Distributed Local Outlier Factor with Locality-Sensitive Hashing

by

Lining Zheng

Thesis submitted
In partial fulfillment of the requirements
For the Master of Computer Science (MCS) degree in
Computer Science

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Outlier detection remains a heated area due to its essential role in a wide range of applications, including intrusion detection, fraud detection in finance, medical diagnosis, etc. Local Outlier Factor (LOF) has been one of the most influential outlier detection techniques over the past decades. LOF has distinctive advantages on skewed datasets with regions of various densities. However, the traditional centralized LOF faces new challenges in the era of big data and no longer satisfies the rigid time constraints required by many modern applications, due to its expensive computation overhead. A few researchers have explored the distributed solution of LOF, but existant methods are limited by their grid-based data partitioning strategy, which falls short when applied to high-dimensional data.

In this thesis, we study efficient distributed solutions for LOF. A baseline MapReduce solution for LOF implemented with Apache Spark, named MR-LOF, is introduced. We demonstrate its disadvantages in communication cost and execution time through complexity analysis and experimental evaluation. Then an approximate LOF method is proposed, which relies on locality-sensitive hashing (LSH) for partitioning data and enables fully distributed local computation. We name it MR-LOF-LSH. To further improve the approximate LOF, we introduce a process called cross-partition updating. With cross-partition updating, the actual global k-nearest neighbors (k-NN) of the outlier candidates are found, and the related information of the neighbors is used to update the outlier scores of the candidates. The experimental results show that MR-LOF achieves a speedup of up to 29 times over the centralized LOF. MR-LOF-LSH further reduces the execution time by a factor of up to 9.9 compared to MR-LOF. The results also highlight that MR-LOF-LSH scales well as the cluster size increases. Moreover, with a sufficient candidate size, MR-LOF-LSH is able to detect in most scenarios over 90% of the top outliers with the highest LOF scores computed by the centralized LOF algorithm.

Acknowledgements

I would like to thank everyone who made this possible.

First of all, I would like to express my sincere gratitude and appreciation to my supervisor, Dr. Azzedine Boukerche, who opened the doors for me and made this journey possible. I am also grateful for his continuous support and help both in research and in life. His cheerful sense of humor always gives me the courage to face new challenges.

Secondly, I am very grateful for Dr. Peng Sun, who meticulously revised my thesis and provided most valuable feedback. Besides, many thanks must be given to Dr. Robson De Grande for his patient mentoring and endless encouragement in my darkest days. His serious but enthusiastic attitude for academics has deeply influenced me. I also want to thank Claude Gravel and Qianjia Shy Huang for proofreading my thesis and being such good friends.

Thirdly, I would like to thank all the group members of PARADISE lab. We created a lot of precious memories together.

Lastly, I would like to express my deepest gratitude to my parents, who respected my every decision and have been incredibly supportive throughout my pursuit of the Mater's degree.

