Design of Fast and Scalable Clustering Algorithm on Spark

Aakash Raj Pokhrel
Department of Computer Science
Lamar University
Beaumont, TX USA
apokhrel1@lamar.edu

Sujing Wang
Department of Computer Science
Lamar University
Beaumont, TX USA
sujing.wang@lamar.edu

ABSTRACT
Clustering is a popular unsupervised data mining technique. It has been applied in various data mining and big data applications. Efficient clustering algorithms and implementation techniques are keys to cope with the scalability and performance requirements of big data analysis. This paper introduces the design and implementation of a density-based clustering algorithm that can deal with big data efficiently and effectively. We present a parallel Shared Nearest Neighbor (SNN) clustering algorithm using the k-dimensional tree (k-d tree) to reduce search time to improve efficiency. The proposed algorithm is implemented in a distributed environment using the Spark framework. The effectiveness of the proposed algorithm has been evaluated through a case study involving four data sets, Bristol Crime Stats, 911 call, Complex9, and TLC Trip datasets.

CCS Concepts
• Information systems → Information systems applications → Data mining → Clustering

Keywords
Clustering; Shared Nearest Neighbor Clustering; K-dimensional Tree; Spark Platform.

1. INTRODUCTION
Clustering is one of the popular data mining techniques that can be applied to many application domains, such as finance, biology, political science, sports, etc. Shared Nearest Neighbor (SNN) [1] is a well-known density-based clustering algorithm that can handle high dimensional data and find clusters of different densities, sizes, and shapes. However, as the volume of the available data becomes extremely large, traditional techniques running on a single machine are inefficient in analyzing big data. Therefore, new techniques are needed to address these challenges and to provide efficient solutions to analyze this wealth of big data. Many researchers have extended traditional clustering algorithms such as K-means [2], DBSCAN [3], and SNN [1] on high-performance computer clusters for big data analysis. Meanwhile, different tree-based data structures, such as k-dimensional tree (k-d tree) [4] and R-tree [5], have been implemented to improve the performance of clustering big data. In this paper, we propose a methodology to improve the traditional Shared Nearest Neighbor clustering algorithm by utilizing the powerful Spark platform [6] and k-dimensional (k-d) tree [4] for big data analysis, called Spark-KDT-SNN. The traditional SNN has time and space complexity as O(n2) and O(nk), respectively, where n is the total number of objects in the dataset, and k is the number of nearest neighbors [1]. When utilizing k-d tree to perform the k nearest neighbors‟ queries, the runtime complexity is reduced to O(nlogn). The main contributions of this paper are:

1: We develop and implement Spark-KDT-SNN, an efficient density-based clustering algorithm on Spark.

2: We utilize k-d tree data structure to reduce search time.

3: We evaluate the performance and scalability properties of the Spark-KDT-SNN algorithm with real datasets.

The organization of the paper is as follows. In section 2, we discuss the related works. We introduce the Spark-KDT-SNN algorithm in section 3. In section 4, we evaluate the performance of Spark-KDT-SNN with case studies on real datasets. Section 5 concludes our study and discusses potential future works.

2. LITERATURE REVIEW
Shared Nearest Neighbor (SNN) [1] is one of the density-based clustering algorithms. SNN can be applied to many application domains. SNN clusters data as DBSCAN does, except that the number of nearest neighbors that a pair of points share is used to access the similarity. In SNN, the density of a point p is defined as the sum of the similarities between p and its‟ k nearest neighbors. SNN can find clusters of different sizes, shapes, densities, and can handle noises in the dataset. There are a few works based on the classic SNN algorithm in the literature. Ye et al. [7] introduced a new definition of density that considers both the number of shared nearest neighbors and the distance between data objects to optimize the traditional SNN algorithm. Sharma et al. [8] proposed an enhancement of the SNN clustering algorithm, called fuzzy shared nearest neighbor (FSNN). FSNN has the capability of handling the data lying in the boundary regions utilizing a fuzzy concept. Singh et al. [9] introduced an incremental extension to SNN, called IncSNN-DBSCAN. IncSNN-DBSCAN can find clusters on a dataset to which frequent inserts are made. Li et al. [10] introduced a new clustering method, called SNNC, based on Shared Nearest Neighbor for hyperspectral optimal band selection.

