BILINGUAL PUBLISHING CO.
Pioneer of Global Academics Since 1984

Artificial Intelligence Advances
https://ojs.bilpublishing.com/index.php/aia

ARTICLE

Efficient Parallel Processing of k-Nearest Neighbor Queries by Using a Centroid-based and Hierarchical Clustering Algorithm

Elaheh Gavagsaz[*]

Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran

ARTICLE INFO

Article history
Received: 27 April 2022
Revised: 19 May 2022
Accepted: 20 May 2022
Published: 26 May 2022

Keywords:
Classification
k-Nearest Neighbor
Big data
Clustering
Parallel processing

ABSTRACT

The k-Nearest Neighbor method is one of the most popular techniques for both classification and regression purposes. Because of its operation, the application of this classification may be limited to problems with a certain number of instances, particularly, when run time is a consideration. However, the classification of large amounts of data has become a fundamental task in many real-world applications. It is logical to scale the k-Nearest Neighbor method to large scale datasets. This paper proposes a new k-Nearest Neighbor classification method (KNN-CCL) which uses a parallel centroid-based and hierarchical clustering algorithm to separate the sample of training dataset into multiple parts. The introduced clustering algorithm uses four stages of successive refinements and generates high quality clusters. The k-Nearest Neighbor approach subsequently makes use of them to predict the test datasets. Finally, sets of experiments are conducted on the UCI datasets. The experimental results confirm that the proposed k-Nearest Neighbor classification method performs well with regard to classification accuracy and performance.

1. Introduction

Due to developments in technology, information gathering has become an ongoing and relatively economical activity. That has led to such an exponential increase in the available data rate. Social networks, sensor networks, mobile phones, and tablets are examples of applications that generate tons of data every day [1-3]. The large volume of data can be useful if the correct knowledge extraction methods could leverage it. There is a major challenge for researchers and industry that standard machine learning methods can not address the volume, diversity, and complexity of the vast data collection [4]. Hence, the current learning methods need to be improved and scaled to leverage such a volume of data.

The k-Nearest Neighbor method is one of the most popular techniques for both classification and regression purposes [5]. Because of the simplicity, clarity, and high performance of the k-Nearest Neighbor algorithm, it is one of the ten most popular data mining methods [6]. It is also a

*Corresponding Author:
Author] Elaheh Gavagsaz,
Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran;
Email: elaheh.gavagsaz@srbiau.ac.ir; egavagsaz@yahoo.com

DOI: https://doi.org/10.30564/aia.v4i1.4668

non- parametric lazy learning technique. Non-parametric implies that the k-Nearest Neighbor approach makes no assumptions about data distribution. Lazy learning represents that this technique prevents model generation before a query is generated, as opposed to other methods of creating a model based on the training data. Therefore, the method requires the storage of all the training data. Next, it selects the k nearest training data for each test data item. Then, it measures the distance or similarity of all training data and every element of test data. This pairwise calculation must be repeated against the entire training data for all of the test data items [7,8]. The approach is impractical in big data. Hence, a parallel and distributed computing environment must be used to produce results in a reasonable period of time.

Recent programming models provide ideal environments to cope with these problems by providing all the computational and memory resources we need. MapReduce framework [9,10] is a programming model for distributed computations and large-scale data processing on clusters. Its architecture offers the appropriate scalability and fault tolerance. Hadoop [11] is one of the first implementations of this framework and is also an open-source implementation. This is an effective method for handling data-intensive applications based on data locality [12,13]. This technology is commonly used in diverse fields such as web search, log analysis and data mining. This, with its unquestionable developments, has some limits. One of the major disadvantages of Hadoop is the retrieval of data from a distributed file system. Therefore, it is inefficient for applications like iterative algorithms or multi-step of data.

Spark [14] was designed to overcome the limitations of the Hadoop. It is also an open-source of the MapReduce framework providing in-memory computation. Spark introduces Resilient Distributed Datasets (RDDs) that are read-only datasets with data items distributed over the computing nodes. RDDs are creating an appropriate kind of distributed shared memory to implement iterative algorithms. This in-memory computation results in a significant improvement in efficiency. The most significant use of in-memory data is in machine learning scenarios [15]. Spark has been highlighted for its ease of use as an effective tool and has been used in many scenarios [16-22].

This paper proposes a new k-Nearest Neighbor classification method which uses a parallel centroid-based and hierarchical clustering algorithm (KNN-CCL) to reduce and obtain the appropriate training dataset. This method is a two-phases approach based on MapReduce, implemented within the Spark framework. In the first phase, a parallel centroid-based clustering algorithm has been used to separate the sample of the training dataset into multiple clusters. In the second phase, the cluster nearest to each test data item was chosen as the training dataset. Then, for each test data item, the k-Nearest Neighbor classification was applied to predict this in the training dataset. The experimental results on real datasets confirm that the proposed k-Nearest Neighbor classification method performs well with regard to classification accuracy and performance compared with the conventional classification k-Nearest Neighbor and similar previous method. The main contributions of this paper can be summarized as follows:

- A new model is proposed to improve k-Nearest Neighbor classification method by using a two-phases approach based on MapReduce, implemented within the Spark framework.
- A parallel centroid-based and hierarchical clustering algorithm is applied to separate the sample of training dataset into multiple parts.
- The appropriate cluster for each test data item is determined as new training dataset to reduce the calculation of the k-Nearest Neighbor approach.
- Sets of experiments are conducted on the UCI datasets. The experimental results confirm that the proposed k-Nearest Neighbor classification method performs well in terms of classification accuracy and performance.

The rest of the paper is structured according to this. The related works are briefly reviewed in Section 2. Section 3 deals with the overall system and proposed workflow process for the k-Nearest Neighbor classification. The training process is discussed briefly in Section 4 that consists of the clustering method and its parallelization. Then, Section 5 explains the testing process. The experimental results are summarized in Section 6. Finally, Section 7 concludes the paper.

2. Related Work

Research into the Nearest Neighbor technique has attracted considerable attention because of its simplicity and effectiveness since it was proposed in 1967 [23]. This method has been widely used in various applications including recommendation systems, databases, pattern recognition, data compression, DNA sequencing, etc.

The Nearest Neighbor search is an optimization problem to find a minimum distance or maximum similarity instance set [5,8]. A famous generation of Nearest Neighbor searches is k-Nearest Neighbor. The search for k-Nearest Neighbor finds *k* instances that minimize distance function. This approach has two different implementations: exact k-Nearest Neighbor search, and approximate k-Nearest

Neighbor search. The former performs a pair-wise comparison in all instances. It is obvious that the k-Nearest Neighbor has high computational cost. Although, there are some techniques for indexing the feature space and reducing computational complexity. The second one does not strive to implement a precise k-Nearest Neighbor. It uses subsets of the training data to minimize calculations [24,25].

By applying sophisticated data structures and pruning techniques, the traditional k-Nearest Neighbor methods in big data have attempted to solve the problem in a single machine [26-30]. However, we are just interested in distributed and parallel approaches. Triguero et al. proposed an evolutionary method for classification based on sampling [31]. It uses two phases of MapReduce: In the first phase, after sampling, a decision tree is established in each map. The test set is classified into the second phase by applying the set of trees. This uses a partial k-Nearest Neighbor over the subsets of training data. Du et al. described a clustering of density peaks dependent on k-Nearest Neighbor for large datasets [32]. This represents the key component analysis for processing large-dimensional data. This method is sensitive to the parameters of density estimation.

Deng et al. suggested a novel approach by using two different steps to cluster big data [33]. In the first step, k-means clustering is performed to separate the entire training dataset. Thus, this step creates k clusters and cluster centers. In the second step, the nearest cluster center is found for each test data item and the corresponding cluster is used as the training dataset for it. Then, a k-Nearest Neighbor is applied for the classification of the test data. Although distributed solutions are not included in this approach, it works relatively well in terms of accuracy and efficiency.

