Hindawi
Mathematical Problems in Engineering
Volume 2020, Article ID 7898230, 14 pages
https://doi.org/10.1155/2020/7898230

Research Article

Partition Selection for Large-Scale Data Management Using KNN Join Processing

Yue Hu,[1,2] **Ge Peng**,[1] **Zehua Wang**,[1] **Yanrong Cui**,[1,2] **and Hang Qin**[1,2]

[1]*Computer School, Yangtze University, Hubei 434023, China*
[2]*Hubei Graduate Workstation with Jingpeng Software Group Co., Ltd., Jingzhou, Hubei, China*

Correspondence should be addressed to Hang Qin; 68681700@qq.com

Received 29 May 2020; Accepted 10 August 2020; Published 8 September 2020

Guest Editor: Liangtian Wan

For the data processing with increasing avalanche under large datasets, the k nearest neighbors (KNN) algorithm is a particularly expensive operation for both classification and regression predictive problems. To predict the values of new data points, it can calculate the feature similarity between each object in the test dataset and each object in the training dataset. However, due to expensive computational cost, the single computer is out of work to deal with large-scale dataset. In this paper, we propose an adaptive vKNN algorithm, which adopts on the Voronoi diagram under the MapReduce parallel framework and makes full use of the advantages of parallel computing in processing large-scale data. In the process of partition selection, we design a new predictive strategy for sample point to find the optimal relevant partition. Then, we can effectively collect irrelevant data, reduce KNN join computation, and improve the operation efficiency. Finally, we use a large number of 54-dimensional datasets to conduct a large number of experiments on the cluster. The experimental results show that our proposed method is effective and scalable with ensuring accuracy.

1. Introduction

In recent years, with the wide deployment of cloud computing, network, and radio, Internet of Things (IOT) products have been widely used in the natural sciences, such as industrial pollution areas [1], mobile devices [2], vehicle communication systems [3], and radar Systems [4]. In the face of complex and diverse signals, the traditional radars have limited performance in DOA estimating [5]. How to enhance useful signals and extract useful information is the focus of current research [6]. With the development of signal processing technology, researchers propose a variety of algorithms to achieve this purpose. Typical algorithms include the multiple signal classification (MUSIC) and its variations [3, 7], ULA-based method [8], reduced-complexity OGSBL [9], PARAFAC decomposition [10], Tensor-based subspace algorithm [11], and NNM [12].

To date, searching for approximate objects from the vast amount of useful data is a very basic and critical operation. With the continuous expansion of network scale, the data scale presents explosive growth with large-volume, complex, and growing datasets for multiple and autonomous sources. As a result, the k nearest neighbors (KNN) algorithm with high accuracy, insensitive to outliers, and no data input assumptions represents an important paradigm shift in the evolution of partition selection with the trained metric. Existing KNN algorithm-based method assume the classification can be improved by learning a distance metric from labeled examples.

In the classification process, the KNN algorithm calculates the distance (similarity) between the data sample to be classified and the entire known dataset. This method is simple, convenient, and inexpensive for small datasets. When dealing with such large-scale data, the complexity of similarity calculation increases dramatically with the intolerable calculation cost, which directly affects the classification efficiency and accuracy. When the dataset has more attributes, the impact is more evident, and the dimension catastrophes tend to occur to make distances very far when calculated in high-dimensional space. At this point, a natural idea is to introduce the idea of distribution with implementation of data parallel support.

In particular, MapReduce is a distributed parallel programming framework proposed by Google for processing large datasets on large-scale clusters, in terms of a streamlined computational framework to assemble sequential and parallel computation. The programming model via MapReduce was originally designed to simplify large-scale data calculations [13, 14]. In recent years, MapReduce has been studied in many ways [15–18] for large-scale data-intensive computing under data-intensive [19], CPU-intensive, and memory-intensive applications, such as in the fields of smart cities [20], biological data management [21], spatial geometry calculation [22], and distributed computing over a wireless interference network [23].

The goal of this paper is to propose an effective data partitioning strategy. The proposed KNN algorithm designed based on MapReduce framework is mainly used to solve the problem of too much computation and low classification efficiency. With this programming framework, we can divide KNN's computing tasks into several small tasks and assign them to several computing nodes to calculate at the same time for the speedup the operation. Only by dividing the data reasonably and actually reducing the calculation cost practically can the running efficiency of the algorithm be effectively improved.

The contributions of this paper are as follows:

(1) We introduce the idea of the Voronoi diagram to partition the sample objects and design the partition selection strategy to find the optimal relevant partition for the sample to be tested, thereby avoiding the extracalculation brought by irrelevant data.

(2) We address the MapReduce framework and propose a vKNN algorithm, which is implemented on the Hadoop cluster with KNN join processing, nearest center points selecting, relevant-partition selecting, and vKNN processing.

(3) We conduct many experiments using real datasets to study the effects of various parameters on the algorithm. The results show that our proposed algorithm is effective and scalable with the accuracy in relevant-partition selection.

The contents of the paper are structured as follows. Section 2 reviews the related work, Section 3 formally defines the problems to be solved in this paper, Section 4 describes the related technologies involved in this paper, Section 5 details the implementation and improvement of the KNN algorithm based on MapReduce, Section 6 reports the experimental results, and finally Section 7 summarizes the entire study.

2. Related Work

Existing KNN method assumed that the classification algorithm can be widely applied in the field of machine learning and large-scale data analysis. In order to better apply the traditional KNN algorithm, previous studies mainly used two kinds of methods, *i.e.*, speeding up the process of finding k nearest neighbors and eliminating irrelevant data to reduce the overall computation. For instance, Cui et al. [24] introduced a B^+-tree method that maps high-dimensional data points in one dimension. The one-dimensional distance computed in the principal component space and the first principal components of the sample points were indexed using a B^+-tree. At the same time, the principal components were adopted to filter neighboring query points to improve query efficiency. When working with high-dimensional data, most indexing methods cannot scale up well and perform worse than sequential scanning. Xia et al. [25] designed and implemented KNN join algorithm based on block nested join Gorder using a grid-based sorting method, which can effectively assign similar objects to the same grid. Amagata et al. [26] proposed a dynamic set KNN self-join algorithm to trim unnecessary computations using index technology.

However, the processing power of the single processor greatly limits the development of the KNN algorithm, which also makes the application of parallel and distributed compute imperatively. In recent years, MapReduce has been fully practiced in the field of machine learning [27–29]. In order to solve those problems of KNN algorithm, Zhang et al. [30] proposed the HBNJ algorithm implemented by Hadoop and its improved algorithm H-BRJ in document. However, due to the large impact of data size on the efficiency of the algorithm, this research focused more on approximate queries. Moutafis et al. [31] proposed a four-stage algorithm, where three optimization strategies were used to trim distant points, balance the number of reducers, and halve the output, which significantly reduced the computation time. In most studies, people chose to use the first data partitioning to reduce data calculation, such as R-tree, Δ-tree, Quad-Tree, and KDB-Tree [32]. These spatial partitioning-based indexing techniques will dramatically reduce efficiency as dimensions increase. Zhang et al. [30] proposed a Z-value-based partitioning strategy. The result of the algorithm depends to a large extent on the quality of the z-curve, which may cause problems in the processing of high-dimensional data. Ji et al. [33] proposed a distance-based partitioning method. However, this grid-based partitioning method is considered valid only for low-dimensional datasets. We also use a partitioning strategy. In this paper, we introduce the concept of Voronoi diagram because it can be applied to any dimension of data [34]. We use Voronoi diagrams to aggregate similar data so that irrelevant data can be clipped. The Voronoi diagram was proposed with the famous structure of computational geometry. It is widely used in many fields such as geometry, architecture, and geography [35–38]. Voronoi diagrams can partition data into set spaces and are effective in the study of local neighborhoods for each partition [39]. At the same time, Voronoi diagrams can help improve the performance of distance join queries [40].

3. Problem Formulation

In this part, we give the definition of KNN Join with its formulation. Table 1 lists some symbols and their corresponding meanings involved mainly in this paper.

Table 1: Symbols and their meanings.