Using k-d tree to improve the efficiency of clustering algorithms has been widely studied in the literature as well. Walter et al. [11] improved the agglomerative clustering by implementing k-d tree to find the best fit cluster for the agglomerative process. Vijayalakshmi [12] proposed the KDT-DBSCAN clustering algorithm for solving the search complexity problem using k-d

ICCBDC'20, August 26–28, 2020, Virtual, United Kingdom

ACM ISBN 978-1-4503-7538-2/20/08…$15.00
https://doi.org/10.1145/3416921.3416942

tree. Otair [13] compared k-d tree with a brute force search for spatial clustering and found that k-d tree to be the most optimal among these two for the search of a neighboring point. However, both the classic SNN and the improved versions discussed above are inherently sequential and not well-suited for processing big data that require high-performance computing infrastructure and parallel computing systems.

Parallel implementation of different clustering algorithms has been proposed in the literature. Kumari et al. [14] presented two sequential implementations of the classic SNN algorithm, *NaiveR-SNN*, and *R-SNN*, which uses R-tree for executing neighborhood queries efficiently and exploiting spatial locality to minimize memory usage. They also introduced parallel implementations of R-SNN by employing the Single Process Multiple Data (SPMD) model. Anchalia et al. [15] introduced a combiner in the mapper function to implement k-means using the MapReduce paradigm. He et al. [16] implemented an efficient DBSCAN algorithm in a 4-stage MapReduce paradigm, called MR-DBSCAN. Han et al. [17] proposed a scalable DBSCAN algorithm with the Spark framework. Wang et al. [18] designed the MapReduce-based Shared Nearest Neighbor clustering algorithm called MR-SNN. As we can see that most of the studies focus on improving the sequential SNN algorithm or adapting it to MapReduce. However, Spark has several advantages compared to MapReduce. Our work focus on improving the efficiency of the traditional SNN algorithm utilizing k-d tree structure and Spark framework for big data analysis.

3. SPARK-KDT-SNN

This section introduces the detailed algorithm design and implementation of a new parallel Shared Nearest Neighbor clustering using the big data framework Spark. Meanwhile, we apply *k-d tree* in our algorithm to reduce the search time. We first briefly introduce the traditional SNN clustering algorithm, then SNN using k-d tree, called KDT-SNN, and the parallel implementation of KDT-SNN on Spark, called Spark-KDT-SNN.

3.1 Shared Nearest Neighbor Clustering

In SNN [1], the similarity between two points p and q is the number of points they share among their k nearest neighbors as follows:

$$similarity(p,q) = |NN(p) \cap NN(q)| \quad (1)$$

where *NN(p)* is the set of k nearest neighbors of a point p. The density of a point p is defined as the sum of the similarities between p and it‟s k nearest neighbors as follows:

$$density(p) = \sum_{i=1}^{k} similarity(p, q_k) \quad (2)$$

After assessing the density of each point in the dataset D, SNN identifies all core points; all points in the dataset D that have the SNN density of at least *MinPs* and forms the clusters around the core points like DBSCAN.

$$CoreP(D) = \{p \in D | density(p) \geq MinPs\} \quad (3)$$

SNN can find clusters of different sizes, shapes, and can handle noises in the dataset. Moreover, SNN copes better with high dimensional data and deals well with datasets having varying densities. SNN does not require the number of clusters to be determined in advance.

3.2 K-D Tree

The k-dimensional tree (k-d tree) a space-partitioning data structure for organizing points in a k-dimensional space. The nearest neighbor search algorithm aims to find the point in the tree that is nearest to a given input point. This nearest neighbor search can be done efficiently by using the k-d tree properties to quickly eliminate large portions of the search space

3.3 KDT-SNN

We improved the traditional SNN clustering algorithm using *k-d tree* to index all data points to support the k nearest neighbors‟ search. The proposed algorithm, called KDT-SNN, reduces the time complexity of SNN to O($n$log$n$).