There are several studies conducting k-Nearest Neighbor join queries in the MapReduce [34,35]. The method described by Moutafis et al. applies five phases of the MapReduce [34]. The data space is subdivided into a grid of cells of equal size. The number of training data is determined per cell at first phase. In the second phase, an initial k-Nearest Neighbor list is generated for each test data, based on the training data in the same cell. In the third phase, the lists from the previous phase are verified by collecting more training data from neighboring cells. For each item of test data, plane-sweep technique is applied when required. The lists of k-Nearest Neighbor are joined into the final lists in the fourth phase. In the final phase, the classification of each test data item is performed based on the class of its neighbors. In this approach, load balancing is improved by the implementation of an adaptive partitioning scheme based on Quadtrees.

Chatzimilioudis et al. have introduced a batch-oriented algorithm called Spitfire [36]. This algorithm has its own distributed procedure and does not follow a MapReduce model. It employs three steps: split, refine, and replicate. At first, the data space is partitioned into disjoint sub-areas. Then, at each split, the k-Nearest Neighbors are calculated and replicated in splits. The final result is determined in the last step. This algorithm is implemented using the distributed file system Tachyon and Parallel Java Library for MPI.

Sun et al. proposed a method to classify facial images using Hadoop and k-Nearest Neighbor [37]. It splits the training dataset into several disjoint parts using the Map phase. This method uses three MapReduce processes for scanning the images, extracting facial features, and recognition for every single item of test data. This method conducts the MapReduce iteratively for each test data item resulting in very time-consuming operations. The exact implementation of the k-Nearest Neighbor algorithm is shown by Maillo et al. [38]. It uses a MapReduce process to classify the test dataset against the training dataset. The training dataset is divided into a certain number of disjoint partitions during the Map phase. The whole test dataset is sent to all maps. Hence, the test dataset is read line by line from the HDFS. The distance of each test data item is measured against the training dataset in each map. The class label and distance of k nearest neighbors are saved for each test data item. Each map sends its result to a single reduce task when its processing is complete. Then, the reduce task determines final neighbors for every test data item from the lists obtained during the map phase. Maillo et al. extended this method by the use of multiple reducers and in-memory solutions [39]. These methods have some limitations: first, these techniques add an enormous number of parameters. Second, the test data are inefficiently iterated on the driver. However, this paper introduces an approximate k-Nearest Neighbor algorithm, it has relatively good classification accuracy. In addition, the proposed method reduces the time complexity of the k-Nearest Neighbor classification compared to other distributed k-Nearest Neighbor methods.

3. System Overview

k-Nearest Neighbor algorithm calculates the distance in the training dataset between each test data item and all training data items and returns k nearest items. Linear time complexity is required to find the exact k nearest neighbors. Let n be the number of the training dataset and d be the number of dimensions, $O(nd)$ is the computational complexity for each test item [40]. Thus, it is very expensive for big data. This paper presents a new training process for the k-Nearest Neighbor classification. The proposed method does not perform an exact k-Nearest Neighbor but it obtains high accuracy in classification.

Figure 1 shows proposed workflow process for k-Nearest Neighbor classification. Before the test process, a proposed clustering method will partition a sample of the training dataset into several clusters. The training data are more similar in each cluster than in other clusters. Then, the cluster centers are determined for all of the clusters obtained. The next step shall be to select the nearest cluster center to each test item. For each test item, its corresponding cluster is applied as a new training dataset. k-Nearest Neighbor classification is used to predict class of test items.

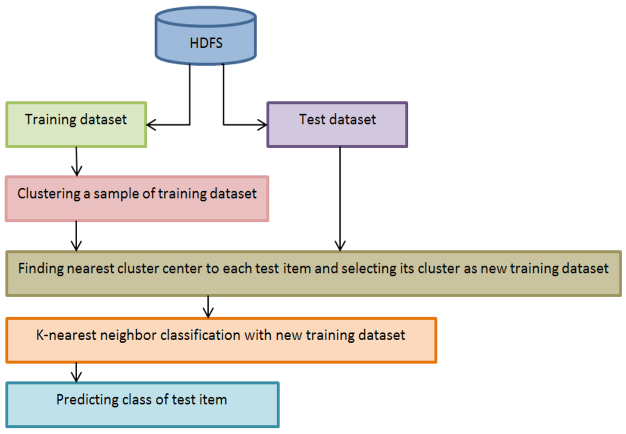

HDFS
Training dataset
Test dataset
Clustering a sample of training dataset
Finding nearest cluster center to each test item and selecting its cluster as new training dataset
K-nearest neighbor classification with new training dataset
Predicting class of test item

Figure 1. The proposed workflow process for the k-Nearest Neighbor classification.

The proposed algorithm consists of two main processes: training and testing. The training process is designed to partition a sample of the training dataset into appropriate clusters. A proper method of clustering is required for this process. The testing process is organized to find the cluster that is nearest to each test item. Then, the selected cluster is used as the training dataset to classify the test item. The k-Nearest Neighbor algorithm is applied in this process.

4. Training Process

Cluster analysis is one of the fundamental and widely used knowledge-discovery techniques. Clustering algorithms can be used in various applications for data mining such as recommendation systems, data compression, and data preprocessing. As described earlier, clustering is the task of grouping a set of data in such a way that data in the same group are more similar to each other than data in other groups. In other words, data have high similarity within a cluster and low similarity between clusters. An example of the clustering is shown in Figure 2.

The most common methods of clustering can be divided into four main categories: density-based clustering, grid-based clustering, partition-based clustering, and hierarchical clustering. The density-based clustering methods are unsupervised learning techniques [41,42]. Their principal idea is that a cluster is determined by an adjacent area of high data density. Low-density areas are the dividing parts and data in those parts are considered outliers.

The second category of clustering algorithms is the grid-based clustering method [43,44]. A set of grid cells is defined within data space. Then, each data item is allocated to a suitable grid cell. The density of each cell shall be calculated in the next step. In the end, the adjacent groups of dense cells establish clusters.

The partition-based clustering methods establish initial partitions of a dataset into a set of clusters of a given number [45]. Then, they iterate refinements to maximize intra-cluster similarity and inter-cluster dissimilarity. The partition-based clustering approach is also called objective function clustering. The explanation for this is that a certain objective function is minimized. For instance, the k-means algorithm [46] minimizes the mean square error of each data item in a cluster as regards its cluster centroid. This method is simple, fast, and effective. It is widely used in practical applications. However, it depends on selecting the *k* initial points.

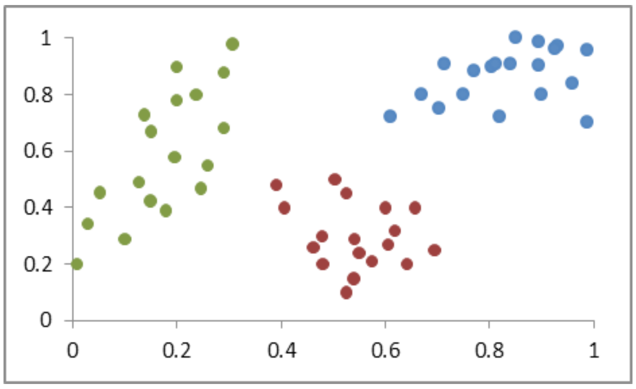


Figure 2. An example of clustering.

The hierarchical clustering algorithms [47] form a hierarchy for clusters. They have two strategies for data decomposition: agglomerative and divisive methods. An agglomerative method is a bottom-up approach. At first, every item in the data is in a cluster. Then, an appropriate criterion merges pairs of clusters as one. A divisive method is a top-down approach. First, all items in the data are in one cluster. Next, the cluster is separated until each item of the data is a separate cluster. These methods reflect the proper accuracy of clustering. For instance, BIRCH [48] is an accurate method that does not rely on initial parameters for its results and can not be influenced by outliers. However, it is not as fast as the k-means method.

The hierarchical clustering methods lose performance and scalability, as they try to form clusters of high quality.

However all studies define unsupervised learning as the most important clustering feature, all major methods require some user information. For instance, the k-means method requires the number of clusters to be obtained and the hierarchical clustering methods usually need to get the termination condition. A new clustering using a binary splitting method is proposed by Mazzeo et al., called CLUBS$^+$, to resolve this major limitation [49]. It is a centroid-based and hierarchical clustering method profiting from the combination of divisive and agglomerative algorithms. First, Masciari et al. demonstrated the idea of divisive and agglomerative corporate behavior [50], and then its functionality defined [51]. CLUBS$^+$ has improved its predecessor approach with two refinements. It surpasses the detection of outliers for the recognition of high-density outliers from low-density clusters areas. It also creates ideal ellipsoid clusters around the centroids. Lanni et al. represented a parallel version of CLUBS$^+$ called CLUBS-p [52]. It scales linearly with respect to dataset size. These characteristics make it a very powerful algorithm for clustering large quantities of data. Because of the listed features, this paper uses a parallel version of CLUBS$^+$ to cluster the training dataset.