Table of Contents

List of Tables vii

List of Figures viii

1 Introduction 1

1.1 Background . . . 1

1.2 Motivation and Objective . . . 2

1.3 Contributions . . . 4

1.4 Thesis Outline . . . 5

Nomenclature 1

2 Preliminaries 6

2.1 MapReduce and Spark . . . 6

2.1.1 MapReduce . . . 6

2.1.2 Apache Spark . . . 8

2.1.2.1 Architecture . . . 8

2.1.2.2 Resilient Distributed Datasets . . . 9

2.2 Locality-Sensitive Hashing . . . 10

2.2.1 Formalization . . . 11

2.2.2 A LSH Function For Euclidean Distance . . . 12

2.2.3 Two-layered LSH . . . 13

3 Literature Review **15**

3.1 Outlier Definition . . . 15

3.2 Outlier Detection with Labeled Training Data . . . 16

3.2.1 Supervised Outlier Detection . . . 16

3.2.2 Semi-supervised Outlier Detection . . . 17

3.3 Unsupervised Outlier Detection . . . 19

3.3.1 Proximity-based Approaches . . . 19

3.3.1.1 Nearest-neighbor-based Approaches . . . 20

3.3.1.2 Clustering-based Approaches . . . 26

3.3.2 Projection-based Approaches . . . 29

3.3.3 High-dimensional Outlier Detection . . . 34

3.3.4 Outlier Detection in Data Streams . . . 39

3.3.4.1 Distance-based Outlier Detection in Data Streams . . . 40

3.3.4.2 Density-based Outlier Detection in Data Streams . . . 43

3.3.4.3 Clustering-based Outlier Detection in Data Streams . . . 46

3.3.5 Distributed Outlier Detection . . . 49

4 Distributed Local Outlier Factor in MapReduce **53**

4.1 MR-LOF: A Baseline Distributed LOF Approach in MapReduce . . . 54

4.1.1 Compute K-Nearest Neighborhood . . . 56

4.1.2 Compute Local Reachability Density . . . 57

4.1.3 Compute Final LOF RDD . . . 59

4.1.4 Complexity Analysis . . . 59

4.1.4.1 Shuffle Cost . . . 60

4.1.4.2 Time Complexity for Computation . . . 61

4.2 MR-LOF-LSH: A Distributed LOF Approach in MapReduce with Locality-Sensitive Hashing . . . 62

4.2.1 LSH Data Partitioning . . . 63
4.2.2 Parallel Computation of LOF . . . 66
4.2.3 Cross-partition Updating . . . 68
4.2.4 Complexity Analysis . . . 69
4.2.4.1 Shuffle Cost . . . 70
4.2.4.2 Time Complexity for Computation . . . 70
4.2.4.3 In Comparison with MR-LOF . . . 71

5 Experimental Evaluation **72**
5.1 Experimental Infrastructure . . . 73
5.2 Datasets . . . 73
5.3 Notable Implementation Details . . . 74
5.3.1 Normalization . . . 74
5.3.2 Duplicate Handling . . . 75
5.4 Evaluation Metrics . . . 76
5.5 Experimental Results . . . 76
5.5.1 Elapsed Execution Time . . . 76
5.5.2 Evaluation of Recall with Different Numbers of Partitions and Candidate Sizes . . . 81
5.5.3 Impact of Varying LSH-related Parameters on Recall . . . 86

6 Conclusion and Future Work **89**
6.1 Conclusion . . . 89
6.2 Future Work . . . 90

References **93**

List of Tables

2.1 Example: amplifying the probability gap of a LSH hash family 12

3.1 Outlier detection with feedback 18
3.2 Nearest-neighbor-based outlier detection 20
3.3 Clustering-based outlier detection 26
3.4 Projection-based outlier detection 30
3.5 Outlier detection for high-dimensional data 35
3.6 Distance-based outlier detection in data streams 40
3.7 Density-based outlier detection in data streams 43
3.8 Clustering-based outlier detection in data streams 46
3.9 Distributed outlier detection 50