Symbol	Definition
$R(S)$	The d-dimensional dataset
$r(s)$	The data object in $R(S)$
$\mathfrak{R}^d$	The d-dimensional metric space
$\mathrm{dis}(r, s)$	The distance from r to s
k	The number of nearest neighbors
$\mathrm{knn}(s, R, k)$	The k nearest neighbors of s from R
$\mathrm{knnJ}(R, S)$	The KNN Join of R and S
$\max(\mathrm{knn}(s, R, k))$	The maximum distance from s to its k nearest neighbors from R
N	The number of mappers
N	The number of center points
$\mathscr{P}$	The set of center points
p_i	The point in $\mathscr{P}$
P_i	The partition corresponds to p_i
P^x	The partition where x is located
P_i^s	The partition corresponding to the ith center point close to s

Let R and S be two different d-dimensional datasets, and $r(s)$ is the data object in $R(S)$. For the convenience of discussion, we introduce the geometric space to represent them. $\mathfrak{R}^d$ is a d-dimensional geometric space, R and S can be regarded as a sample point set in $\mathfrak{R}^d$, and the data objects r and s can be viewed as a d-dimensional sample point; then, we have $r \in R\,(s \in S)$. In order to avoid loss of generality, the distance measurement method adopted in this paper is Euclidean distance. Also, the distance between the data objects r and s, denoted as $\mathrm{dis}(r, s)$, can be calculated as follows:

$$\mathrm{dis}\,(r, s) = \left(\sum_{l=1}^{d} \left| r^l - s^l \right|^2 \right)^{1/2}, \tag{1}$$

where $\mathrm{dis}(r, s) \geq 0$ and the necessary condition for $\mathrm{dis}(r, s) = 0$ is $r = s$.

The similarity between data objects r and s, denoted as $\mathrm{sim}(r, s)$, is

$$\mathrm{sim}\,(r, s) = \frac{1}{1 + \mathrm{dis}\,(r, s)}, \tag{2}$$

where the greater the distance, the greater the difference between objects and the smaller the similarity.

Definition 1 (KNN). Given a sample set R, a newly input sample point s. The KNN operation of them, denoted as $\mathrm{knn}\,(s, R, k)$, involves the k nearest neighbors of s from R. The formal description is as follows:

$$\mathrm{knn}\,(s, R, k) = \left\{ z_1, z_2, \ldots, z_k \,\middle|\, z_1, z_2, \ldots, z_k \in R \right\}, \tag{3}$$

for $\forall z_j \in R - \{z_1, z_2, \ldots, z_k \mid z_1, z_2, \ldots, z_k \in R\}$, and we have

$$d\left(s, z_j\right) \geq d\left(s, z_k\right) \geq \cdots \geq d\left(s, z_2\right) \geq d\left(s, z_1\right). \tag{4}$$

Definition 2 (KNN Join). Given two sample sets R and S. The KNN join operation denoted as $\mathrm{knnJ}\,(R, S)$ returns each object $s \in S$ with its k nearest neighbors from R. The formal description is as follows:

$$\mathrm{knnJ}\,(R, S) = \{s, \mathrm{knn}\,(s, R, k) \mid \text{for all } s \in S\}. \tag{5}$$

4. MapReduce for Data Processing under Voronoi-Based KNN Processing

4.1. MapReduce for Flexible Data Processing. MapReduce, a distributed parallel programming framework, is a member of the core designs of Hadoop [41, 42]. It separates the users from the bottom layer of the system. When users write the corresponding programs, they only need to write the *Map* function and *Reduce* function to give what needs to be calculated and how to calculate automatically by the framework. Meanwhile, MapReduce has been extensively used due to its high fault tolerance and scalability.

MapReduce is mainly used for parallel computation of large amounts of data. A MapReduce program contains only two functions: *Map* function and *Reduce* function. The corresponding processing of these two functions can be customized by the user. At the beginning of the calculation task, Hadoop divides the entire job into two sequential phases: *Map* phase and *Reduce* phase. The model first breaks down the computational tasks of large-scale data that need to be processed into many individual tasks. These individual tasks can be run in parallel on a Hadoop server cluster; then, the model combines the results calculated by the cluster and calculates the final result. In the *Map*/*Reduce* phase, there are multiple instance tasks, which can be executed in parallel on each node. The MapReduce programs save both input and output results in HDFS. They use migration to transfer data to the nearest available node only if the node does not have local data or cannot process local data [43].

Figure 1 shows the specific execution process of MapReduce. The *Client* program divides the file data according to the parameter (m). The *ResourceManager* picks the idle nodes in the cluster and assigns the corresponding compute resources to them. At the same time, the *Job* is assigned the same number of *Map* tasks with the number of file blocks.

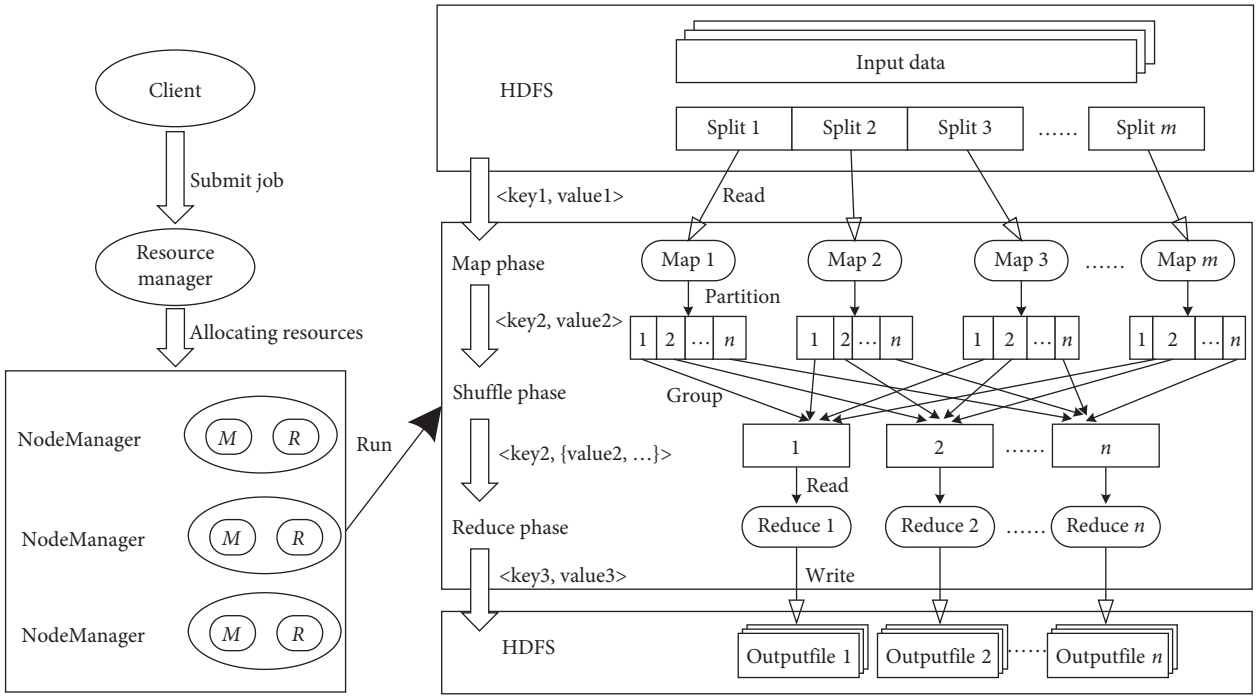

Figure 1: The workflow of the MapReduce.

Before starting the *Map* task, the file on the node is read and parsed into <*key*1, *value*1> key-value pairs by line, and then the *Map* function converts it into new <key2, Value2> key-value pairs. Next, the model uses the *Hash* function to partition and sort <*key*2, *value*2> key-value pairs, and groups them to <*key*2, {*value*2, . . .}> according to the *key*2. Finally, the *Reduce* function accepts the data, generates new <*key*3, *value*3> pairs from the corresponding business logic, and saves it in HDFS. When the cluster resources are not sufficient to host all *Map* (*Reduce*) tasks at the same time, the corresponding tasks are started in batches. In addition, the first *Reduce* task can only be started after the last *Map* task has been executed [44].

4.2. Voronoi Diagram with Partition Selection. The Voronoi diagram, also known as Dirichlet diagram, plays an important role in computational geometry. In the field of mathematics, this diagram is a decomposition of a given space, the simplest form to decompose a plane. The division yields that all points in each area are closer to the center of the area than to other centers.