The KDT-SNN algorithm requires three parameters: *Eps*, *MinPts*, and k. k is the number of nearest neighbors. *MinPts* is used to find the core points. It is a threshold to determine whether a point is a core point. *Eps* is used as a threshold for density. It should be smaller than k. Figure 1 shows the pseudocode of the KDT-SNN clustering algorithm. KDT-SNN first arranges all the data points in a *k-d tree*, the k nearest neighbors of each point in the dataset D are found directly from the *k-d* tree build at the first step.

KDT-SNN (D, k, MinPs, Eps)

Input: dataset D, the number of the nearest neighbors k, the core point threshold *MinPs*, the similarity threshold *Eps*

Output: set of clusters C_i.

```
1.  Build k-d tree for dataset D: k-d tree(D).
2.  Mark every p in D “unchecked”
3.  for all p in D
4.      get k nearest neighbors from k-d tree (D)
5.  end for
6.  For all pair of events p and q in D
7.      Compute similarity (p,q) based on k-d
    tree(D)
8.  end for
9.  for all p in D
10.     Compute density(p)
11.     If density(p) is greater than MinPs then
12.         Mark p as „core“;
13.     end if
14. end for
15. for all core p in D
16.     If p is marked “unchecked” then
17.         Form a cluster Ci, of points which can
        be reached from core p and add the core
        points that can be reached from core p.
18. end if
19.     Mark all points in Ci as „checked“
20. end for
21. return set of clusters Ci
```

Figure 1. Pseudocode of KDT-SNN

3.4 Spark

Spark is an open-source cluster computing framework developed by AMPLab at the University of California, Berkeley in 2009 for performing big data analytics on distributed clusters. Spark was introduced to address the limitations of the MapReduce paradigm which essentially requires reading of input from the disk, mapping the input with some map function, and saving the results from the reduced operations back to the disk. Spark introduced a data structure called resilient distributed dataset (RDD) [6] which is immutable and distributed over different nodes in the cluster. Besides, Spark supports programming languages Java, Scala, R and Python, and provides ease of use for application developers.

Spark has its standalone cluster manager and supports other cluster managers such as Hadoop YARN and Apache Mesos. Spark supports Hadoop Distributed File System (HDFS), MapR File System (MapR-FS), Cassandra, OpenStack Swift, Amazon S3, Kudu, etc. Spark consists of five main components, i.e., Spark Core, Spark SQL, Spark Streaming, MLlib (Machine Learning Library), and GraphX. Spark enables in-memory computation in its restricted distributed shared memory which makes Spark faster than MapReduce. Therefore, we adopt Spark as the underlying parallel processing platform to design and implement the distributed version of KDT-SNN clustering, called Spark-KDT-SNN. We will introduce the design of the Spark-KDT-SNN clustering algorithm in section 3.5.

3.5 Spark-KDT-SNN

In this section, we present the design and parallel implementation of KDT-SNN on Spark, called Spark-KDT-SNN. Spark-KDT-SNN does spatial data partitioning using *k-d tree*. Both multimode parallelism at the node level and multicore parallelism at the core level are exploited. Spark-KDT-SNN consists of four major steps, i.e., building k-d tree, partitioning, local clustering, and global merging. In the first step, k-d tree for dataset *D* is built. In the second step, data are divided into equal partitions based on k-d tree index to distribute among all computer nodes. The objective of partitioning is to divide the input dataset *D* into smaller partitions that can be clustered on each node in parallel. In the third step, each note constructs its *k-d tree* and performs local clustering in parallel. All local clusterings are done in parallel by exploiting both multi-node and multi-core parallelism. All local clustering results are saved as intermediate results. In the fourth step, all local clustering results are then aggregated to get the global clustering. A cluster may span several partitions, resulting in multiple local clusters that belong to different partitions. Therefore, we need to merge the local clusters into global ones. To identify a cluster that spans more than one partitions, the core points identified by local KDT-SNN and their clustering ID lists should be examined. To explain the merging procedure, we first define *Eps* criteria. A pair of core points are said to satisfy *Eps* criteria if they are within *Eps* radius of each other. We randomly select a core point from each local cluster and call it representative points for that local cluster. If any pair of representative points satisfy *Eps* criteria, the two local clusters presented by these two representative points need to be merged and relabeled in the merge procedure to get the final global clustering results. The formal description of Spark-KDT-SNN is as follow:

Step 1: build the k-d tree for dataset D, KDTree(D).