4.1 Symbols in the pseudocode and their descriptions 56
4.2 Symbols in the complexity analysis and their descriptions 60

5.1 Default values for the parameters 73
5.2 Overview of the datasets used for evaluation 74

List of Figures

2.1 Word Count: a MapReduce example 7

2.2 Apache Spark Architecture [1] 9

2.3 Illustration of two-layered LSH 14

3.1 INFLO addressing the limitation of LOF: a 2-dimensional example 23

3.2 The intuition of ABOD [2] 37

4.1 Overview of RDD transformations 55

4.2 The overview of MR-LOF-LSH 64

4.3 Illustration of the two-layered mapping from d-dimensional data to 1-dimensional hash value space with segments 65

5.1 Execution time comparison on a cluster of 10 nodes 77

5.2 Execution time comparison varying the cluster size 78

5.3 Test of Scalability of MR-LOF-LSH-CU 80

5.4 Test of recall on Synthetic dataset against different settings of $nPartitions$ and $candidateTimes$ 82

5.5 Test of recall on CoverType dataset against different settings of $nPartitions$ and $candidateTimes$ 83

5.6 Test of recall on KDDCup99 dataset against different settings of $nPartitions$ and $candidateTimes$ 84

5.7 Varying parameter w 87

5.8 Varying parameter $nLSHFunctions$ 88

Chapter 1

Introduction

This chapter first introduces the background of outlier detection. Then we focus on what motivates the research in outlier detection and what is aimed to achieve, after which the main contributions of the thesis are summarized. Lastly, it gives an outline of the thesis.

1.1 Background

In the earliest days, detecting outliers was motivated by data cleansing: removing outliers from the dataset so that parametric statistical models would be able to fit the training data more smoothly. Soon more attention has been turned towards outliers themselves as outliers often represent interesting and critical information, e.g., cyber-attacks in a network, mechanical faults caused by defective industrial equipment, erroneous [3] or malicious [4,5] behavior of a wireless sensor network device, etc. Moreover, with the fast development of the techonology in many domains, data escalate in complexity and volumes. For instance, various emerging protocols have been developed for wireless sensor networks [6–8] to address different challenges: coverage issues [9], energy efficiency [10, 11], security [12], etc. Hence, plenty of research efforts have been devoted to developing high-performance outlier detection techniques to suit the complexity and massiveness of contemporary data. Real-life application scenarios include:

- Intrusion detection systems [13–16]: detecting unusual and malicious activities in computer systems or network systems, based on collected data such as operating system calls and network traffic.

- Fraud detection: identifying credit card frauds [17], financial transaction frauds [18], insurance claim frauds [19], etc.
- Medical anomaly diagnosis [20, 21]: discovering potential disease risks or abnormal patient conditions based on the data collected by medical equipment.
- Fault or defect detection in industrial equipment and products [22, 23].
- Anomaly detection in wireless sensor networks [24]: detecting unexpected anomalous behaviors of the devices caused by unreliable power sources, unstable network connectivity, etc.
- Anomaly detection in urban traffic flow [25]: identifying unexpected and deviant flow values that could be caused by traffic congestions [26], traffic accidents, etc.

Outlier detection is challenging. One important reason is the lack of labeled data due to the rarity of outlier instances. Thus many methods are inherently unsupervised. A typical kind of outlier detection task is: given a set of data instances, identifying those instances (could be a fixed number) that deviate significantly from the rest. However, it is hard to propose a universal mathematical measurement for deviation that suits all datasets and scenarios. Moreover, due to the unsupervised nature, there is a gap between statistically eccentric instances and the instances of interest to users in real life. Over the years, a plethora of research works have emerged in the literature. On the one hand, state-of-the-art techniques (e.g., subsampling and ensembling [27], density peak clustering [28], deep learning [29], etc.) are being adopted to develop general, accurate, and efficient outlier detectors. On the other hand, with the rapid advancement of technologies in various domains (e.g., computer hardware, electronic devices, Internet applications, medical equipment, financial infrastructure, etc.), data come in larger quantities and higher complexity. Thus outlier detection faces new challenges: identifying outliers in data with extremely high dimensionality, in unbounded large volumes of data streams, and in distributed data of large scales, etc.

1.2 Motivation and Objective

As introduced in the previous section, outlier detection is a significant data mining technique that plays a crucial role in a broad range of applications. Local Outlier Factor

(LOF) [30] has become one of the most popular outlier detection methods over the past decades and has inspired plenty of subsequent works [31–34]. Based on the local relative density, LOF is very effective at identifying outliers in datasets containing regions of very different densities.

However, new challenges emerge with the advent of the big data era. Due to the increasing availability of digital information as well as the advancement of technologies for capturing and storing data at a low price, the amount of business data is growing exponentially. A study by McKinsey [35] has reported an annual growth of up to 40% in stored data.