To illustrate this, we can take a two-dimensional plane as an example. Given a dataset R, each object in R can be regarded as a point of $\mathfrak{R}^2$. Partitioning using a Voronoi diagram, means selecting n objects as the center points and assigning all objects in R to the partition corresponding to their nearest center points. This divides the entire data space into n partitions, as shown in Figure 2. The large blue circle represents the center points of the Voronoi diagram, and each point represents an object. The green points represent the k nearest neighbors of

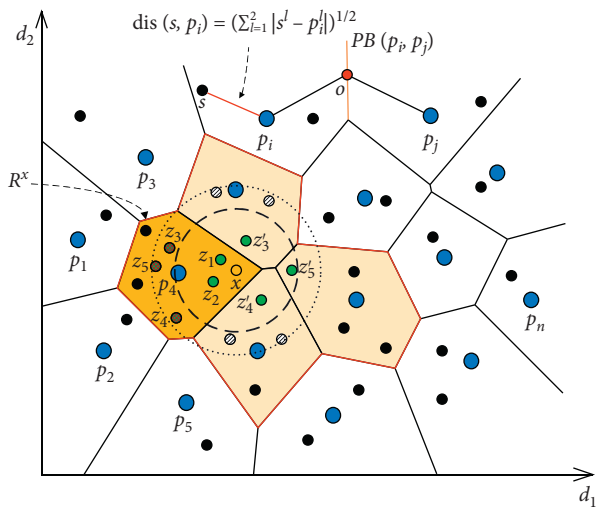

Figure 2: Voronoi diagram of the plane.

object x, where the grid points represent the false k nearest neighbors. The orange area is where the area of object x is located, and the region enclosed by the red line is the set of regions, where the true k nearest neighbors of object x are located. The dashed areas correspond to the spatial ranges of the true k nearest neighbors and the false k nearest neighbors, respectively.

For the sake of brevity, let $\mathcal{P}$ be the selected set of center points, where $\mathcal{P} = \{p_1, \ldots, p_n\}$. Given two center points p_i and p_j, $\mathrm{PB}(p_i, p_j)$ represents the hyperplane dividing the partitions, where p_i and p_j are located for the point o on the hyperplane $PB(p_i, p_j)$, and we have

$$\forall o \in PB\left(p_i, p_j\right), \quad \operatorname{dis}(o, p_i) = \operatorname{dis}\left(o, p_j\right). \tag{6}$$

According to formula (1), for any sample points s, located in the corresponding subspace of p_i, the distance between s and p_i, denoted as $\operatorname{dis}(s, p_i)$, is

$$\operatorname{dis}(s, p_i) = \left(\sum_{l=1}^{d} \left|s^l - p_i^l\right|^2\right)^{1/2}, \tag{7}$$

for $\forall p_j \in \mathcal{P} - \{p_i\}$, and we have

$$\operatorname{dis}(s, p_i) < \operatorname{dis}\left(s, p_j\right). \tag{8}$$

The distance from s to $PB(p_i, p_j)$, denoted as $\operatorname{dis}(s, PB(p_i, p_j))$, can be calculated as follows:

$$\operatorname{dis}\left(s, PB\left(p_i, p_j\right)\right) = \frac{\left(\operatorname{dis}\left(s, p_j\right)\right)^2 - \left(\operatorname{dis}(s, p_i)\right)^2}{2 \times \operatorname{dis}\left(p_i, p_j\right)}. \tag{9}$$

Figure 3 shows the distance $\operatorname{dis}(s, PB(p_i, p_j))$. Based on the characteristics of the Voronoi diagram, we can transform the process of finding k nearest neighbors in R in Definition 1 into the process of finding k nearest neighbors in the partition. Now, our work only considers the partition where the sample point is located.

Hypothesis 1 Given a sample set R, perform a KNN operation on the newly input sample point x, that is, find k nearest sample points of x in the partition p^x where x is located. The formal description is as follows:

$$\operatorname{knn}(x, R, k) = \operatorname{knn}\left(x, p^x, k\right). \tag{10}$$

In this case, the computational effort of the running process is significantly reduced. However, when x is near the p^x boundary of the partition where it is located, it is easy to produce large errors by roughly limiting the size of the partition involved in the calculation. Suppose $k = 5$, as shown in Figure 2, the sample point x to be measured is located in the partition corresponding to p_4. According to Hypothesis 1, the k nearest sample points are $\{z_1, z_2, \ldots, z_5\}$. And when we look through Figure 2, it is easy to find that the real k nearest sample points are $\{z_1, z_2, z_3', z_4', z_5'\}$, whereas the sample points $\{z_3', z_4', z_5'\}$ outside of the partition. This means that, in the actual operation process, only the samples in the partition where x is located may not necessarily yield true results. Consequently, we introduce a new concept: relevant partition, which is given in conjunction with Definition 1 as follows.

Definition 3. (relevant partition). Given the sample set R, the corresponding relevant partition R^x for the newly input sample point x is the partition set of k nearest neighbors. We have

$$R^x = p^{z_1} \cup p^{z_2} \cup \cdots \cup p^{z_k}. \tag{11}$$

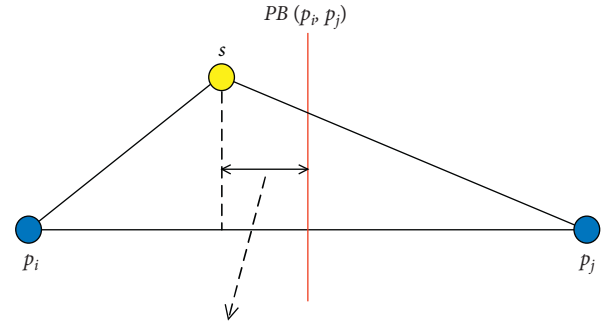

Figure 3: The distance from (s) to $PB(p_i, p_j)$.

Therefore, KNN operation on x and R can be converted to finding the k nearest neighbors of x in the relevant partition R^x, and the formal description is as follows:

$$\operatorname{knn}(x, R, k) = \operatorname{knn}\left(x, R^x, k\right). \tag{12}$$

5. KNN Algorithm with MapReduce Performance Improvement

5.1. KNN Join Processing. The basic idea of the KNN join algorithm based on MapReduce is in general agreement with that of the KNN algorithm. Firstly, the MapReduce program divides the input test dataset, each node calculates the distance between the test samples in the corresponding slice and each sample in the training dataset, finds out the k nearest neighbors, and selects the label with the largest proportion of these adjacent points. As shown in Algorithm 1, the KNN join processing can be formulated by the following.

In the *Map* function, it first sets the parameter k of the algorithm. Next, it calculates the Euclidean distance between each sample r of the training dataset R and the test sample s and stores the labels of the k nearest training samples into trainLabel. The form of the input data <*key*, *value*> is <*row number*, *sample*>; the form of the output data <*key*, *values*> is <*sample*, *the label of adjacent sample*>. The *Hash* function partitions, sorts, and groups these intermediate results by key values. The *Reduce* function then reads them. The design of the *Reduce* function is relatively simple. Its main task is to obtain the label with the maximum number of k labels and assign it to the test sample. First, iterate through the data passed by the *Map* function in turn, and if the current data exists in the *HashMap*, add 1 to its value. If the current data does not exist in the *HashMap*, mark its value as 1, and add it to the *HashMap*. Finally, the label with the largest value in *HashMap* is used as the prediction label. The form of input data <*key*, *values*> is <*sample*, *set* (*the labels of adjacent k samples*)>. The form of output data <*key*, *value*> is <*sample*, *the prediction label*>.

Obviously, the method is too expensive. It simply assigns computing tasks to the computing nodes, and each mapper needs to connect a subset from S to the entire dataset R. Considering only the distance calculation to be performed

```
Input: k, R, S
Output: s.predictLabel//prediction label of s
map: <row number, s>
  foreach r ∈ R do
    dis = dis(r, s);//calculate the Euclidean distance between r and s
    for i = 0 to k do
      if dis < distance[i] then//find the minimum k distances
        distance[i] = dis;
        trainLabel[i] = r.label;
        break;
  for j = 0 to k do
    output(s, trainLabel[i]);
reduce: <s, Labels>
  hmp = new HashMap(); //create a HashMap object hmp
  foreach label ∈ Labels do//count the number of each label
    if hmp.get(label) ! = NULL then//if the label exists in hmp
      label.value ++; //take the value of the label and add 1
      hmp.put(label) = label.value;//update the value of the label in hmp
    else//if the label does not exist in hmp
      hmp.put(label) = 1; //set the value of the label to 1 and insert to hmp
  predictLabel = hmp.maxvalue; //the label with the largest value as the prediction label
  output(s, predictLabel);
```

Algorithm 1: KNN join processing.

for each node, the amount of computation reaches $|R| \cdot (|S|/n)$. When the dataset involved in the calculation is large, the amount of computation is still a huge amount in comparison. In addition, the comparison of similarity may exceed the computing capacity of the node, resulting in the task being killed. Therefore, it is unreliable to rely solely on MapReduce to slice computational tasks for efficient classification. A better idea is to reduce the number of samples in R that is involved in the calculation.