Step 2 Participation: divide the input dataset D into N partitions, S_1, S_2,…, S_n based on the k-d tree such that $D = \bigcup_{i=1}^{N} S_i$ and $S_i \cap S_j = \emptyset$ for $i \neq j$. Each partition S_i is distributed on node C_i where i =1, 2, …, n.

Step 3 Local Clustering: cluster all partitions concurrently using KDT-SNN on each node C_i, i.e. call KDT-SNN (S_i, *Eps*, *MinPs*) concurrently on C_i for i = 1, 2, …., N.

Step 4 Global Merging: merge the local clustering results obtained from each partition S_i into a global clustering result for the input dataset *D*.

The merging procedure relies on two fundamentals of the original KDT-SNN algorithm. First, all the core points in a cluster satisfy *Eps* criteria. Second, all the non-core points in a cluster are closest to one of the core points in that cluster. This in turn implies that if any two core points in different partitions satisfy *Eps* criteria, the corresponding local clusters need to be merged. The merging stage ensures that all the data points in the input dataset *D* are assigned a global cluster index. All computations in the merging stage are performed at the driver node and none of the operations in this stage is distributed. However, this stage only requires the comparison of the representative points hence, the contribution of this computation latency to the overall computation latency is almost negligible.

4. CASE STUDY

In this section, we discuss several case studies to evaluate the performance of KDT-SNN and Spark-KDT-SNN using three datasets, i.e., Bristol Crime Stats [19], Complex9 dataset [20], TLC Trip Record Data [21], and 911 Call dataset [22]. Complex9 is a 2-dimensional dataset with different shapes. TLC Trip record data is the taxi trip records of New York City, which has different attributes including pick-up and drop-off locations, trip distances, rate types, etc. Figure 1 shows a subset of TLC Trip Record Data.

4.1 Results for KDT-SNN and SNN

We first compare the performance of KDT-SNN with SNN using the same configuration for Bristol Crime Stats, Complex9, and TLC Trip Record Data. Figures 2, 3, and 4 show the visualization of the clustering results of KDT-SNN for each dataset, respectively. Each color represents one cluster on the figures. We can see that the KDT-SNN clustering algorithm can effectively identify clusters of different sizes, shapes, and densities.