With a large scale of datasets to process, traditional centralized data mining and machine learning methods fall short for a few reasons. First, the resources of an individual computer may not be sufficient to perform the computation tasks, due to limitations in disk storage, memory and CPU. Second, the centralized algorithms may not be able to satisfy the rigid time constraints required by many modern applications, e.g., real-time big data analytic applications. Moreover, the datasets themselves are tending to become more distributed.

Due to the complex nature of the LOF method combined with the big data challenge, a distributed solution for LOF is highly desirable. Yan et al. [36] recently proposed the first distributed solution for LOF in MapReduce, which has exhibited promising performance in processing time. However, a critical limitation of their work is the grid-based data partitioning strategy they have adopted to enable the fully distributed processing of individual partitions, which makes it unsuitable for high-dimensional data. The reason is that the number of partitions grows exponentially with the number of data dimensions. This may lead to two issues: sparse partitions when the number of dimensions is high and the data size is comparatively small; and high duplication rate. We will elaborate on the latter.

Suppose each data attribute is split into t bins and there are m attributes in total, there will be t^m partitions in the grid. In their approach, each partition is extended with a supporting area, which may contain the data points the core partitions needs for the k-NN search. During the k-NN search, the data points in the supporting area are copied and transferred to the core partition from nearby partitions in the grid. The number of adjacent partitions of each partition is 2^m. If the supporting area of a partition is extensive in a dimension, it can span several other partitions in that dimension. Thus, a data point can appear in the supporting areas of many other partitions. This means that

the duplication rate of the data points and thus the communication overhead in the cluster are also exponential to the number of data dimensions.

To address the high dimension issue as well as the big data challenge, we take a different path where we adopt a data partitioning approach based on two-layered locality-sensitive hashing (LSH). We aim to develop a LOF solution that is highly distributed and thus achieve an enormous gain in execution time compared to the centralized algorithm.

1.3 Contributions

The main contributions of this thesis are as follows.

First, a baseline MapReduce solution for LOF in Spark, named MR-LOF, is described. We also conduct complexity analysis, which reveals its high communication and computation overhead. Although compared to the centralized LOF method, it can still significantly reduce the processing time.

Then a distributed approximate LOF method in Spark is proposed, which exploits LSH for data partitioning to enable a fully distributed fashion of data processing. We name it MR-LOF-LSH.

We also develop a strategy called cross-partition updating for MR-LOF-LSH, in which the actual global k-NN and related information are collected for the outlier candidates. We introduce cross-partition updating in hope of producing more accurate approximations of LOF.

Finally, extensive experiments are conducted to evaluate the baseline method and MR-LOF-LSH. We compare the execution time of centralized LOF, MR-LOF and MR-LOF-LSH. Experiments on the scalability of MR-LOF-LSH are also performed. We also evaluate the accuracy of MR-LOF-LSH by varying different parameters. Both real world and synthetic datasets are used, which are representative of many usage scenarios and exhibit variances in the results. The results demonstrate the promising performance of MR-LOF-LSH.

1.4 Thesis Outline

The rest of the thesis is organized as follows. We begin with the introduction of necessary preliminaries in Chapter 2, in which we talk about the MapReduce paradigm, the Spark framework as well as LSH. Chapter 3 is the literature review. We give the definitions of an outlier and present different categories of outliers. We briefly discuss supervised and semi-supervised outlier detection methods in literature then focus on unsupervised methods. In Chapter 4, we present both the baseline distributed LOF method and our proposed MR-LOF-LSH. Experiments and evaluations are described in Chapter 5.

Chapter 2

Preliminaries

This chapter presents the preliminaries to the proposed methods. We first gvie an introduction of the MapReduce paradigm and Apache Spark. Then we give information about LSH.