4.1 Clustering Using Binary Splitting

CLUBS$^+$ is introduced as a parameter-free clustering algorithm that uses the binary splitting to partition data space. It is also a centroid-based and fast hierarchical clustering algorithm collaborating with the advantages of the divisive and agglomerative techniques. This algorithm consists of four phases: (1) divisive (2) intermediate refinement (3) agglomerative and (4) final refinement. The input dataset is split into several rectangular blocks by a divisive algorithm in the first phase. Then, a refinement process is performed on these blocks. That leads to the formation of new clusters of points. Some points are introduced as outliers and classified into one cluster. It uses an agglomerative algorithm in the third phase to merge some clusters into one cluster. Finally, clusters from the preceding phase and outliers from the second phase are refined and the final clusters and final set of outliers are obtained in this phase. Figure 3 represents the CLUBS$^+$ algorithm operations.

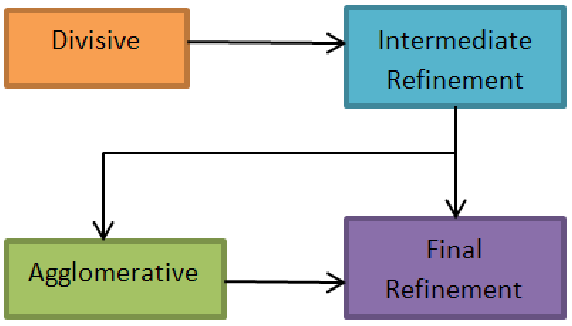


Figure 3. The four phases of the CLUBS$^+$ algorithm.

4.1.1 Problem Definition

By default, a dataset $D=\{p_1, p_2, ..., p_n\}$ is considered where p_i is a d-dimensional point. The aim is to obtain a cluster set $CS=\{C_1, C_2, ..., C_k\}$ for the points in D such that points in the same cluster are the most similar, and points in the different clusters are the least similar. It is notable that each point p_i is assigned only one cluster. In this paper, the Euclidean distance will be used as a measure of similarity between pairs of points. It means that the lower distance between pairs of points represents the greater similarity.

Assume $p \in D$ is a d-dimensional point and p_i is indicating its i-th coordinate. The Euclidean distance of point p to point q is given in Equation (1):

$$\|p-q\| = \sqrt{\sum_{i=1}^{d} (p_i - q_i)^2} \tag{1}$$

The following defines several concepts and analytical characteristics that will be used in the CLUBS$^+$ algorithm.

Suppose that $C=\{p_1, p_2, ..., p_m\}$ is a common cluster that includes m d-dimensional points. Let p_{ij} denotes j-th coordinate of p_i. The point $\overline{cp} = (\overline{cp_1}, \overline{cp_2}, ..., \overline{cp_d})$ is the centroid point of C such that $\overline{cp_j}$ is the average of the j-th coordinates of all points in C. The formula $\overline{cp_j}$ is given in Equation (2):

$$\overline{cp_j} = \frac{1}{m} \sum_{i=1}^{m} p_{ij} \quad , (1 \le j \le d) \tag{2}$$

For this algorithm, a further measurement called *WCSS* (Within Cluster Sum of Squares) is used. For each cluster the *WCSS* is defined as follows:

$$WCSS(C) = \sum_{i=1}^{m} \|p_i - \overline{cp}\|^2 = \sum_{i=1}^{m}\sum_{j=1}^{d} (p_{ij} - \overline{cp_j})^2 \tag{3}$$

This measure can be determined effectively using some computed aggregate information for each cluster.

Theorem 1

Assume *C* is an individual cluster with a set of *m d*-dimensional points. Then,

$$WCSS(C) = \sum_{j=1}^{d} (Q_j - \frac{S_j^2}{m}) \tag{4}$$

where, $S_j = \sum_{i=1}^{m} p_{ij}$ is the sum of the *j*-th coordinates and $Q_j = \sum_{i=1}^{m} p_{ij}^2$ is the sum of the squares of the *j*-th coordinates of the points in cluster *C* [49].

Suppose $CS=\{C_1, C_2, ..., C_k\}$ is a set of clusters over a dataset, $D = \{p_1, p_2, ..., p_n\}$, and each C_i cluster has m_i points. The definition of *WCSS*(*CS*) is in Equation (5):

$$WCSS(CS) = \sum_{i=1}^{k} WCSS(C_i) \tag{5}$$

Let $\overline{tcp}$ be the centroid point on dataset *D.* The *BCSS* (Between Clusters Sum of Squares) of the cluster set *CS* is calculated by Equation (6):

$$BCSS(CS) = \sum_{i=1}^{k} (m_i \times \|\overline{cp_i} - \overline{tcp}\|^2) \tag{6}$$

Assume dataset *D* is a single cluster, and calculate *WCSS*(*D*) on the basis of Equation (4). For each arbitrary set of clusters *CS* over *D*, we get it [49]:

$$WCSS(D) = WCSS(CS) + BCSS(CS) \tag{7}$$

It implies for every partition of *D,* the sum of *WCSS* and *BCSS* is constant. This method employs another theorem that is used to merge and split clusters.

Theorem 2

Let $CS=\{C_1, C_2, ..., C_k\}$ be a set of clusters over the dataset *D* and let C_i and C_j be two clusters in *CS*. Suppose CS_{ij} is obtained from the cluster set *CS* by exchanging clusters C_i and C_j with their union $\{C_i \cup C_j\}$. Therefore,

$$\Delta WCSS(CS_{ij}) = WCSS(CS_{ij}) - WCSS(CS) \tag{8}$$

The $\Delta WCSS(CS_{ij})$ can be calculated by Equation (9):

$$\Delta WCSS(CS_{ij}) = \frac{m_i \times m_j}{m_i + m_j} \|\overline{cp_i} - \overline{cp_j}\|^2 \tag{9}$$

where, m_i and m_j denote the number of points in C_i and C_j and $\overline{cp_i}$ and $\overline{cp_j}$ represent the centroid points of C_i and C_j respectively [49]. This equation is replaceable as follows:

$$\Delta WCSS(CS_{ij}) = \frac{m_i \times m_j}{m_i + m_j} \sum_{k=1}^{d} (\frac{S_{i,k}}{m_i} - \frac{S_{j,k}}{m_j})^2 \tag{10}$$

where, $S_{i,k}$ ($S_{j,k}$) is the sum of all points in cluster C_i (C_j) in k-th dimension. This equation is applied in the divisive phase. When a cluster is divided into two clusters, this results in a reduction of the *WCSS.* On the other hand, when two clusters are merged into one cluster, this causes the *WCSS* to grow. This decrease and increase can be determined by Equation (10). These definitions and theorems determine the main strategy of the CLUBS$^+$ method.

4.1.2 The Divisive Phase

In this phase, a top-down partitioning of the dataset is performed to gain rectangular blocks of points. Each block contains points similar to each other, as much as possible. This means this algorithm is attempting to reduce the *WCSS.* This purpose is also followed by other clustering algorithms such as k-means.

Identifying a partition that minimizes the *WCSS* even for two-dimensional points is an NP-hard problem [53]. A greedy algorithm is applied in the CLUBS$^+$ method. In each step, each block is divided into a pair of blocks to improve the specific clustering criteria.

Given *D* as an input dataset of *n d*-dimensional points. *D* is regarded as a single block, entered into a priority queue, *Q*. A block is removed from *Q* and split into a pair of blocks, while *Q* is not empty. The new blocks replace the previous one in *Q* if the partition is successful, otherwise, the block will be identified as the final block.

Blocks in *Q* are sorted by their *WCSS* in descending order because blocks with larger *WCSS* have higher splitting priorities. Furfaro et al. demonstrated that as a greedy algorithm is used for hierarchical clustering to minimize the overall variance with a limited number of clusters, the selection of the cluster with the largest variance leads to the best results for each iteration [54]. Therefore, the block with the largest *WCSS* is evaluated for splitting at each step. Two critical points should be considered for producing efficient splits: (1) calculating the best split and (2) assessing the efficiency of the split.