5.2. Relevant-Partition Selecting and vKNN Processing. We consider the Voronoi diagrams for KNN join within the MapReduce framework. The basic idea is to partition the data using the Voronoi diagram and clip the unqualified data to reduce the amount of calculation. There are three main steps as follows.

5.2.1. Preprocessing Step. Input dataset R and partition R using Voronoi diagram. First, randomly select N samples from the dataset R as the initial center point. Then, we use k-means clustering method to analyze the dataset globally to obtain the center points set $\mathcal{P}$ and the corresponding data clusters.

5.2.2. Nearest Center Point Selecting Step. In this step, we use the output of the above processes and dataset S as input objects. Find the k nearest center points of each sample in S and save its index and distance information to help clip the unqualified data.

Algorithm 2 shows the execution of the mappers at this stage. Before the program starts, we can load the preprocess center point data into the main memory of each mapper. After each mapper reads the sample object s, it traverses each center point and calculates the distance of s from all the center points. In order to reduce the cost of data transmission between nodes, we only save the index and distance information of the k center points nearest to s. We use *TreeMap* to store relevant information so that we can get the first k center points more quickly, where *TreeMap* itself is an ordered set of key values. All elements remain in a specific order and are sorted in ascending order by default by the value of the key. So, we can easily get information about the nearest k center points.

5.2.3. vKNN Processing Step. In this step, we use the output from the previous two processes as input. On the basis of the distance, we can filter out the relevant partition R^s corresponding to each sample s in S to find out the labels of the k nearest neighbors. Finally, reducer counts the labels and outputs the label that appears most often.

In order to ensure the accuracy of the prediction results, the selected relevant partition R^s contains all knn(s, R, k) as minimum as possible. How to determine the relevant partition R^s that will participate in the final calculation is an issue we need to consider now.

Theorem 1. *Given a sample point s and a center point p_i, s is located in the partition p_i corresponding to p_i, and we have*

$$p_i \in R^s. \tag{13}$$

Proof. As s is located in the partition p_i, s is very similar to the sample points in p_i. This means that the probability of having the nearest neighbors in p_i is greater than that in any

Input: S, $\mathcal{P}$
Output: <s, $\{P_1^s, P_2^s, \ldots, P_k^s\}$>//the partitions corresponding to the k nearest center points of s
foreach $s \in S$ **do**
  tmp = TreeMap(); //create a TreeMap object tmp
  nearPointSet = [ ]; //create an empty set nearPointSet
  foreach $p \in \mathcal{P}$ **do**//insert the distance and index information tmp.put(dis(p, s), $p.index$);
  for $i = 0$ **to** k **do**//read the first k center points' information nearPointSet.append(tmp.next()); //put the information to nearPointSet

Algorithm 2: Nearest center point selecting.

other partition. In other words, p_i are more likely to have the nearest neighbors.

Based on the analysis of Hypothesis 1, we may have errors in predicting using only the sample points within p_i. We can design a selection strategy to determine whether other partitions meet the criteria.

Definition 4. Given two center points p_i and p_j, $\mathrm{PB}(p_i, p_j)$ addresses the hyperplane dividing the subspace, where p_i and p_j are located, s is located in the partition p_i corresponding to p_i, and the maximum distance from s to it in its k nearest neighbor samples, denoted as θ, is

$$\theta = \max\left(\mathrm{knn}\left(s, p_i, k\right)\right). \tag{14}$$

Theorem 2. *Given two center points p_i and p_j, $PB(p_i, p_j)$ is the hyperplane dividing the subspace where p_i and p_j are located, s is in the partition p_i corresponding to p_i, and the necessary condition for $p_j \in R^s$ is*

$$\mathrm{dis}\left(s, PB\left(p_i, p_j\right)\right) < \theta. \tag{15}$$

Figure 4(a) shows the case of $\mathrm{dis}(s, PB(p_i, p_j)) > \theta$, and Figure 4(b) shows the case of $\mathrm{dis}(s, PB(p_i, p_j)) < \theta$, respectively. When $\mathrm{dis}(s, PB(p_i, p_j)) > \theta$, there is no intersection between the hypersphere with a radius of θ and the hyperplane $PB(p_i, p_j)$. That is to say, $\forall x \in p_j, \mathrm{dis}(s, x) > \theta$ all hold. At this time, we can directly discard the partition p_j. When $\mathrm{dis}(s, PB(p_i, p_j)) < \theta$, the hypersphere with the radius of θ intersect with the hyperplane $PB(p_i, p_j)$. It means that there is probably a sample point x in the partition p_j, making $\mathrm{dis}(s, x) < \theta$, which also means that the calculated $\mathrm{knn}(s, R^s, k)$ are not the real nearest neighbors. Therefore, we need to add the partition p_j to the relevant partition R^s.

In (Algorithm 3), due to the center point information sorted using TreeMap before, according to Theorem 1, we can get that the initial relevant partition R^s is p_1^s. We calculated k nearest neighbors of s on R^s and saved them in knnDisSet. Next, we judge the subsequent partitions in sequence according to Theorem 2. If the partition $p_i^s (i < k)$ makes $\mathrm{dis}(s, PB(p_1, p_i)) < \theta$, then the partition p_i^s may contain the actual k nearest neighbors of s. We need to include the partition p_i^s in R^s and calculate the k nearest neighbors of s in the latest correlation partition. When a certain partition $p_i^s (i < k)$ appears where $\mathrm{dis}(s, PB(p_1, p_i)) > \theta$, it means that the partition p_i^s does not contain the actual k nearest neighbors of s, and we can discard the partition directly. Meanwhile, since the center point of the subsequent partitions are farther away from s, we also believe that they do not contain the actual k nearest neighbors of s, so we do not continue to make judgments and discard them directly. Finally, we can assume that the knnDisSet stores actual k nearest neighbors of s.

Algorithm 4 describes the specific details of vKNN. Before running the *Map* function, the program loads the center point data. In the *Map* function, call the *selectTrainSet*() function to calculate the k nearest neighbors. After the *selectTrainSet*() function finishes executing, knnSet [k] accepts the k nearest neighbor samples returned. Meanwhile, the *Map* function creates an array trainLable[k] to hold their labels. In the *Reduce* function, we use the *HashMap* to count the labels, find out the label that occur most frequently, and merge the output with s.

In addition, we partition the data, and each partition is roughly the same size, about $|R|/N$. The provided partition selection policy limits the number of filtered partitions. This means that even in the worst case, each sample only needs to be compared with $k \times |R|/N$ training samples to find k nearest neighbors, which greatly reduces the computational effort of the entire KNN join process. Meanwhile, we use k-means clustering in the partitioning process, which makes the sample points in each partition highly similar, and the partition to be filtered is also the most similar to the test sample. Therefore, in theory, we can still maintain a high degree of accuracy while drastically reducing the number of training samples involved in the calculation.

6. Experimental Evaluation

6.1. Experimental Environment and Dataset. The experimental platform used to evaluate the performance of the proposed algorithm is mainly configured as Intel (R) Core (TM) i5-8300H 2.3 GHz processor, 16G memory, and 500G NVMe hard disk. The Hadoop cluster consists of six virtual machines, and each allocating 2G of memory and 40G of hard disk. On each node, we install CentOS 8.0 operating system with Java 1.8.0 and Hadoop 2.10.0. We select one of them as the Master node and the other five as Slave node to be managed through VMware® Workstation 15 Pro. The development environment used in the experiment is Eclipse-jee-2019-12-R-linux-gtk-x86_64.

The experimental data in this study uses the Forest CoverType dataset, a standard dataset in the UCI database.

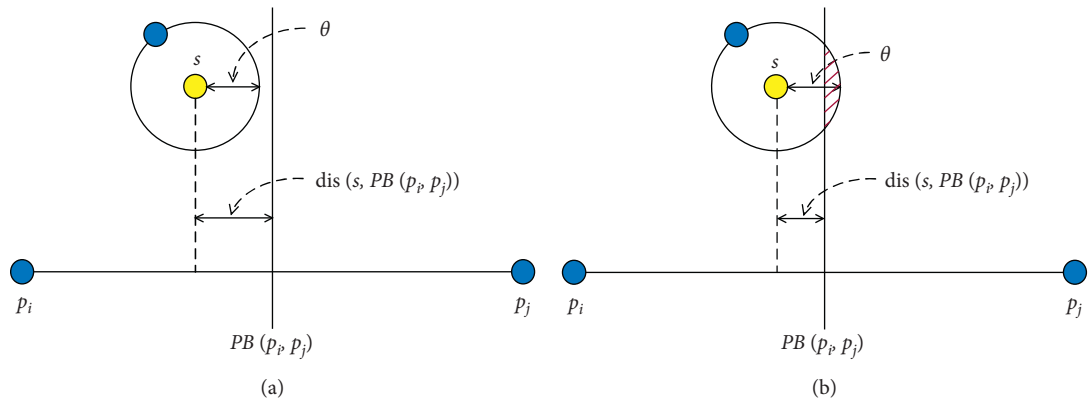

Figure 4: Partition selection. (a) Discard P_j. (b) Reserve P_j.