2.1 MapReduce and Spark

2.1.1 MapReduce

MapReduce [37], introduced in 2004, is a paradigm of computation for distributed and parallel processing of large-scale datasets. The abstraction of MapReduce is inspired by the map and reduce primitives in functional languages. MapReduce has several implementations, including Google's internal implementation and the popular open-source Hadoop[1]. A MapReduce implementation usually takes care of task scheduling, hardware faults, task failures and communication among the machines in a cluster. Users only need to write two types of functions: map and reduce, which specifies how the data should be processed. In most cases, the data to be processed are stored in a distributed file system such as HDFS[2]. The MapReduce framework takes into consideration the data locality [38] in order to have data blocks processed in a nearby computation machine so that the bandwidth overhead can be minimized.

[1] https://hadoop.apache.org/

[2] https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html

In short, a typical MapReduce computation consists of three procedures: map, shuffle and reduce.

- Map: A number of map tasks each take one or more data partitions/chunks from the distributed file system and convert them into a sequence of key-value pairs. The map functions written by users determine what these key-value pairs are like and how they are generated.
- Shuffle: The key-value pairs generated from various data partitions are sorted by their keys globally so that the those with the same key can be sent together to a reduce task assigned to deal with specific keys.
- Reduce: Each reduce task deals with a set of keys. All the key-value pairs with the same key are combined in a way defined by the reduce functions created by the user, for example, summing up the values.

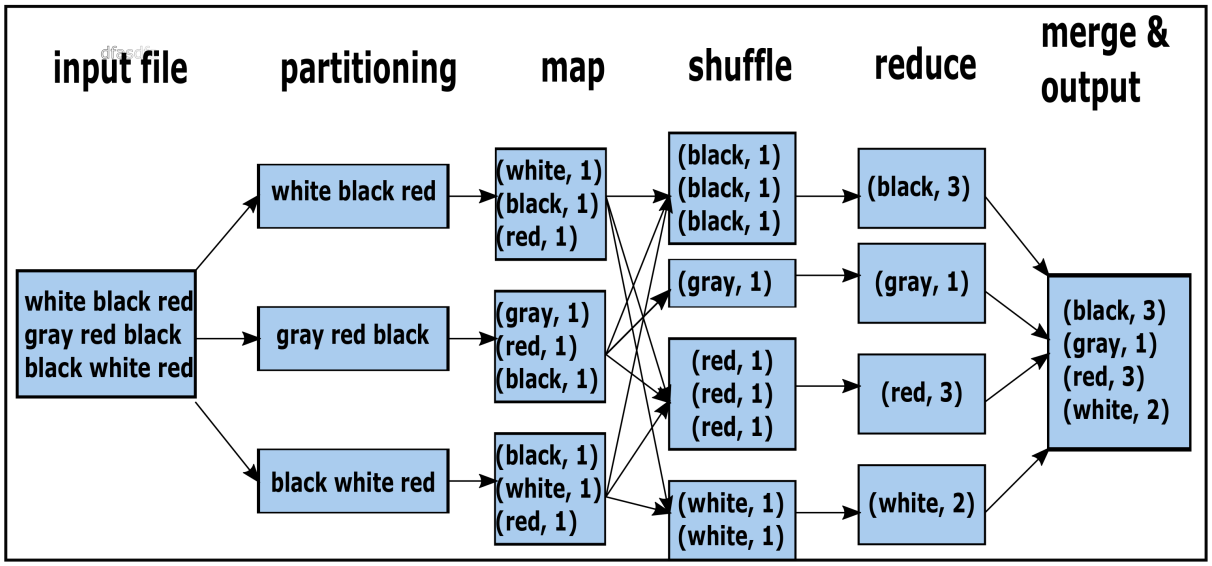


Figure 2.1: Word Count: a MapReduce example

A simple example called "Word Count" is demonstrated in Figure 2.1. What Word Count does is to count the occurrences of each word in the input text file. Firstly, the file is partitioned and stored in a distributed file system. The map tasks are defined to create key-value pairs with the word encountered as the key and 1 as the value. Through shuffling, key-value pairs with the same key are grouped and sent to the same reduce task. Then the reduce tasks sum up the number of occurrences for each individual word. The final output is created by merging the intermediate results of reduce tasks.