The calculating of the best split

To find the best split for a block, *WCSS* should be minimized. In other words, the aim is to partition a block into a pair of blocks to maximize the $\Delta WCSS$ (Equation (9)). Blocks are partitioned using hyperplanes that are orthogonal to some dimension. Each split is identified by a pair <*dim*, *value*> where the separating dimension is *dim* and the separating position is *value*. The points of block *B* are partitioned into a pair of blocks $\{B_1, B_2\}$ by split <*dim*, *value*>, where B_1 contains the points of *B* whose *dim*-th coordinate is less than or equal to the *value* and B_2 contains the remaining points.

It is essential to measure cumulative marginal distributions on each dimension before splitting a block. Assume $B = \{p_1, p_2, ..., p_n\}$ is a set of n *d*-dimensional points and p_{ij} is *j*-th coordinate of p_i. The marginal sum of the *j*-th dimension of B is defined as Equation (11):

$$ms_j: R \rightarrow R^d, ms_j(v) = \sum_{p_i \in B \ \wedge p_{ij} = v} p_i \quad (11)$$

The marginal count of the j-th dimension of B can be calculated by Equation (12):

$$mc_j: R \rightarrow N,\ mc_j(v) = \left|\{p_i \mid p_i \in B \bigwedge p_{ij} = v\}\right| \quad (12)$$

Therefore, the cumulative marginal distribution at coordinate v of dimension i is the sum (count) of points whose coordinates in the i-th dimension are less than or equal to v:

$$cms_j(v) = \sum_{p_i \in B \ \wedge p_{ij} \leq v} p_i\ , \quad (13)$$

and

$$cmc_j(v) = \left|\{p_i \mid p_i \in B \wedge p_{ij} \leq v\}\right| \quad (14)$$

These functions can be represented as arrays and determined by a linear scanning of the data in B. It is possible to calculate $\Delta WCSS$ in constant time as the cumulative marginals are precomputed. To determine the best split, it is important to find a pair <dim, value> that will maximize $\Delta WCSS$.

Evaluating of the effectiveness of the split

One essential problem in any divisive algorithm is deciding when to prevent the further division of the current clusters. The reason is that the additional division may not improve the overall quality of the current cluster set. There are many criteria in the literature that estimate the quality of the current cluster set [55]. In this paper, the CH-index is identified as the most appropriate criterion [56].

The *CH-index* for a set of k clusters $CS=\{C_1, C_2, ..., C_k\}$ over a dataset D, with $|D|=n$ is defined as Equation (15):

$$CH - index(CS) = \frac{BCSS(CS)}{WCSS(CS)} \times \frac{n-k}{k-1} \quad (15)$$

The new *CH-index* is calculated after each step. If the split increases the new *CH-index*, the split is effective and the divisive phase continues. Otherwise, a "local" criterion is checked according to the existence of a "valley" in the marginal distribution. Even though a split could not increase the *CH-index*, a very large local discontinuity could justify the split of a block anyway. The divisive phase is shown in Algorithm 1. The algorithm for this phase consists of two sub-algorithms that determine the best split and effectiveness of the split.

Algorithm 1 Divisive Phase

```
Input: $D = \{p_1, p_2, \ldots, p_n\}$, a dataset and a set of $n$ $d$-dimensional points.
Output: $B = \{B_1, B_2, ..., B_t\}$, a partitioning of $D$ into $t$ blocks.
1: Let $Q=\emptyset$ be a priority queue and $B=\emptyset$.
2: Let $D$ be a single block and is entered into $Q$.
3: Set $t=1$, $max\_CH=0$;
4: Set $wcss= WCSS(D)$;
5: Set $bcss= 0$;
6: while ($Q!=NULL$)
7:   $S \leftarrow Q$.deleteQueue();
8:   Compute ($dim$, $value$, $\Delta WCSS$) for the best split of $S$.
     //The $dim$ is the splitting dimension and the $value$ is the
     splitting position
    // Compute $\Delta WCSS$ based on Equation(10)
9:   If split is effective then
10:    $\{S1, S2\} \leftarrow$ make split ($S$);
11:    $Q$.addQueue($S1$);
12:    $Q$.addQueue($S2$);
13:    $wcss= wcss - \Delta WCSS;$
14:    $bcss= bcss + \Delta WCSS;$
15:    $t=t+1$;
16:    $CH = \frac{bcss}{wcss} \times \frac{n-t}{t-1}$;
17:    If $CH > max\_CH$ then
18:      $max\_CH = CH;$
19:    end if
20:  else
21:    $B \leftarrow B \cup \{S\}$;
22:  end if
23: end while
24: return $B$
```

4.1.3 The Intermediate Refinement Phase

The overall dataset was partitioned in several blocks in the previous phase. However, some blocks contain outliers or noise points. The aim of the intermediate refinement phase is (1) to recognize and separate blocks containing only noise points, and (2) to create well-rounded clusters for the remaining blocks.

The main idea is the low density of blocks including only outliers. Additionally, due to the random nature of the outliers, the density of these blocks is uniform. First, the blocks are sorted by their densities in ascending order (As shown in Algorithm 2). Then, changes in density between adjacent blocks are observed and the first jump determines the candidates for outlier-blocks. In the next step, the blocks that are considered to be outlier should be examined. For this reason, each candidate block is considered to have a small hypercube around the centroid point. When the hypercube has significantly less density than the whole block, the candidate block is identified as an outlier block otherwise it is known as a cluster block.

Following the determination of the cluster blocks and outlier blocks, the second target of this phase is pursued. To this end, the blocks near every block B_i in the set are found. Suppose C_k is a cluster identified from cluster set with the centroid point $\overline{cp} = (\overline{cp_1}, \overline{cp_2}, \ldots, \overline{cp_d})$. The distance between each point $p_i = \{p_{i1}, p_{i2}, \ldots, p_{id}\}$ and each cluster block C_k is calculated by Equation (16):

$$dist(p_i, C_k) = \sum_{j=1}^{d} \left(\frac{\|p_{ij} - cp_j\|}{r_i} \right)^2 \quad (16)$$

where r_i is the radius of the cluster along the i-th dimension that can be estimated by Equation (17):

$$r_i = 3 \times \sqrt{\frac{WCSS_i(C_k)}{n}} \quad (17)$$

such that n is the number of points in the cluster block C_k and $WCSS_i(C_k)$ is calculated by Equation (4) only for i-th coordinates of the points in C_k [57]. It is notable that if $dist(p_i, C_k)$=1, there is an ellipsoid whose center is the centroid point of C_k and whose radius on the i-th dimension is r_i. Consequently, each point p_i is allocated to cluster RC_k with the lowest $dist(p_i, C_k)$, and the minimum distance is less than or equal to 1. Otherwise, p_i is classified as an outlier in RC_0.

Hence, ellipsoids are used in this method instead of spheres. It causes some dimensions to stretch or shrink to the previous phase. However, some blocks contain outliers or noise points. The aim of the intermediate adjust the data in hypercubes. The step could increase the effectiveness of the algorithm. Therefore, each point is assigned to the cluster to minimize its distance from the cluster's centroid relative to the radius of the cluster.

