knowles, "Fifty years of pulsar candidate selection: From simple filters to a new principled real-time classification approach," *Monthly Notices Roy. Astronomical Soc.*, vol. 459, no. 1, pp. 1104–1123, 2016.
[15] M. Koklu and I. A. Ozkan, "Multiclass classification of dry beans using computer vision and machine learning techniques," *Comput. Electron. Agricult.*, vol. 174, Jul. 2020, Art. no. 105507.
[16] S. Sharma and D. Toshniwal, "Scalable two-phase co-occurring sensitive pattern hiding using MapReduce," *J. Big Data*, vol. 4, no. 1, pp. 1–18, Dec. 2017.
[17] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.
[18] X. Liu, X. Wang, S. Matwin, and N. Japkowicz, "Meta-MapReduce for scalable data mining," *J. Big Data*, vol. 2, no. 1, pp. 1–21, Dec. 2015.
[19] V. Ricardo and D. Youssef, "A perspective view and survey of meta-learning," *Artif. Intell. Rev.*, vol. 18, pp. 77–95, Sep. 2001.
[20] J. Ye, J.-H. Chow, J. Chen, and Z. Zheng, "Stochastic gradient boosted distributed decision trees," in *Proc. 18th ACM Conf. Inf. Knowl. Manage. (CIKM)*, 2009, pp. 2061–2064.
[21] M. Weimer, S. Rao, and M. Zinkevich, "A convenient framework for efficient parallel multipass algorithms," in *Proc. LCCC, NIPS Workshop Learn. Cores, Clusters Clouds*, 2010, pp. 1–4.
[22] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, "Twister: A runtime for iterative MapReduce," in *Proc. 19th ACM Int. Symp. High Perform. Distrib. Comput. (HPDC)*, 2010, pp. 810–818.
[23] A. Agarwal, O. Chapelle, M. Dudík, and J. Langford, "A reliable effective terascale linear learning system," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1111–1133, 2014.
[24] R. D. Datasets, *A Fault-Tolerant Abstraction for in-Memory Cluster Computing*, vol. 12, M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, I. Stoica, Eds. Mumbai, India: NSDI, 2012.
[25] J. Rosen, N. Polyzotis, V. Borkar, Y. Bu, M. J. Carey, M. Weimer, T. Condie, and R. Ramakrishnan, "Iterative MapReduce for large scale machine learning," 2013, *arXiv:1303.3517*.
[26] M. M. Ghiasi and S. Zendehboudi, "Application of decision tree-based ensemble learning in the classification of breast cancer," *Comput. Biol. Med.*, vol. 128, Jan. 2021, Art. no. 104089.
[27] *Amazon EC2*. Accessed: Jan. 1, 2018. [Online]. Available: https://aws.amazon.com/ec2
[28] *Amazon EC2 Spot Instances*. Accessed: Jan. 1, 2018. [Online]. Available: https://aws.amazon.com/ec2/spot
[29] *Amazon EC2 Dashboard*. Accessed: Jan. 1, 2018. [Online]. Available: https://console.aws.amazon.com/ec2
[30] *Boto3 Documentation*. Accessed: Jan. 1, 2018. [Online]. Available: https://boto3.readthedocs.io
[31] *AWS Command Line Interface Documentation*. Accessed: Jan. 1, 2018. [Online]. Available: https://aws.amazon.com/documentation/cli
[32] *Apache Spark*. Accessed: Mar. 26, 2022. [Online]. Available: http://spark.apache.org
[33] *DataMPI*. Accessed: Mar. 21, 2022. [Online]. Available: http://datampi.org
[34] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th Symp. Operating Syst. Design Implement. (OSDI)*. San Francisco, CA, USA: USENIX Association, Dec. 2004, pp. 1–7.
[35] J. Liu, S. Tang, G. Xu, C. Ma, and M. Lin, "A novel configuration tuning method based on feature selection for Hadoop MapReduce," *IEEE Access*, vol. 8, pp. 63862–63871, 2020.
[36] *Release Notes HDP-2.1.1*. Accessed: Mar. 19, 2022. [Online]. Available: https://docs.cloudera.com/HDPDocuments/HDP2/HDP-2.1.5/bk_releasenotes_hdp_2.1/content/ch_relnotes-hdpch_relnotes-hdp-2.1.1-knownissues-mapreduce.html
[37] S. Maldonado, E. Carrizosa, and R. Weber, "Kernel penalized K-means: A feature selection method based on kernel K-means," *Inf. Sci.*, vol. 322, pp. 150–160, Nov. 2015.
[38] H. Dudeja and C. Modi, "Runtime program semantics based malware detection in virtual machines of cloud computing," in *Proc. Int. Conf. Inf. Process.*, Cham, Switzerland, 2021, pp. 3–16.
[39] M. Ali, S. I. Ali, D. Kim, T. Hur, J. Bang, S. Lee, B. H. Kang, and M. Hussain, "UEFS: An efficient and comprehensive ensemble-based feature selection methodology to select informative features," *PLoS ONE*, vol. 13, no. 8, Aug. 2018, Art. no. e0202705.
[40] J. Gómez-Ramírez, M. Ávila-Villanueva, and M. Á. Fernández-Blázquez, "Selecting the most important self-assessed features for predicting conversion to mild cognitive impairment with random forest and permutation-based methods," *Sci. Rep.*, vol. 10, no. 1, pp. 1–15, Dec. 2020.
[41] D. M. Atallah, M. Badawy, A. El-Sayed, and M. A. Ghoneim, "Predicting kidney transplantation outcome based on hybrid feature selection and KNN classifier," *Multimedia Tools Appl.*, vol. 78, no. 14, pp. 20383–20407, Jul. 2019.
[42] A. O. Ojo, J. A. Hanson, H.-U. Meier-Kriesche, C. N. Okechukwu, R. A. Wolfe, A. B. Leichtman, L. Y. Agodoa, B. Kaplan, and F. K. Port, "Survival in recipients of marginal cadaveric donor kidneys compared with other recipients and wait-listed transplant candidates," *J. Amer. Soc. Nephrol.*, vol. 12, no. 3, pp. 589–597, Mar. 2001.
[43] A. O. Ojo, R. A. Wolfe, L. Y. Agodoa, P. J. Held, F. K. Port, S. F. Leavey, S. E. Callard, D. M. Dickinson, R. L. Schmouder, and A. B. Leichtman, "Prognosis after primary renal transplant failure and the beneficial effects of repeat transplantation: Multivariate analyses from the United States renal data system," *Transplantation*, vol. 66, no. 12, pp. 1651–1659, Dec. 1998.
[44] A. Akl, A. Mostafa, and M. A. Ghoneim, "Nomogram that predicts graft survival probability following living-donor kidney transplant," *Exp. Clin. Transplantation*, vol. 6, no. 1, pp. 30–36, 2008.
[45] T. S. Brown, E. A. Elster, K. Stevens, J. C. Graybill, S. Gillern, S. Phinney, M. O. Salifu, and R. M. Jindal, "Bayesian modeling of pretransplant variables accurately predicts kidney graft survival," *Amer. J. Nephrol.*, vol. 36, no. 6, pp. 561–569, 2012.
[46] S. Krikov, A. Khan, B. C. Baird, L. L. Barenbaum, A. Leviatov, J. K. Koford, and A. S. Goldfarb-Rumyantzev, "Predicting kidney transplant survival using tree-based modeling," *ASAIO J.*, vol. 53, no. 5, pp. 592–600, 2007.
[47] R. S. Lin, S. D. Horn, J. F. Hurdle, and A. S. Goldfarb-Rumyantzev, "Single and multiple time-point prediction models in kidney transplant outcomes," *J. Biomed. Informat.*, vol. 41, no. 6, pp. 944–952, Dec. 2008.
[48] K. Topuz, F. D. Zengul, A. Dag, A. Almehmi, and M. B. Yildirim, "Predicting graft survival among kidney transplant recipients: A Bayesian decision support model," *Decis. Support Syst.*, vol. 106, pp. 97–109, Feb. 2018.
[49] Y. Zhang, T. Cao, S. Li, X. Tian, L. Yuan, H. Jia, and A. V. Vasilakos, "Parallel processing systems for big data: A survey," *Proc. IEEE*, vol. 104, no. 11, pp. 2114–2136, Nov. 2016.