2.1.2 Apache Spark

Over the years, Hadoop MapReduce became a very popular MapReduce implementation for cluster computation. However, Hadoop suffers from a number of shortcomings, which have motivated the invention of Apache Spark [39]. Hadoop MapReduce is built around an acyclic data flow model, in which the intermediate results of individual operations are repeatedly written and read from the disk. This model is not capable of efficiently expressing many popular applications such as some machine learning algorithms (e.g., SVM, k-means clustering) that reuse the dataset multiple times to optimize the output models. Apache Spark, on the other hand, completes the data processing in memory and thus often outperforms Hadoop MapReduce by more than 10 times. Aside from batch tasks, Apache Spark can also respond to interactive queries in real time due to the memory-based property while Hadoop suffers from long latency.

As the most actively developed open source framework for parallel and distributed data processing on clusters, Spark supports multiple popular programming languages such as Scala, Python, Java and R. The core abstraction of Spark is called resilient distributed datasets (RDDs), which represents a fault-tolerant immutable (read-only) collection of objects distributed across a cluster of machines. Users can specify how they want the data to be processed by manipulating the RDDs with two types of operation: transformation and action, which will be covered in detail later. In additional to Spark Core, which performs tasks similar to Hadoop MapReduce, Apache Spark also encompasses several extensional components based on Spark Core, namely Spark SQL[3], Spark Streaming[4], MLlib[5] and GraphX[6].

2.1.2.1 Architecture

As illustrated in Figure 2.2, there are three components in the architecture of Apache Spark running in the cluster mode: the cluster manager, the Spark driver and executors. The Spark driver and executors constitute a Spark application while the cluster manager is a pluggable external service that allocates resources across applications. There exist various choices for the cluster manager, including Apace YARN[7], Apache Mesos[8] and Spark's

[3]https://spark.apache.org/sql/
[4]https://spark.apache.org/streaming/
[5]https://spark.apache.org/mllib/
[6]https://spark.apache.org/graphx/
[7]https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html
[8]http://mesos.apache.org/

built-in standalone cluster manager. The driver is the central coordinator process where the *main* method in the user program runs. The driver is responsible for mainly two duties: converting a user program into units of physical execution called tasks and scheduling tasks in appropriate executors taking into consideration of data locality. As for executors, they are processes running on the machines in the cluster that perform computation. The results are either returned to the driver or output to the distributed file system. Executors also provide in-memory storage for cached RDDs.