Algorithm 2 Intermediate Refinement Phase

```
Input: B= {B_1, B_2, ..., B_t}, a partitioning of D into t blocks.
Output: RC={RC_0, RC_1, ..., RC_u}, a clustering of the points
  where RC_0 is the only block that contains outliers .
1: Let density = {d_1, d_2, ..., d_t} be the densities of blocks in B.
2: Let centroid_density={cd_1, cd_2, ..., cd_t} be the densities of
  small hypercubes around the centroid points of blocks.
3: density1 ← Sort (density) // in ascending order.
4: for i=1; i ≤ t-1; i++ do
5:   Jump[i]= density1[i+1]/ density1[i];
6: end for
7: Set sum=0;
```

Algorithm 2 Intermediate Refinement Phase

```
8: for i=1; i ≤ t-1; i++ do
9:   sum= sum + Jump[i];
10: end for
11: avg= sum / (t-1);
12: Set sw= 0;
13: for i=1; i ≤ t-1 && sw==0; i++ do
14:   If Jump[i] > avg then
15:     MDO= density1[i];  // maximum density of the outlier
          block
16:     sw= 1;
17:   end if
18: end for
19: for i=1; i ≤ t; i++ do
20:   If density[i] ≤ MDO &&
        centroid_density[i] < 2* density[i] then
21:     outlier ← outlier ∪ { B_i };
22:   else
23:     cluster ← cluster ∪ { B_i };
24:   end if
25: end for
26: for each B_i ∈ B do
27:   NB ← near_blocks(B_i, cluster)  // find indexes of near
                                      // blocks to B_i in cluster.
28:   for each p_j ∈ B_i do
29:     Set c=0, d_min=1;
30:     for each k ∈ NB do
31:       If dist(p_j, C_k) ≤ d_min then
32:         d_min= dist(p_j, C_k);
33:         c=k;
34:       end if
35:     end for
36:     RC_c ← RC_c ∪ { p_j };
37:   end for
38: end for
39: return RC
```

4.1.4 The Agglomerative Phase

The main goal of this phase is to enhance the overall quality of the clustering. Some clusters that were created in the previous phase are merged in this phase. A pair of clusters are evaluated for merging at each stage. The process of merging is conducted provided it leads to the growth of the *CH-index* and the least increase of *WCSS*. This means that they will be replaced by the union of the two clusters. If the merge of these two clusters leads to a decrease in the *CH-index,* the merge is not done.

There is no limit to selecting which clusters to merge, as opposed to the divisive phase. In the divisive phase, each rectangular block is divided into two rectangular parts. In the agglomerative phase, there is no constraint on the shape of the resulting cluster from the merging process. This phase is very quick, as it is precomputed based on prior information and does not require data access.

4.1.5 The Final Refinement Phase

Some clusters have an irregular shape that is produced in the previous phase. On the other side, the final outliers must be identified. The main goal of the final refinement phase is to enhance the quality of the clusters and identify the final outliers. The final refinement phase is close to the end part of the intermediate refinement phase. Algorithm 3 shows the final refinement phase that the final set of clusters and the final outliers are its output. The complexity of the CLUBS$^+$ algorithm is $O(n \times k)$ where n is the number of the points and k is the number of clusters that were found following the divisive phase [52].

Algorithm 3 Final Refinement Phase

```
Input: RC={$RC_0$, $RC_1$, ..., $RC_u$}, a clustering of the points where $RC_0$ is the only cluster that contains outliers.
Output: C={$C_0$, $C_1$, ..., $C_u$}, a clustering of the points where $C_0$ is the only cluster containing outliers.
1: cluster ← {$RC_1$, $RC_2$, ..., $RC_u$};
2: for each $B_i \in RC$ do
3:   NB ← near_blocks($B_i$, cluster);  // find indexes of near clusters to $B_i$ in cluster.
4:   for each $p_j \in B_i$ do
5:     Set c=0, $d_{min}$=1;
6:     for each $k \in NB$ do
7:       If $dist(p_j, C_k) \le d_{min}$ then
8:         $d_{min} = dist(p_j, C_k)$;
9:         c=k;
10:      end if
11:    end for
12:    $C_c \leftarrow C_c \cup \{p_j\}$;
13:  end for
14: end for
15: return C
```

4.2 The Parallelization of CLUBS$^+$

As stated earlier, the MapReduce framework is used to parallelize our algorithms. The basic idea is to split the dataset on a large scale into small datasets. The small data sets are then distributed to the nodes (workers) for calculations. The nodes (workers) are coordinated by a master node. The datasets are processed by Map function and Reduce function, and the final results are then obtained.

Analysis of the CLUBS$^+$ method steps is essential to parallel its execution. Some operations require entire access to the data. Therefore, the most important operations are checked to parallel them in the four phases of the method.

4.2.1 The Parallelization of Divisive Phase

The divisive phase consists of two sub-algorithms that find the best split and evaluate the effectiveness of the split. For these objectives, the measurement of the cumulative marginals is required in this phase. This calculation requires access to the entire dataset. Assume input dataset D is loaded into a distributed file system (HDFS) by the data fragments called splits. Each split S_i is sent to a mapper for processing. Therefore, data set $D\{S_1, S_2, ..., S_t\}$ is sent to the mappers. For each dimension, the vectors ms and mc (Equation (11) and Equation (12)) can be calculated independently and in parallel on each split S_i. Then, the final vectors ms and mc can be acquired by summing and counting the obtained vectors on all S_i. The reducers are getting the final results according to the MapReduce paradigm. Other operations can be handled by the master node.

4.2.2 The Parallelization of the Refinement Phases

The density of the blocks has to be measured in the intermediate phase to assess the outlier blocks. After the divisive phase, the master node obtains a list of the blocks with their range and density. Furthermore, the density of each block around the centroid is required to find out the outlier blocks. For this purpose, a small range is deemed for each block around the centroid. Then, the number of points that fall in each of these specified areas is counted in parallel. The only mathematical operation involved in these phases is the sum, which is completely parallelizable. For every part of a block's data, the number of points falling within a specified range can be counted independently. Then, the total count is obtained by the sum of partial counts. As described before, this count permits the master node to identify the outlier blocks.

Another important operation performed during both refinement phases is the assignment of points to the clusters, which can be performed independently on each worker node. Cluster information is in the master node that has to be sent to all the worker nodes. Then, each worker node will measure the distance between points and clusters and allocate points to the appropriate cluster. After the assignment of the points, some information must be updated by the master node. To this end, each worker node computes certain information for the master node, such as the new centroid, the number of points, and the new radius. The master node would then get updated information about clusters by aggregating all the values from the worker nodes.

4.2.3 The Parallelization of the Agglomerative Phase

In this phase, some clusters are merged to improve the overall effectiveness of the clustering. For this purpose, some precomputed prior information about clusters is required and does not require access to the entire data. The master node has the required information and the operation can be performed by a worker node.

Four phases of the CLUBS$^+$ method have been reviewed for parallelization. The parallelization of this method is used to cluster the training dataset for the next step of the proposed k-Nearest Neighbor classification.

5. Testing Process

Assume that the parallel CLUBS$^+$ algorithm produces u clusters and cluster centers, then the cluster with the nearest cluster center is found for each test item. The cluster found is being used as the new training dataset for the test data item. The k-Nearest Neighbor algorithm is used to classify the test data item into the new training dataset. Due to the selection of a cluster with high similarity to the test data item, the proposed approach also has relatively high classification accuracy. The proposed method (KNN-CCL) is represented in the Algorithm 4.

Algorithm 4 KNN-CCL algorithm

```
Input: $X=\{x_1, x_2, ..., x_n\}$, the training dataset.
    $Y=\{y_1, y_2, ..., y_m\}$, the test dataset.
Output: $CL=\{cl_1, cl_2, ..., cl_m\}$, the class label of each test data item.
1: Produce $u$ clusters by Algorithm 1 to Algorithm 3.
2:  $C=\{C_1, C_2, ..., C_u\}$;
3: for each $y_i \in Y$ do
4:  Set $min$=0, $d_{min}=M$; // $M$ is a very big value
5:  for each $C_j \in C$ do
4:     Compute distance $dist\ (y_i, C_j)$ between $y_i$ and the cluster center of $C_j$.
5:    If $dist(y_i, C_j) \le d_{min}$ then
6:       $d_{min}= dist(y_i, C_j)$;
       $min=j$;
5:    end if
6:  end for
6:  Use $C_{min}$ as new training dataset for $y_i$.
7:  Find k-nearest neighbor for $y_i$ in $C_{min}$.
8:  $knn_i=\{nn_1, nn_2, ..., nn_k\}$;
```

Algorithm 4 KNN-CCL algorithm

```
8:  Predict the class label $cl_i$ for $y_i$ based on $knn_i$.
9:  $CL \leftarrow CL \cup \{cl_i\}$;
10: end for
11: return $CL$
```

The size of the new training dataset in this method is much smaller than the size of the original training dataset. Hence, the calculation of the k-Nearest Neighbor algorithm is reduced, and the KNN-CCL algorithm improves the classification quality. Furthermore, some previous methods require the number of clusters or the termination condition to be obtained. The CLUBS$^+$ algorithm requires none of them and the aforementioned issues can not affect the overhead of clustering and classification accuracy.