Input: s, $\{p_1^s, p_2^s, \ldots, p_k^s\}$
Output: knn(x, R^s, k)
selectTrainSet() {
  $R^s = p_1^s$; //determine the initial Relevant-partition
  for $i = 2$ **to** k **do**
    knn(x, R^s, k); //calculate the k nearest neighbors of s in R^s
    θ = max(knn(s, R^s, k)); //calculate the maximum distance from s to its k nearest neighbors from R
    dis$(s, PB(p_1, p_i)) = (\text{dis}(s, p_i))^2 - (\text{dis}(s, p_1))^2/2 \times \text{dis}(p_1, p_i)$; //calculate the distance from s to $PB(p_1, p_i)$,
    if (dis$(s, PB(p_1, p_i)) > \theta$) **then**//$p_i^s$ does not belong to R^s
      break;
    else//p_i^s belongs to R^s
      $R^s = R^s \cup p_i^s$; //add the partition p_i^s to R^s
}

Algorithm 3: Relevant partition selecting.

Input: $\{p_1^s, p_2^s, \ldots, p_k^s\}$, S, k
Output: s.predictLabel//prediction label of s
map: <*row number*, s>
  knnSet[k] ← **selectTrainSet**(); //calculate the k nearest neighbors
  for $i = 0$ **to** k **do**//fetch the k nearest neighbors label
    trainLable[i] = knnSet[i].label;
  for $j = 0$ **to** k **do**
    output(s, trainLabel[j]);
reduce: <s, Labels>
  hmp = new HashMap(); //create a HashMap object hmp
  foreach label ∈ Labels **do**//count the number of each label
    if hmp.get(label) ! = NULL **then**//if the label exists in hmp
      label.value ++; //take the value of the label and add 1
      hmp.put(label) = label.value; //update the value of the label in hmp
    else//if the label does not exist in hmp
      hmp.put(label) = 1; //set the value of the label to 1 and insert to hmp
  predictLabel = hmp.maxvalue;//the label with the largest value as the prediction label
  output(s, predictLabel);

Algorithm 4: vKNN processing.

The dataset has 581012 records, each comprising 54-dimensional features (10 quantitative variables, 4 binary wilderness areas, and 40 binary soil type variables) and seven labels: Spruce/Fir, Lodgepole Pine, Ponderosa Pine, Cottonwood/Willow, Aspen, Douglas-fir, and Krummholz, each represented by a number from 1–7. For simplicity, we first

randomly selected 200,000 data from the dataset as the train set R, and the rest as the test set S.

Consequently, we evaluated the methods mentioned in our experiments. For the two methods described in Section 5, we set the number of reducers to 1 by default.

6.2. Experimental Evaluation Indicators. There are many available evaluation indicators in the classification tasks of machine learning. The two most common types are accuracy and error rate. For a given test set S, the classification error rate is defined as

$$E(S) = \frac{1}{|S|} \sum_{s \in S} \mathrm{I}\left(s_{\text{type}} \neq s_{\text{new\_type}}\right), \tag{16}$$

and the accuracy is defined as

$$\mathrm{Acc}(S) = 1 - E(S) = \frac{1}{|S|} \sum_{s \in S} \mathrm{I}\left(s_{\text{type}} = s_{\text{new\_type}}\right), \tag{17}$$

where s_{type} refers to the actual label of s and $s_{\text{new\_type}}$ refers to the label of s predicted by the model.

In addition, we will evaluate the proposed method in terms of the elapsed time and the acceleration ratio. The elapsed time involves the global time for the MapReduce program to run. The acceleration ratio is the ratio of the elapsed time of the original version to the improved version of the relevant parameters:

$$\text{Speedup} = \frac{\text{original\_time}}{\text{improved\_time}}, \tag{18}$$

where *original_time* is the time when the original version runs and *improved_time* is the time when the program runs after the improved parameter.

6.3. Evaluation of Experimental Results

6.3.1. Effect of Different Center Point Sizes. For our first experiment, we analyze the effect of the number of center points on the performance of vKNN. To further illustrate the situation, the number of mappers is given ($n = 4$). Then, we randomly select 600, 800, 1000, 1200, and 1400 pieces of data from the training set in 5 times as the initial center point set. In Figure 5(a), the execution time of Algorithm 2 increases approximately linearly as N increases. This is because when finding the nearest k center points for each element of S, the distance between each element and each center point needs to be calculated with a time complexity of $O(|S| \times N)$. When the center point increases linearly, it means that the computation time will also increase linearly. We adopt the *TreeMap* for sorting distances, so we need not spend any extra time on it. In Figure 5(b), we can see that vKNN execution time decreases as N increases. This is because, as the number of center points increases, the training set is divided more finely when dividing the partitions, *i.e.*, the number of distance calculations needed for each sample decreases accordingly. Also, we notice that the actual reduction is getting smaller as the center point increases. This is because when the partitions are divided more finely, the probability of occurrence of the scenario shown in Figure 4(b) increases as the set of corresponding partitions is determined. It is shown that some of the samples correspond to a larger set of partitions than before, which also results in increased computation time. Figure 5(c) shows how the accuracy of the algorithm varies from 600 to 1400. As N increases, there is little change in accuracy, which also indicates that the choice of N has no effect on the accuracy of the proposed method.

6.3.2. Effect of the Number of Nearest Neighbors. Next, we study the effect of k on the performance of the two algorithms. Similarly, given the number of mappers ($n = 4$). Figure 6 shows the experimental results of k increasing from 3 to 20 gradually.

Figures 6(a) and 6(b) address the operation of two programs of vKNN algorithm. The running time of the algorithm increases approximately with the increase of k value, which means that the vKNN algorithm is not sensitive to the change of the k value. In Algorithm 2, the effect of k is mainly reflected in the following aspects: we added relevant information about k nearest center points of each sample to the training set. The greater the k value, the more information will be added, and the communication cost of data will increase. When the vKNN algorithm is run at the end, the communication cost of the file on HDFS increases accordingly. The increase of k value shows that θ will be larger in the same sample, and the hypersphere with this radius will be larger and easier to cross the hyperplane, as shown in Figure 4(b). At this point, more sample points will be added to the relevant partition, resulting in more computational effort.

Figure 6(c) shows the change of vKNN algorithm accuracy in the process of k increasing from 3 to 20. When k goes from 3 to 5, the algorithm accuracy is improved. As k continued to increase, the accuracy begins to decrease slightly. Explain that the k value of vKNN algorithm is not as large as possible, and we need to select the appropriate k value for the specific situation.

Figure 6(d) shows that vKNN performs better than KNN for the results of the two methods. The execution time of KNN increases linearly with the increase of k. The influence of k value on KNN algorithm is mainly reflected in the selection of the nearest k sample points. However, because the KNN algorithm itself is too computationally intensive when dealing with large datasets, the increase in the amount of computations caused by the increase in k value is less obvious than the amount of computations itself.