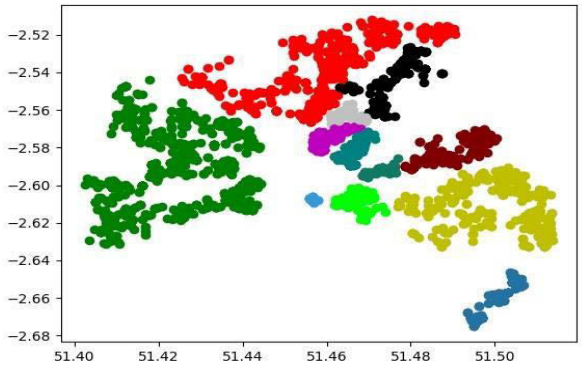

Figure 2. KDT-SNN clustering result of Bristol dataset

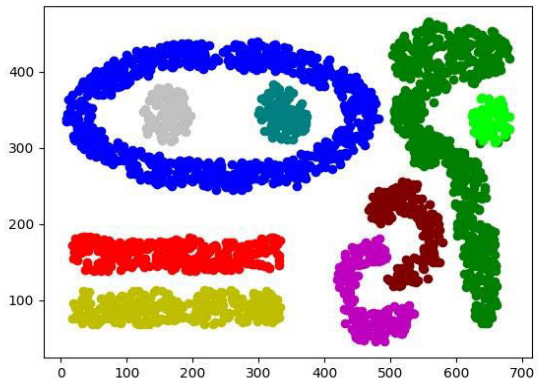

Figure 3. KDT-SNN clustering result of Complex9 dataset

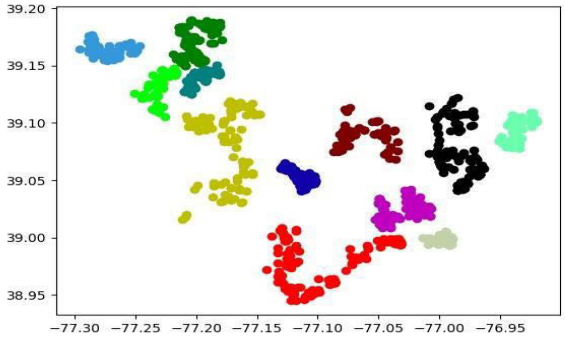

Figure 4. KDT-SNN clustering result of TLC Trip dataset

We also compare KDT-SNN and SNN in terms of runtime for three datasets using the same parameter values. The results are listed in Table 1. Table 1 shows that KDT-SNN is faster than SNN for all three datasets.

Table 1. Runtime Results of KDT-SNN and SNN

Dataset	K	MinPs	Eps	KDT-SNN	SNN
Bristol	40	22	20	48.0505s	514.59s
Complex9	30	18	13	6.297s	37.73s
TLC Trip	28	13	13	10.88s	100.08s

4.2 Results for Spark-KDT-SNN

We use Bristol Crime Stats and 911 Call for the experiments. We fist conduct experiments on a non-distributed single-JVM mode to facilitate the development and testing. In this non-distributed single-JVM deployment mode, Spark spawns all the execution components in the single JVM. Then we conduct the experiments on a lab-size three-node cluster. Each node has an Intel Xeon Processor E5-2603 V4 1.7 GHz processor, 32 GB DDR4 memory, and one 2 TB hard disk. The operating system is Ubuntu 16.04 LTS. For the Spark platform, we install Spark 2.1.0. We chose Spark"s standalone cluster manager as the cluster manager and HDFS as the distributed file system.

We ran the Spark-submit script at the master node. Therefore, the driver program is hosted on the master node. Client mode is selected for two main reasons. First, it does not require storing the results in the file or database as in cluster deployment mode. Secondly, it helps in the dynamic analysis of the results. Moreover, this mode enabled us to decide where to run the driver program. Client mode, however, does not utilize the cluster manager"s ability to find a slave having enough available resources to execute the driver program. Moreover, the driver program cannot be monitored from Spark master web UI like other workers in this mode. Spark framework divides Spark-KDT-SNN into seven stages.

Table 2. The Runtime of Spark-KDT-SNN on Bristol

No. of Cores	No. of Nodes	Time (Seconds)
2	1	900.448
4	1	519.85
6	1	402.71
12	2	227.13
18	3	175.055

Table 3. The Runtime of Spark-KDT-SNN on 911

No. of Cores	No. of Nodes	Time (Seconds)
2	1	5597.4667
4	1	2870.3048
6	1	2224.7991
12	2	1860.25
18	3	1151.66

5. CONCLUSION

We developed a scalable density-based clustering algorithm called Spark-KDT-SNN. It improves the traditional SNN clustering algorithm by utilizing k-d tree as the data structure to compute the nearest neighbor to reduce the computation time. We also presented the parallel implementation of Spark-KDT-SNN for distributed systems. The experimental results demonstrate the efficiency and performance of the Spark-KDT-SNN algorithm. One of the future works is to compare the performance of Spark-KDT-SNN with other density-based clustering algorithms on Spark.

6. ACKNOWLEDGMENTS

This work is supported in part by National Science Foundation (NSF) award OAC- 1726500.

7. REFERENCES

[1] L. Ertoz, M. Steinbach, V.kumar, “A New Shared Nearest Neighbor Clustering Algorithm and its Application,” Workshop on Clustering High Dimensional Data and its Applications at 2nd SDM, 2002.

[2] J. A. Hartigan, M. A. Wong, “Algorithm AS 136: A K-means clustering algorithm,” J.Royal Statistical Soc. C, vol.28, 1979, pp.100-108

[3] M. Ester, H. Kriegel, J. S, X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise,” Proc. 2nd Int. Conf. on KDD, pp. 226-231, 1996.

[4] J.L. Bentley, “Multidimensional Binary Search Trees Used for Associative Searching,” Comm. ACM, vol. 18, pp. 509-517, 1975.