6. Experiments

This section describes the factors and points related to the experimental study. It also analyzes the results obtained from various experimenal studies.

6.1 Experimental Framework

The following measures are considered in this paper for evaluating the performance of the method proposed:

- Classification accuracy: this represents the number of correct classification with respect to the total number of instances.
- Execution time: the total execution time spent by classifiers in the classification of a given test set against a training dataset shall be collected. It includes reading and distributing all data and all computations carried out using comparative methods.

As described in previous sections, the proposed method KNN-CCL uses the parallel CLUBS$^+$ algorithm to partition the training dataset. Then, it will provide the appropriate cluster for each test data item as a new training dataset. Ultimately, it uses the k-Nearest Neighbor classification to predict each test data item in the new training dataset. The whole training dataset is not used in experiments. A random sampling method [21] is applied to sample 20% of the training dataset. We took the classification of k-Nearest Neighbor (KNN) as a baseline to demonstrate the effectiveness of the KNN-CCL algorithm and made a comparison between KNN, KNN-CCL, and LC-KNN [33]. LC-KNN uses k-means clustering to separate the training dataset. We used a similar random sampling approach before the clustering process to sample 20% of the training dataset. We also repeat the k-means algorithm 10 times and use the cluster centers afterward. Each experimental group is repeated three times, and the average values are reported for a more reliable result.

Three real datasets will be used for the experimental study. Pendigits, Letter, and Satimage are extracted from the UCI machine learning repository [58]. The Pendigits is a dataset about pen-based recognition of handwritten digits. This includes 10992 instances, 16 features, and 10 classes. The Letter is a letter recognition dataset. It contains a total of 20000 instances, 16 features, and 26 separate classes (capital letters in the English alphabet). The Satimage is a dataset of satellite images. This involves 6435 instances, 36 features, and 7 classes.

All experiments were carried out on the same environment, an Intel Xeon E5-2680v2 with a total of 10 cores (20 threads) and 128 GB of memory. The experiments were executed using Spark 1.6.1 and Scala 2.10.5.

6.2 Experimental Results

This section describes and discusses the experimental results. First, we consider the number of clusters as a cluster to compare the performance of the algorithms in terms of classification. Therefore, we set k=1 and make use of classification accuracy and execution time as evaluations of the classification tasks. Our experimental results are reported in Table 1. Table 1 demonstrates that the proposed KNN-CCL method has improved 5~9 times over the KNN, and 2~4 times over the LC-KNN in terms of execution time. This means that the proposed distributed solution reduces the execution time of the proposed algorithm compared to the other two algorithms. In classification accuracy evaluation, the KNN-CCL and LC-KNN are 4%~10% lower and 2%~11% lower than KNN. Although LC-KNN uses a serial algorithm and algorithm KNN-CCL uses a parallel algorithm and distributes data between clusters, algorithm KNN-CCL can be equivalent or even better than algorithm LC-KNN in terms of accuracy evalution. Therefore, the KNN-CCL performs well in terms of classification accuracy and execution time with *k*=1 according to experimental results.

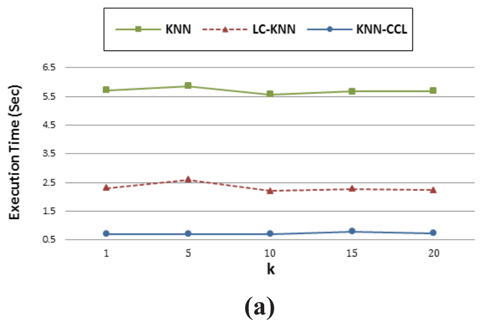
A group of experiments on three datasets is performed by selecting different *k* values for KNN, KNN-CCL, and LC-KNN. For this group of experiments, for particular, three methods were conducted on the datasets with *k*=1, 5, 10, 15, and 20, respectively. Comparison of the execution time of three algorithms KNN, KNN-CCL, and LC-KNN with different *k* values is shown in Figure 4.

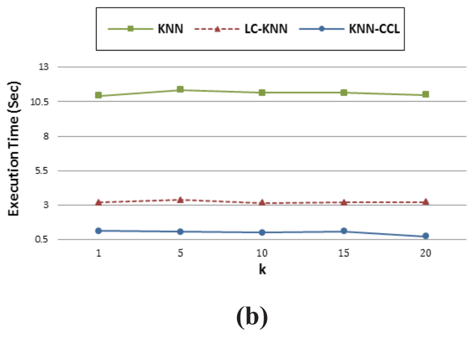
From Figure 4, we observed that the execution times for three algorithms are linear with varying *k* values. In other words, for three algorithms, greater *k* values slightly increase the execution time over different datasets. As concluded from Figure 4, despite the increase in the number of clusters, the execution times of the algorithms will not differ much. The proposed algorithm KNN-CCL requires less execution time in all cases. The reason for this is that the KNN-CCL uses a parallel centroid-based and hierarchical clustering method to provide the appropriate training dataset for each test data item. The LC-KNN method uses the k-means algorithm which requires it to repeat the k-means algorithm by 10 times to obtain the appropriate training dataset. Thus, The LC-KNN method takes more execution time than the KNN-CCL method. On the other hand, this strategy (KNN-CCL) takes advantage of parallel execution to speed up computation.

Figure 5 illustrates the classification accuracy evaluation of the three algorithms KNN, KNN-CCL, and LC-KNN over three datasets with various *k* values. In Figure 5(a), the classification accuracy of KNN-CCL for the Pendigits dataset is 5%~12% lower than KNN and the classification accuracy of LC-KNN is 2%~10% lower than KNN. KNN-CCL and LC-KNN got almost the same results in most situations. The classification accuracy of three methods over the Letter dataset is shown in Figure 5(b). The KNN-CCL is 9%~22% lower than the KNN and LC-KNN is 11%~29% lower than the KNN. Obviously, the KNN-CCL algorithm provides better results than the LC-KNN. In Figure 5(c), the classification accuracy of both KNN-CCL and LC-KNN is 2%~9% lower than KNN over the Satimage dataset. KNN-CCL and LC-KNN obtained almost the same outcomes for the majority of cases. We observed, from Figure 5, that the classification accuracy of the proposed method KNN-CCL and LC-KNN were approximately equivalent.

Table 1. classification accuracy and execution time (seconds) of three algorithms on three datasets.

Dataset	KNN-CCL		LC-KNN		KNN	
	Accuracy	Time	Accuracy	Time	Accuracy	Time
Pendigits	0.927	0.7	0.9537	2.3	0.9774	5.71
Letter	0.8602	1.14	0.8459	3.2	0.9565	10.94
Satimage	0.8501	0.31	0.8667	1.28	0.8944	1.62





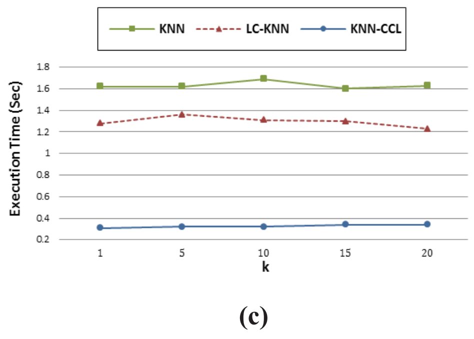
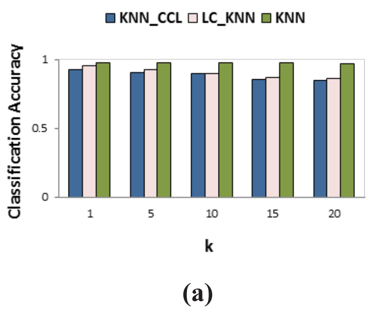
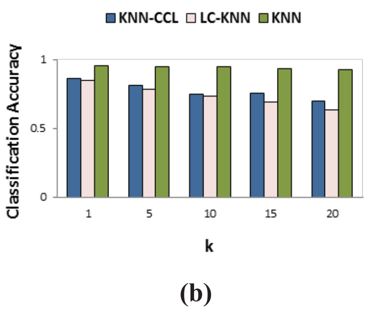


Figure 4. Execution time on three datasets at different values of k: **a** Pendigits **b** Letter **c** Satimage.