6.3.3. Effect of Speedup. Now, we measure the effect of the number of mappers. Given the number of center points ($N = 1000$) and the number of nearest neighbors ($k = 5$), Figure 7 shows the running time and acceleration ratio of vKNN as the number of mappers gradually increases from 1 to 4. Figure 7(a) shows that the run time decreases as the number of mappers increases. However, the scale is shrinking. This is also reflected in Figure 7(b), where the acceleration ratio gradually stabilizes. It is because

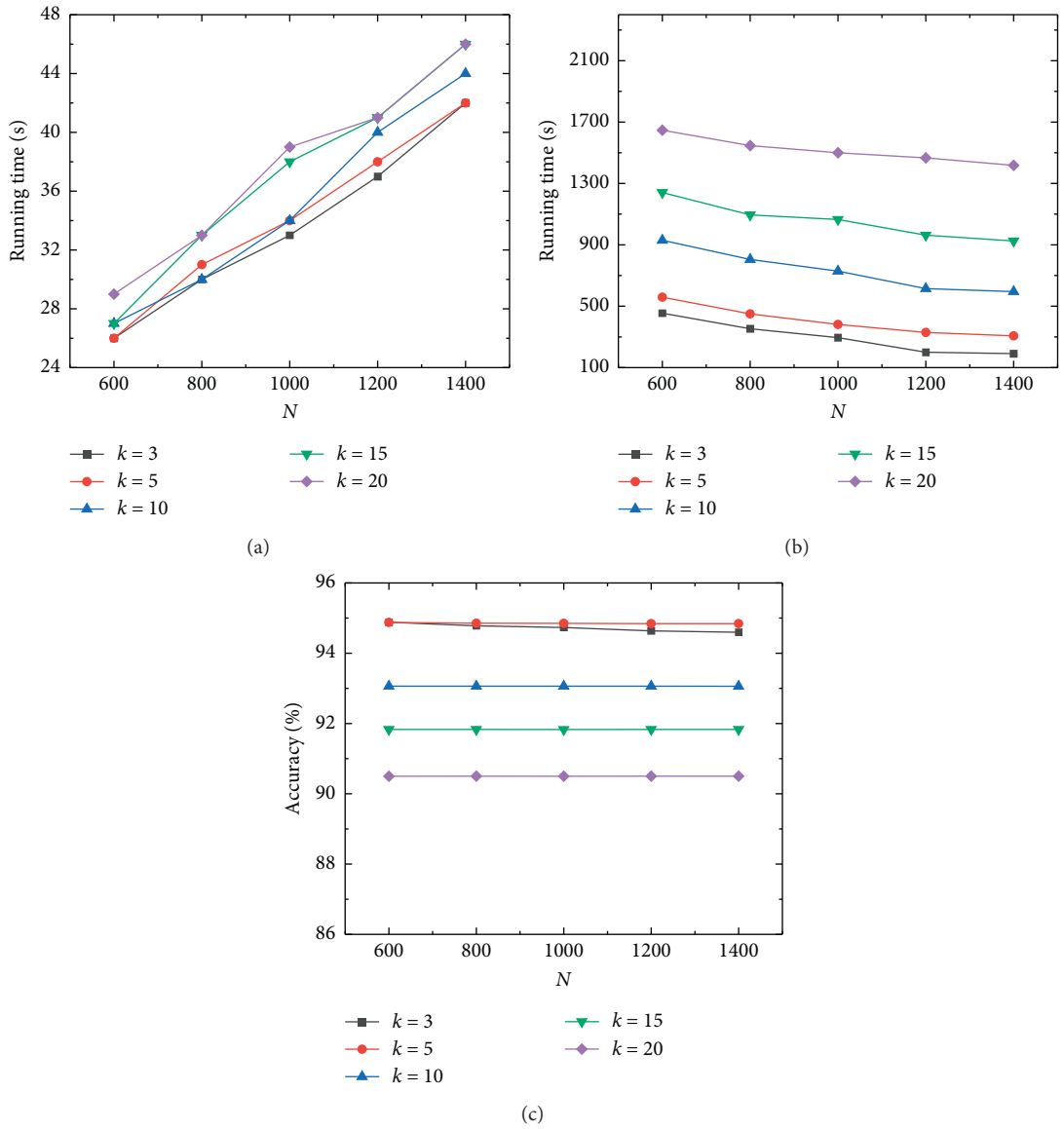

Figure 5: Effect of different center point sizes. (a) Running time of nearest center point selecting. (b) Running time of vKNN. (c) Accuracy of vKNN.

increasing the number of mappers means increasing the number of nodes participating in the operation, at which point the amount of computation allocated to each node will also be reduced, so the execution time of the map function on each node will be shortened accordingly. With the increasing number of nodes, the computing resources are larger than the required resources actually needed. At this time, the computing resources are not fully utilized, the execution time decreases less and less obvious, and cluster computing resources are also wasted. On the contrary, each node needs to read files from the HDFS, which increases the corresponding communication costs. When the acceleration effect of computing node growth is insufficient to offset the pressure of increased communication costs, the algorithm execution efficiency will decrease. Therefore, the number of mappers needs to be selected appropriately when facing datasets of different sizes.

6.3.4. Effect of Accuracy. Now, we study the accuracy of the two algorithms. Given the number of center points ($N = 1000$) and the number of nearest neighbors ($k = 5$). Figure 8 shows the accuracy of the two methods when k is gradually increased from 3 to 20. In general, vKNN is more accurate than KNN. This is because the partitioned data we use is the result of k-means clustering during the preprocessing phase, in which samples with the same characteristics are divided into the same cluster, *i.e.*, the data in the same partition are similar to each other. Subsequently, we search for the center point closest to the sample to be classified. Distance closest represents less difference and more similarity. Therefore, when the vKNN algorithm is executed and compared with other samples, the higher the degree of similarity with the sample to be classified, the smaller the calculation error will be.

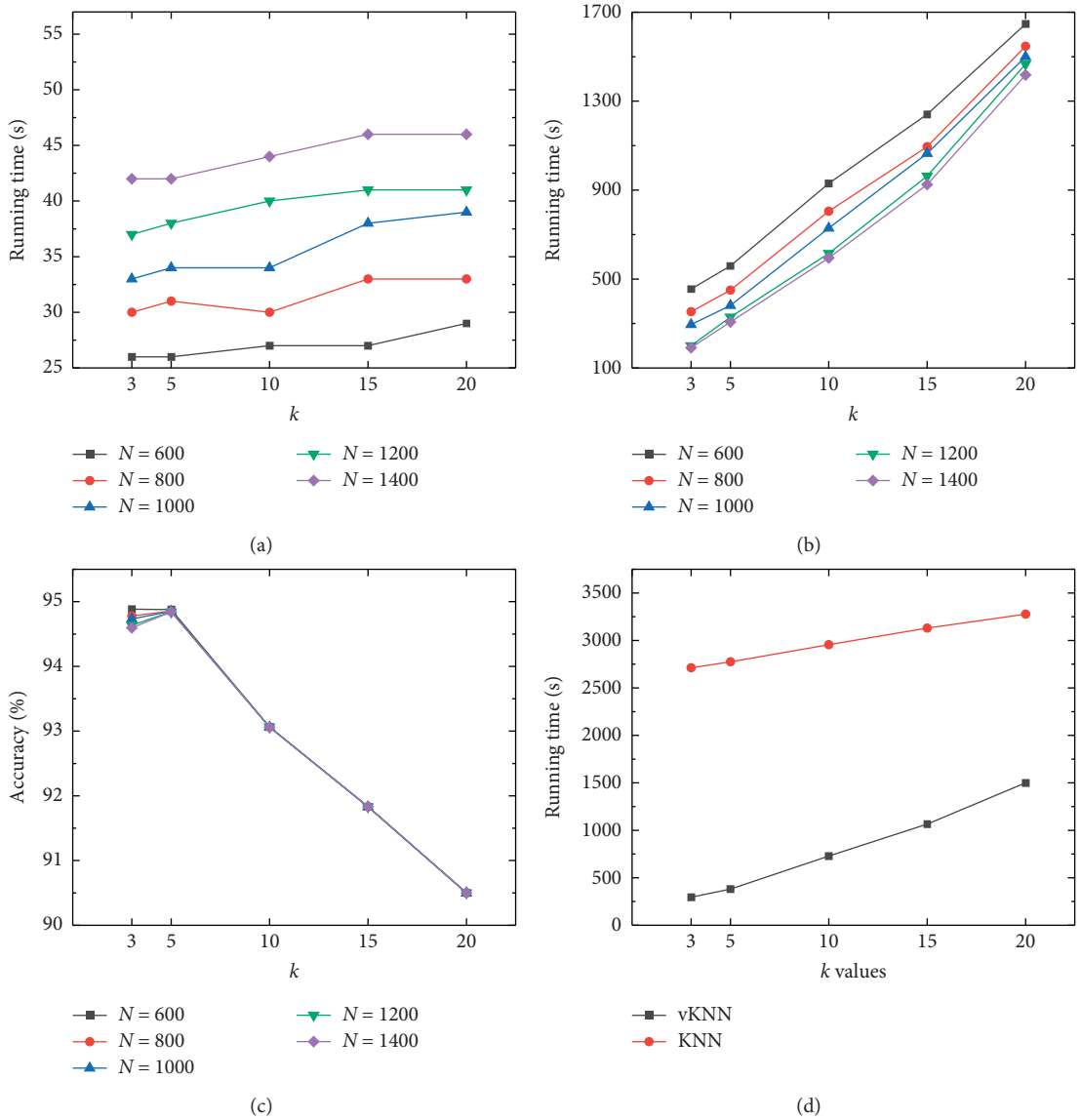

Figure 6: Effect of the number of nearest neighbors. (a) Running time of nearest center point selecting. (b) Running time of vKNN. (c) Accuracy of vKNN. (d) Running time comparison of different methods.