[5] A. Guttman, “R-Trees: A Dynamic index structure for spatial searching,” Proc. ACM SIGMOD, pp.47-57, June 1984.

[6] Apache Spark, [online] Available: https://spark.apache.org/ [Last accessed: Feb. 08, 2020]

[7] H. Ye, H. Lv and Q. Sun, “An improved clustering algorithm based on density and shared nearest neighbor,” *2016 IEEE Information Technology, Networking, Electronic and Automation Control Conference*, Chongqing, 2016, pp. 37-40, doi: 10.1109/ITNEC.2016.7560314.

[8] R. Sharma, K. Verma, “Enhanced shared nearest neighbor clustering approach using fuzzy for teleconnection analysis,” *Soft Comput* 22, 8243–8258 (2018). https://doi.org/10.1007/s00500-017-2767-4

[9] S. Singh and A. Awekar, "Incremental shared nearest neighbor density-based clustering," *2013 the 22nd ACM international conference on Information & Knowledge Management*, October 2013 Pages 1533–1536 https://doi.org/10.1145/2505515.2507837

[10] Q. Li, Q. Wang, and X. Li, “An Efficient Clustering Method for Hyperspectral Optimal Band Selection via Shared Nearest Neighbor”. *Remote Sens.* 2019, *11*, 350.

[11] B. Walter, K. Bala, M. Kulkarni, and K. Pingali, “Fast agglomerative clustering for rendering,” Proc. IEEE Symp. Interactive Ray Tracing, pp. 81-86, 2008. doi:10.1109/RT.2008.4634626

[12] S. Vijayalaksmi, “A Fast Approach to Clustering Datasets using DBSCAN and Pruning Algorithms,” International Journal of Computer Applications, vol. 60, no. 14, pp. 1-7, 2012.

[13] M. Otair, “Approximate k-nearest neighbour based spatial clustering using k-d tree,” IJDMS, vol. 5, no. 1, 2013.

[14] S. Kumari, S. Maurya, P. Goyal, S. S. Balasubramaniam and N. Goyal, "Scalable Parallel Algorithms for Shared Nearest Neighbor Clustering," *2016 IEEE 23rd International Conference on High Performance Computing (HiPC)*, Hyderabad, 2016, pp. 72-81, doi: 10.1109/HiPC.2016.018.

[15] P. Anchalia, “Improved mapreduce k-means clustering algorithm with combiner,” 2014 *the16th International Conference on Computer Modeling and Simulation*, 2014. doi:10.1109/icisa.2013.6579448.

[16] Y. He, H. Tan, W. Luo, H. Mao, D. Ma, S. Feng, and J. Fan, "MR-DBSCAN: An Efficient Parallel Density-Based clustering algorithm using mapreduce," 2011 *IEEE 17th International Conference on Parallel and Distributed Systems*, Tainan, 2011, pp. 473-480, doi: 10.1109/ICPADS.2011.83.

[17] Dianwei Han, Ankit Agrawal, Wei-keng Liao, and Alok Choudhary, “A novel scalable DBSCAN algorithm with Spark,” IEEE, pp. 97879-897, 2016. doi:10.1109/IPDPSW.2016.57

[18] S. Wang and C. F. Eick, "MR-SNN: design of parallel shared nearest neighbor clustering algorithm using mapreduce," *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)*, Beijing, 2017, pp. 312-315, doi: 10.1109/ICBDA.2017.8078831.

[19] Bristol Crime Stats, [online] Available: http://plenar.io/explore/event/bristol_crime_stats. [Accessed: Mar 04, 2019]

[20] The Complex9 dataset, [online] Available: http://dais.cs.uh.edu/datasets/complex9.txt. [Accessed: Mar 06, 2019]

[21] TLC Trip Record Data, [online] Available: https://www1.nyc.gov/site/tlc/about/tlc-record-data.page. [Accessed Mar 05, 2019].

[22] Charleston Calls and Officer Initiated Calls 2009-2013, [online] Available: https://data.world/classicgabe/charleston-calls-and-officer-initiated-calls-2009-2013. [Accessed: Mar 05, 2019]