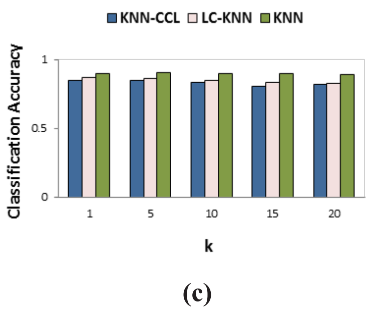
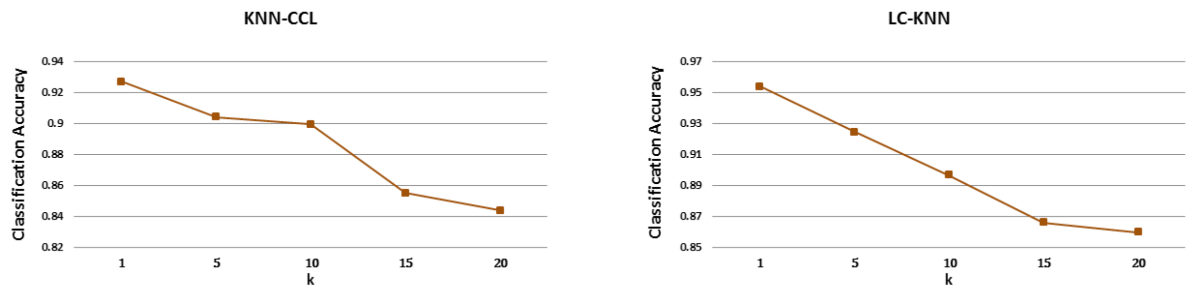


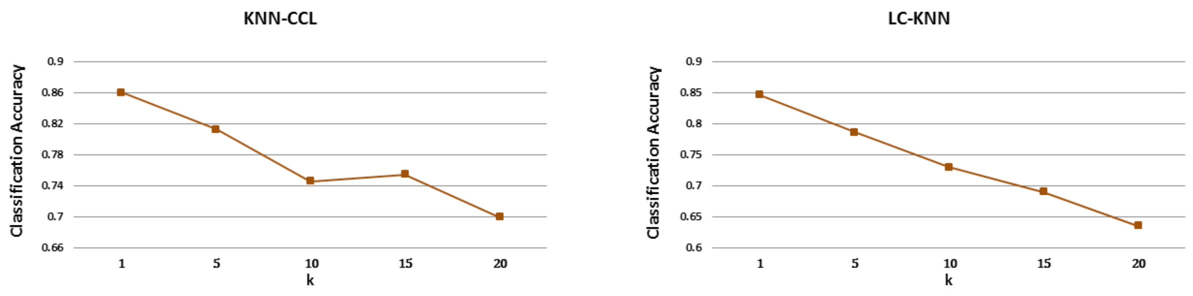
Figure 5. Classification accuracy on three datasets with varying *k* values: **a** Pendigits **b** Letter **c** Satimage.

In the end, we are going to study the classification accuracy of two KNN-CCL and LC-KNN algorithms to classification accuracy of two algorithms over three datasets to *k*=1, 5, 10, 15, and 20. As the value of *k* increases, the overall accuracy of classification decreases. This is because the smaller the training dataset would be, the greater the classification accuracy. Therefore, the difference between the samples is important, and the accuracy of classification will be reduced. In this case, it can be concluded that an acceptable *k* value should be selected. As shown in Figure 6, when two algorithms are more accurate, the *k* value should be set as small as possible. According to this analysis, if higher accuracy is required, a smaller number of clusters should be selected.

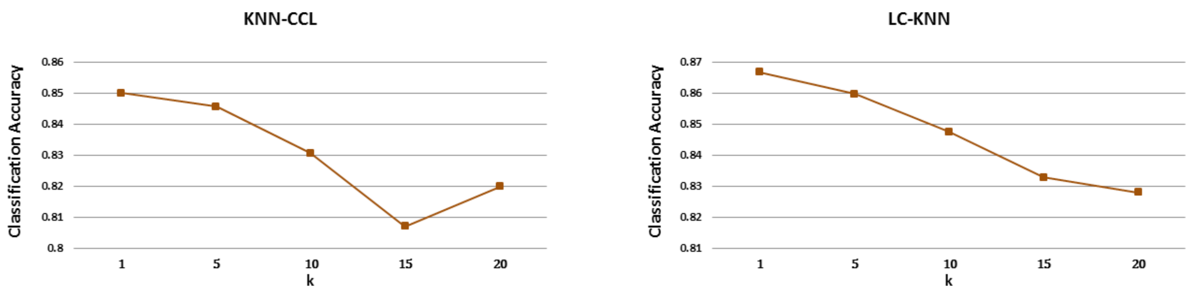
Therefore, according to experimental results, we can infer that the KNN-CCL and LC-KNN algorithms perform well with small values of *k* in terms of classification accuracy, and the proposed method KNN-CCL performs very well in terms of execution time.



(a) Pendigits dataset.



(b) Letter dataset.



(c) Satimage dataset.

Figure 6. Classification accuracy of KNN-CCL and LC-KNN on three datasets with different k.

7. Conclusions

In this paper, an efficient parallel processing of the k-Nearest Neighbor classification method was presented and evaluated. The proposed method is denominated as KNN-CCL uses a parallel centroid-based and hierarchical clustering algorithm to split the sample set of training dataset into several clusters. The introduced clustering algorithm uses four stages of successive refinements and generates high quality clusters. And the suitable cluster was selected as new training dataset for each test data item to decrease the calculation of classification. The k-Nearest Neighbor classification was used for each test data item to predict it in the new training dataset.

The KNN approach was considered the baseline, and a group of comparative experiments was performed by KNN, LC-KNN, and KNN-CCL. The experimental results carried out revealed that the proposed method can handle large dataset and performs well in terms of accuracy and performance in classification.

Conflict of Interest

There is no conflict of interest.

Funding

The authors received no specific funding for this work.

Ethical approval

This article does not contain any studies with human participants or animals performed by any of the authors.

References

[1] Alharthi, A., Krotov, V., Bowman, M., 2017. Addressing barriers to big data. Business Horizons. 60(3), 285-292.

[2] Anagnostopoulos, I., Zeadally, S., Exposito, E., 2016. Handling big data: research challenges and future directions. The Journal of Supercomputing. 72(4), 1494-1516.

[3] Akoka, J., Comyn-Wattiau, I., Laoufi, N., 2017. Research on Big Data – A systematic mapping study. Computer Standards & Interfaces. 54, 105-115.

[4] Minelli, M., Chambers, M., Dhiraj, A., 2013. Big data, big analytics: emerging business intelligence and analytic trends for today's businesses. John Wiley & Sons. 578.

[5] Cover, T., Hart, P., 1967. Nearest neighbor pattern classification. IEEE transactions on information theory. 13(1), 21-27.

[6] Wu, X., Kumar, V., 2009. The top ten algorithms in data mining. CRC press.

[7] Chen, Y.H., Garcia, E.K., Gupta, M.R., et al., 2009. Similarity-based classification: Concepts and algorithms. Journal of Machine Learning Research. 10, 747-776.

[8] Weinberger, K.Q., Saul, L.K., 2009. Distance metric learning for large margin nearest neighbor classification. Journal of Machine Learning Research. 10, 207-244.

[9] Li, J., Liu, Y., Pan, J., et al., 2020. Map-Balance-Reduce: An improved parallel programming model for load balancing of MapReduce. Future Generation Computer Systems. 105, 993-1001.

[10] Dean, J., Ghemawat, S., 2008. MapReduce: simplified data processing on large clusters. Communications of the Acm. 51(1), 107-113.

[11] White, T., 2012. Hadoop: The definitive guide. O'Reilly Media, Inc.

[12] Chen, C.P., Zhang, C.Y., 2014. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. Information Sciences. 275, 314-347.

[13] Guo, Z., Fox, G., Zhou, M., 2012. Investigation of data locality in mapreduce. Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012). 419-426. IEEE.

[14] Zaharia, M., Chowdhury, M., Das, T., et al., 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI 12). 15-28.