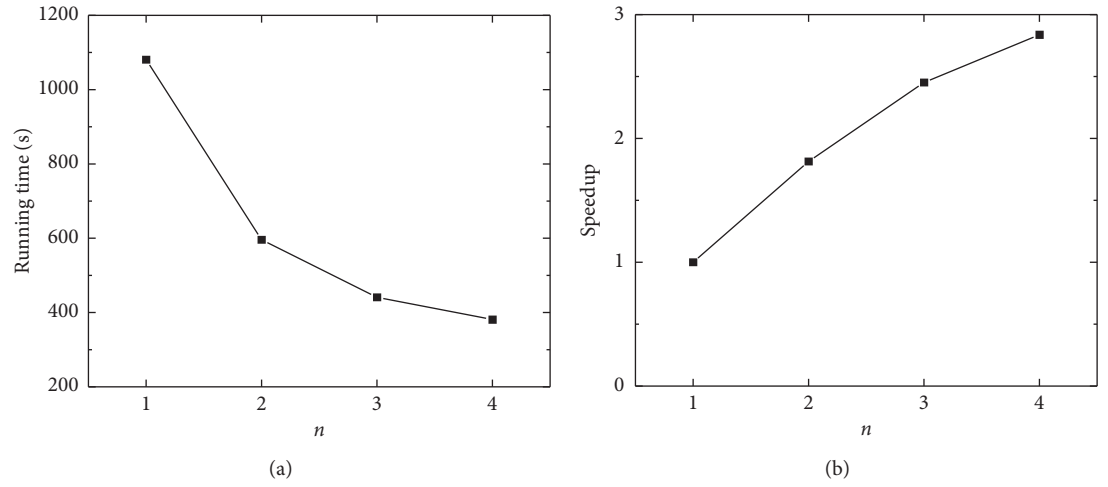

Figure 7: (a) Effect of the number of mappers. (b) Speedup.

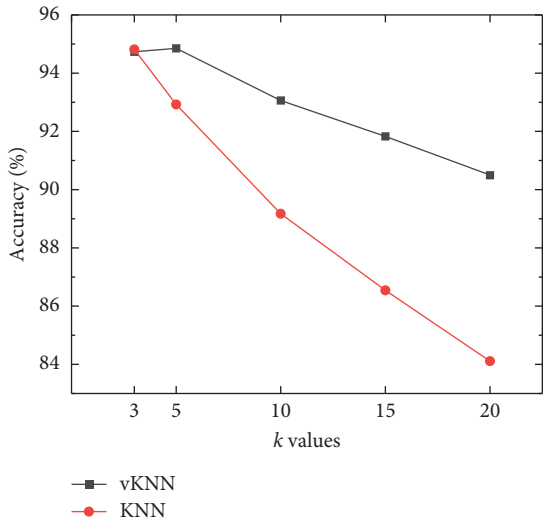

Figure 8: Accuracy of different methods.

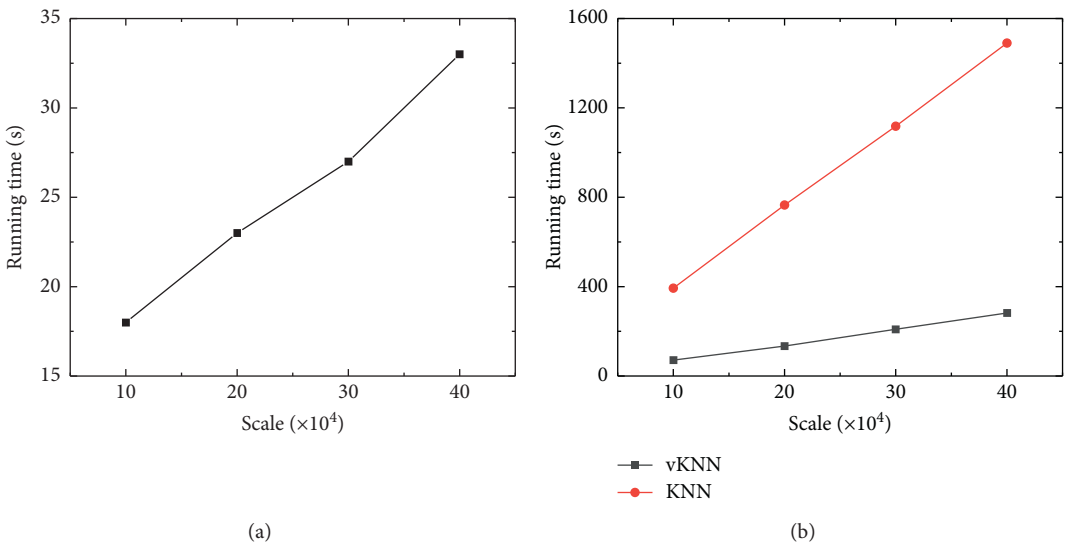

Figure 9: Scalability. (a) Running time of nearest center point selecting. (b) Running time of different methods.

6.3.5. Effect of Scalability. In this section, we randomly extract 100,000 pieces of data from the original dataset as a training set. Then, randomly extract 100,000, 200,000, 300,000, and 400,000 pieces of data from the rest of the set four times as a test set. Based on this, the scalability of the proposed method is evaluated. In the experiment, given the number of center points ($N = 1000$) and the number of nearest neighbors ($k = 5$). Figure 9(a) is the time-consuming result of Algorithm 2, and Figure 9(b) is the time-consuming result of two methods for performing KNN join.

On the whole, the execution time of both methods increases approximately linearly with the increase of training set data size. The reason is that, as the data size increases, the data allocated to each computing node will also increase proportionally, and the computing time of each node will increase in the same proportion. However, when using vKNN algorithm to perform KNN join, the growth is gentler. The average growth time of Algorithm 2 is only 5 seconds, which indicates that the scalability of vKNN algorithm is better than that of KNN algorithm. The time complexity of KNN algorithm is $O(|R| \cdot |S|)$, which is a Cartesian product of the sample set R and S. Obviously, no matter which sample set is increased in size, the effect is enormous. However, vKNN algorithm adopts the idea of partition which alleviates the computational changes caused by the increase of S. Therefore, as the dataset increases, the difference in execution time between the two methods increases.

7. Conclusion

In this paper, we propose a vKNN algorithm based on Voronoi diagram concerning MapReduce-based KNN join scheme. Our algorithm can partition the training set using the idea of Voronoi diagram. Then, we design a partition selection strategy to find the optimal relevant partition for

the sample to be tested, which effectively avoids the enormous amount of computation caused by irrelevant data. This strategy takes full advantage of the parallel processing capabilities of the MapReduce framework and is suitable for large-scale data. A large number of experiments based on real datasets show that our proposed algorithm can accelerate the calculation with good scalability while ensuring accuracy.

Data Availability

The classification data used to support the findings of this study has been taken from the UCI machine learning repository (http://archive.ics.uci.edu/ml/index.php).

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was partially supported by the Hubei Graduate Workstation with Jingpeng Software Group Co., Ltd.

References

[1] L. Wan, Y. Sun, I. Lee, W. Zhao, and F. Xia, "Industrial pollution areas detection and location via satellite-based IIOT," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 6, pp. 2353–2364, 2016.

[2] L. Wan, L. Sun, X. Kong, Y. Yuan et al., "Task-driven resource assignment in mobile edge computing exploiting evolutionary computation," *IEEE Wireless Communications*, vol. 26, no. 6, pp. 94–101, 2019.

[3] X. Wang, L. Wan, M. Huang, C. Shen, Z. Han, and T. Zhu, "Low-complexity channel estimation for circular and noncircular signals in virtual MIMO vehicle communication systems," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 4, pp. 3916–3928, 2020.

[4] F. Wen and J. Shi, "Fast direction finding for bistatic EMVS-MIMO radar without pairing," *Signal Process*, vol. 173, p. 107532, 2020.

[5] L. Wan, X. Kong, and F. Xia, "Joint range-Doppler-angle estimation for intelligent tracking of moving aerial targets," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1625–1636, 2018.