[50] N. Maleki, A. M. Rahmani, and M. Conti, "MapReduce: An infrastructure review and research insights," *J. Supercomput.*, vol. 75, no. 10, pp. 6934–7002, 2019.
[51] M. Soualhia, F. Khomh, and S. Tahar, "Task scheduling in big data platforms: A systematic literature review," *J. Syst. Softw.*, vol. 134, pp. 170–189, Dec. 2017.
[52] B. Zhang, X. Wang, and Z. Zheng, "The optimization for recurring queries in big data analysis system with MapReduce," *Future Gener. Comput. Syst.*, vol. 87, pp. 549–556, Oct. 2018.
[53] R. Li, H. Hu, H. Li, Y. Wu, and J. Yang, "MapReduce parallel programming model: A state-of-the-art survey," *Int. J. Parallel Program.*, vol. 44, no. 4, pp. 832–866, Aug. 2016.
[54] K. Matsuzaki, "Functional models of Hadoop MapReduce with application to scan," *Int. J. Parallel Program.*, vol. 45, no. 2, pp. 362–381, Apr. 2017.
[55] W. Tian, G. Li, W. Yang, and R. Buyya, "HScheduler: An optimal approach to minimize the makespan of multiple MapReduce jobs," *J. Supercomput.*, vol. 72, no. 6, pp. 2376–2393, Jun. 2016.
[56] N. Ahmed, A. L. C. Barczak, T. Susnjak, and M. A. Rashid, "A comprehensive performance analysis of apache Hadoop and apache spark for large scale data sets using HiBench," *J. Big Data*, vol. 7, no. 1, pp. 1–18, Dec. 2020.
[57] *Known Issues for MapReduce*. Accessed: Mar. 27, 2022. [Online]. Available: https://docs.cloudera.com/HDPDocuments/HDP2/HDP-2.1.5/bk_releasenotes_hdp_2.1/content/ch_relnotes-hdpch_relnotes-hdp-2.1.1-knownissues-mapreduce.html
[58] Z. Huang and D. Chen, "A breast cancer diagnosis method based on VIM feature selection and hierarchical clustering random forest algorithm," *IEEE Access*, vol. 10, pp. 3284–3293, 2022.
[59] M. M. Ghiasi and S. Zendehboudi, "Application of decision tree-based ensemble learning in the classification of breast cancer," *Comput. Biol. Med.*, vol. 128, Jan. 2021, Art. no. 104089.



CHEN MA received the Bachelor of Engineering degree from Shanxi University, Taiyuan, China, in 2021. He is currently pursuing the master's degree in computer technology with Xijing University, Xi'an, Shaanxi. His research interests include big data analysis and software engineering.

YUHONG CHI received the master's degree in computer applied engineering from Northeastern University, Liaoning, China, in 2005, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2013. Her research interests include computational intelligence and its applications and data analysis.