[15] Bei, Zh.D., Yu, Zh.B., Luo, N., et al., 2018. Configuring in-memory cluster computing using random forest. Future Generation Computer Systems. 79, 1-15.

[16] Tang, Zh., Zhang, X.Sh., Li, K.L., et al., 2018. An intermediate data placement algorithm for load balancing in Spark computing environment. Future Generation Computer Systems. 78, 287-301.

[17] Gonzalez-Lopez, J., Cano, A., Ventura, S., 2017. Large-scale multi-label ensemble learning on Spark. 2017 IEEE Trustcom/BigDataSE/ICESS, 893-900. DOI: https://doi.org/10.1109/Trustcom/BigDataSE/ICESS.2017.328

[18] Harnie, D., Saey, M., Vapirev, A.E., et al., 2017. Scaling machine learning for target prediction in drug discovery using Apache Spark. Future Generation Computer Systems. 67, 409-417.

[19] Hernández, Á.B., Perez, M.S., Gupta, S., et al., 2018. Using machine learning to optimize parallelism in big data applications. Future Generation Computer Systems. 86, 1076-1092.

[20] Singh, H., Bawa, S., 2017. A MapReduce-based scalable discovery and indexing of structured big data. Future generation computer systems. 73, 32-43.

[21] Gavagsaz, E., Rezaee, A., Javadi, H.H.S., 2018. Load balancing in reducers for skewed data in MapReduce systems by using scalable simple random sampling. The Journal of Supercomputing. 74(7), 3415-3440.

[22] Gavagsaz, E., Rezaee, A., Javadi, H.H.S., 2019. Load balancing in join algorithms for skewed data in MapReduce systems. The Journal of Supercomputing. 75(1), 228-254.

[23] Bhatia, N., 2010. Survey of nearest neighbor techniques. arXiv preprint arXiv:1007.0085.

[24] Zhu, X., Zhang, L., Huang, Z., 2014. A sparse embedding and least variance encoding approach to hashing. IEEE transactions on image processing. 23(9), 3737-3750.

[25] Zhu, X., Zhang, S., Zhi, J., et al., 2010. Missing value estimation for mixed-attribute data sets. IEEE Transactions on Knowledge and Data Engineering. 23(1), 110-121.

[26] Connor, M., Kumar, P., 2010. Fast construction of k-nearest neighbor graphs for point clouds. IEEE

transactions on visualization and computer graphics. 16(4), 599-608.

[27] Liu, T., Moore, A.W., Gray, A., et al., 2004. An investigation of practical approximate nearest neighbor algorithms. Advances in neural information processing systems. 17.

[28] Raginsky, M., Lazebnik, S., 2009. Locality-sensitive binary codes from shift-invariant kernels. Advances in neural information processing systems. 22.

[29] Silpa-Anan, C., Hartley, R., 2008. Optimised KD-trees for fast image descriptor matching. 2008 IEEE Conference on Computer Vision and Pattern Recognition. 1-8. IEEE.

[30] Zhu, X.F., Huang, Z., Cheng, H., et al., 2013. Sparse hashing for fast multimedia search. ACM Transactions on Information Systems (TOIS). 31(2), 9.

[31] Triguero, I., Peralta, D., Bacardit, J., et al., 2015. MRPR: A MapReduce solution for prototype reduction in big data classification. neurocomputing. 150, 331-345.

[32] Du, M., Ding, S., Jia, H., 2016. Study on density peaks clustering based on k-nearest neighbors and principal component analysis. Knowledge-Based Systems. 99, 135-145.

[33] Deng, Zh.Y., Zhu, X.Sh., Cheng, D.B., et al., 2016. Efficient kNN classification algorithm for big data. Neurocomputing. 195, 143-148.

[34] Moutafis, P., Mavrommatis, G., Vassilakopoulos, M., et al., 2019. Efficient processing of all-k-nearest-neighbor queries in the MapReduce programming framework. Data & Knowledge Engineering. 121, 42-70.

[35] Zhang, C., Li, F., Jestes, J., 2012. Efficient parallel kNN joins for large data in MapReduce. Proceedings of the 15th international conference on extending database technology. 38-49.

[36] Chatzimilioudis, G., Costa, C., Zeinalipour-Yazti, D., et al., 2015. Distributed in-memory processing of all k nearest neighbor queries. IEEE Transactions on Knowledge and Data Engineering. 28(4), 925-938.

[37] Sun, K., Kang, H., Park, H.H., 2015. Tagging and classifying facial images in cloud environments based on KNN using MapReduce. Optik. 126(21), 3227-3233.

[38] Maillo, J., Triguero, I., Herrera, F., 2015. A mapreduce-based k-nearest neighbor approach for big data classification. 2015 IEEE Trustcom/BigDataSE/ISPA. 2, 167-172. IEEE.

[39] Maillo, J., Ramírez, S., Triguero, I., et al., 2017. kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for big data. Knowledge-Based Systems. 117, 3-15.

[40] Wu, X., Zhang, C., Zhang, S., 2005. Database classification for multi-database mining. Information Systems. 30(1), 71-88.

[41] Ester, M., Kriegel, H.P., Sander, J., et al., 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. Kdd. 96(24),226-231.

[42] Sander, J., 2010. Density-Based Clustering, in Encyclopedia of Machine Learning, C. Sammut and G.I. Webb, Editors. Springer US: Boston, MA. pp. 270-273.

[43] Amini, A., Wah, T.Y., Saybani, M.R., et al., 2011. A study of density-grid based clustering algorithms on data streams. 2011 Eighth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD). 3, 1652-1656. IEEE.

[44] Wang, W., Yang, J., Muntz, R., 1997. STING: A statistical information grid approach to spatial data mining. Vldb. 97, 186-195.

[45] Gerlhof, C.A., Kemper, A., 1993. Partition-based clustering in object bases: From theory to practice. International Conference on Foundations of Data Organization and Algorithms. Springer, Berlin, Heidelberg. 301-316.

[46] MacQueen, J., 1967. Some methods for classification and analysis of multivariate observations. Proceedings of the fifth Berkeley symposium on mathematical statistics and probability. 1(14), 281-297.

[47] Johnson, S.C., 1967. Hierarchical clustering schemes. Psychometrika. 32(3), 241-254.

[48] Zhang, T., Ramakrishnan, R., Livny, M., 1996. BIRCH: an efficient data clustering method for very large databases. ACM sigmod record, 25(2), 103-114.

[49] Mazzeo, G.M., Masciari, E., Zaniolo, C., 2017. A fast and accurate algorithm for unsupervised clustering around centroids. Information Sciences. 400, 63-90.

[50] Masciari, E., Mazzeo, G.M., Zaniolo, C., 2013. Pacific-asia conference on knowledge discovery and data mining. Springer, Berlin, Heidelberg. 111-122.

[51] Masciari, E., Mazzeo, G.M., Zaniolo, C., 2014. Analysing microarray expression data through effective clustering. Information Sciences. 262, 32-45.

[52] Ianni, M., Masciari, E., Mazzeo, G.M., et al., 2020. Fast and effective Big Data exploration by clustering. Future Generation Computer Systems. 102, 84-94.

[53] Muthukrishnan, S., Poosala, V., Suel, T., 1999. On rectangular partitionings in two dimensions: Algorithms, complexity and applications. International Conference on Database Theory. Springer, Berlin, Heidelberg. 236-256.

[54] Furfaro, F., Mazzeo, G.M., Saccà, D., et al., 2008. Compressed hierarchical binary histograms for summarizing multi-dimensional data. Knowledge and Information Systems. 15(3), 335-380.

[55] Arbelaitz, O., Gurrutxaga, I., Muguerza, J., et al., 2013. An extensive comparative study of cluster validity indices. Pattern Recognition. 46(1), 243-256.

[56] Caliński, T., Harabasz, J., 1974. A dendrite method for cluster analysis. Communications in Statistics-theory and Methods. 3(1), 1-27.

[57] Zhang, T., Ramakrishnan, R., Livny, M. ,1996. BIRCH: an efficient data clustering method for very large databases. ACM sigmod record, 25(2), 103-114.

[58] Dua, D., Graff, C., 2017. Machine learning repository. University of California, Irvine, School of Information and Computer Sciences. Available online at: http://archive. ics. uci. edu/ml.