[6] D. Meng, X. Wang, M. Huang, L. Wan, and B. Zhang, "Robust weighted subspace fitting for DOA estimation via block sparse recovery," *IEEE Communications Letters*, vol. 24, no. 3, pp. 563–567, 2020.

[7] J. Li, X. Zhang, R. Cao, and M. Zhou, "Reduced-dimension MUSIC for angle and array gain-phase error estimation in bistatic MIMO radar," *IEEE Communications Letters*, vol. 17, no. 3, pp. 443–446, 2013.

[8] F. Wen, Z. Zhang, K. Wang, G. Sheng, and G. Zhang, "Angle estimation and mutual coupling self-calibration for ULA-based bistatic MIMO radar," *Signal Processing*, vol. 144, pp. 61–67, 2018.

[9] T. Liu, F. Wen, L. Zhang, and K. Wang, "Off-grid DOA estimation for colocated MIMO radar via reduced-complexity sparse bayesian learning," *IEEE Access*, vol. 7, pp. 99907–99916, 2019.

[10] F. Wen, X. Xiong, and Z. Zhang, "Angle and mutual coupling estimation in bistatic MIMO radar based on PARAFAC decomposition," *Digital Signal Processing*, vol. 65, pp. 1–10, 2017.

[11] F. Wen, X. Xiong, J. Su, and Z. Zhang, "Angle estimation for bistatic MIMO radar in the presence of spatial colored noise," *Signal Processing*, vol. 134, pp. 261–267, 2017.

[12] X. Wang, L. Wang, X. Li, and G. Bi, "Nuclear norm minimization framework for DOA estimation in MIMO radar," *Signal Processing*, vol. 135, pp. 147–152, 2017.

[13] J. Dean and S. Ghemawat, "MapReduce," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[14] J. Dean and S. Ghemawat, "MapReduce," *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[15] S. Dolev, P. Gupta, Y. Li, S. Mehrotra, and S. Sharma, "Privacy-preserving secret shared computations using MapReduce," *IEEE Transactions on Dependable and Secure Computing*, 2019.

[16] J. Liu, P. Wang, J. Zhou, and K. Li, "McTAR: a multi-trigger checkpointing tactic for fast task recovery in MapReduce," *IEEE Transactions on Services Computing*, 2019.

[17] J. Wang, X. Li, R. Ruiz, J. Yang, and D. Chu, "Energy utilization task scheduling for MapReduce in heterogeneous clusters," *IEEE Transactions on Services Computing*, 2020.

[18] J. Ekanayake, H. Li, B. Zhang et al., "Twister: a runtime for iterative MapReduce," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pp. 810–818, Chicago, IL, USA, June 2010.

[19] J. Ekanayake and S. Pallickara, "MapReduce for data intensive scientific analyses," in *Proceedings of the 2008 IEEE Fourth International Conference on eScience*, pp. 277–284, IEEE, Indianapolis, IN, USA, December 2008.

[20] M. Babar, F. Arif, M. A. Jan, Z. Tan, and F. Khan, "Urban data management system: towards big data analytics for internet of things based smart urban environment using customized Hadoop," *Future Generation Computer Systems*, vol. 96, pp. 398–409, 2019.

[21] J. Liu, Q. Liu, L. Zhang, S. Su, and Y. Liu, "Enabling massive XML-based biological data management in HBase," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2019.

[22] Y. Li, A. Eldawy, J. Xue, N. Knorozova, M. F. Mokbel, and R. Janardan, "Scalable computational geometry in MapReduce," *The VLDB Journal*, vol. 28, no. 4, pp. 523–548, 2019.

[23] F. Li, J. Chen, and Z. Wang, "Wireless MapReduce distributed computing," *IEEE Transactions on Information Theory*, vol. 65, no. 10, pp. 6101–6114, 2019.

[24] J. Cui, Z. An, Y. Guo, and S. Zhou, "Efficient nearest neighbor query based on extended B^+-tree in high-dimensional space," *Pattern Recognition Letters*, vol. 31, no. 12, pp. 1740–1748, 2010.

[25] C. Xia, H. Lu, B. C. Ooi, and J. Hu, "Gorder," *Proceedings 2004 VLDB Conference*, vol. 30, pp. 756–767, 2004.

[26] D. Amagata, T. Hara, and C. Xiao, "Dynamic set kNN self-join," in *Proceedings of the IEEE 35th International Conference on Data Engineering (ICDE)*, pp. 818–829, IEEE, Macao, China, April 2019.

[27] W. Zhao, H. Ma, and Q. He, "Parallel *K*-means clustering based on MapReduce," in *Proceedings of the IEEE International Conference on Cloud Computing*, Springer, Beijing, China, pp. 674–679, October 2009.

[28] J. Santos, T. Syed, M. C. Naldi, R. J. G. B. Campello, and J. Sander, "Hierarchical density-based clustering using MapReduce," *IEEE Transactions on Big Data*, 2019.

[29] J. Rosen, N. Polyzotis, V. Borkar et al., "Iterative MapReduce for large scale machine learning," 2013, https://arxiv.org/abs/1303.3517.

[30] C. Zhang, F. Li, and J. Jestes, "Efficient parallel kNN joins for large data in MapReduce," in *Proceedings of the 15th International Conference on Extending Database Technology (EDBT)*, pp. 38–49, Berlin, Germany, March 2012.

[31] P. Moutafis, G. Mavrommatis, M. Vassilakopoulos, and S. Sioutas, "Efficient processing of all-*k*-nearest-neighbor queries in the MapReduce programming framework," *Data & Knowledge Engineering*, vol. 121, pp. 42–70, 2019.

[32] J. Yu, Z. Zhang, and M. Sarwat, "Spatial data management in Apache spark: the geospark perspective and beyond," *Geoinformatica*, vol. 23, no. 1, pp. 37–78, 2019.

[33] C. Ji, T. Dong, Y. Li et al., "Inverted grid-based KNN query processing with mapreduce," in *Proceedings of the 2012 Seventh ChinaGrid Annual Conference*, IEEE, Beijing, China, September 2012.

[34] G. Song, J. Rochas, L. E. Beze, F. Huet, F. Magoules et al., "K nearest neighbour joins for big data on mapreduce: a theoretical and experimental analysis," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 9, pp. 2376–2392, 2016.

[35] C. Bohler, R. Klein, and C.-H. Liu, "An efficient randomized algorithm for higher-order abstract Voronoi diagrams," *Algorithmica*, vol. 81, no. 6, pp. 2317–2345, 2019.

[36] C. Song, J. Cha, M. Lee, and D.-S. Kim, "Dynamic Voronoi diagram for moving disks," *IEEE Transactions on Visualization and Computer Graphics*, 2019.

[37] C.-H. Liu, "A nearly optimal algorithm for the geodesic Voronoi diagram of points in a simple polygon," *Algorithmica*, vol. 82, no. 4, pp. 915–937, 2020.

[38] F. García-García, A. Corral, L. Iribarne et al., "Improving distance-join query processing with voronoi-diagram based partitioning in SpatialHadoop," *Future Generation Computer Systems*, vol. 111, pp. 723–740, 2019.

[39] W. Huang, K. Sun, J. Qi, and J. Ning, "Optimal allocation of dynamic var sources using the Voronoi diagram method integrating linear programing," *IEEE Transactions on Power Systems*, vol. 32, no. 6, pp. 4644–4655, 2017.

[40] F. García-García, A. Corral, L. Iribarne, and M. Vassilakopoulos, "Voronoi-diagram based partitioning for distance join query processing in Spatialhadoop," in *Proceedings of the MEDI Conference*, pp. 251–267, Marrakesh, Morocco, October 2018.

[41] A. Verma, B. Cho, N. Zea, I. Gupta, and R. H. Campbell, "Breaking the MapReduce stage barrier," *Cluster Computing*, vol. 16, no. 1, pp. 191–206, 2013.

[42] L. Fan, H. Li, and C. Li, "The improvement and implementation of distributed item-based collaborative filtering algorithm on hadoop," in *Proceedings of the 34th Chinese Control Conference*, pp. 9078–9083, Hangzhou, China, July 2015.

[43] A. K. Kamn and K. Chitharanjan, "A review on hadoop—HDFS infrastructure extensions," in *Proceedings of the 2013 IEEE Conference on Information & Communication Technologies (ICT)*, pp. 132–137, IEEE, Tamil Nadu, India, April 2013.

[44] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon, "Parallel data processing with MapReduce," *ACM SIGMOD Record*, vol. 40, no. 4, pp. 11–20, 2012.