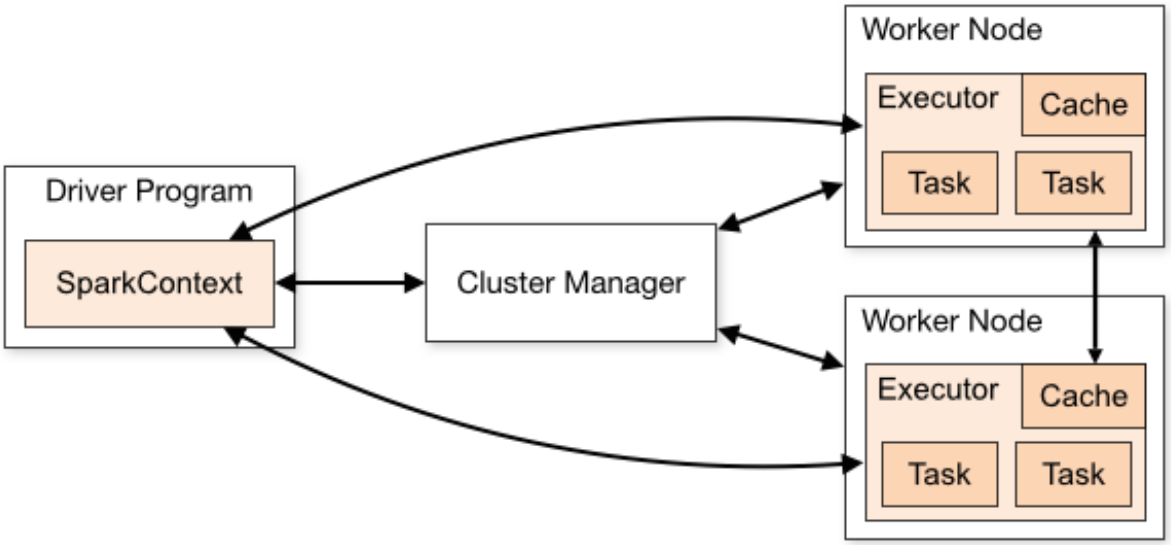


Figure 2.2: Apache Spark Architecture [1]

A typical process of running a Spark application on a cluster is as follows [40]: the user submit an application, and the driver program is launched; the driver program communicates with the cluster manager for allocation of resources to launch executors; cluster manager starts executors which can interact with the driver; the driver runs through the program submitted by the user and send the tasks (including the related application code in the form of JAR or Python files) to the executors; executors perform the computation defined by the tasks and save the results.

2.1.2.2 Resilient Distributed Datasets

The core abstraction of Spark is called Resilient Distributed Datasets (RDD). An RDD is an immutable collection of objects distributed in multiple partitions across the nodes in a cluster. Essentially, all the data processing work is comprised of three types of operations:

creating new RDDs, transforming RDDs and computing a result from RDDs. To create an RDD, the user can load an external dataset or distributing an existing collection of objects in the driver program. RDD has two types of functions: transformations and actions. Transformations derive a new RDD from an existing one (e.g., map function applies a function on every element of the RDD and the collection of the individual results becomes the new RDD) while actions do computations and output a result to the driver program or save it to an external storage system (e.g., reduce function performs an aggregate operation on the collection of objects and return a single result to the driver). The objects in RDDs are distributed and the operations performed on RDDs are also parallel and distributed.

Note that transformations and actions are different because transformations are lazy. This means that the RDDs are not materialized until an action is performed. The advantage of this lazy style is that only the data needed to get the result will be computed after Spark knows the entire chain of transformations. For fault-tolerance considerations, related information is maintained so that when a partition of an RDD is lost, that particular partition of an RDD can be rebuilt. It is also important to know that the feature of persisting an RDD can greatly improve the efficiency of an application. This is due to the fact that RDDs are recomputed every time when an action is run. Thus it is recommended to persist the RDDs that are used repeatedly.

2.2 Locality-Sensitive Hashing

Locality-sensitive hashing (LSH) was first introduced in [41] to address the approximate nearest neighbors problem in the Euclidean space. The general idea behind LSH is to hash items with different hash functions, which are designed to make similar items have a higher probability to be hashed to the same buckets than those items that are less similar to each other. Since then, different LSH families have been developed for different distance measures, including Jaccard distance [42] [43], Cosine distance [44] and Euclidean distance [45]. Besides, some variants focus on improving the speed of existing LSH families [46].LSH has been adopted for various problems, such as high dimensional similarity search [47] [48], outlier detection [49] and distributed clustering [50]. LSH is also well used in the IT industry. For instance, Google uses sim-hash to assess whether a newly crawled web page is a near-duplicate of a previously crawled web page [51].

2.2.1 Formalization

LSH can be formalized as follows [41]:

Definition 1. *A family* $\mathcal{H} = \{h : S \to U\}$ *is called* (r_1, r_2, p_1, p_2)*-sensitive if for any two objects* $v_1, v_2 \in S$

- $Pr\{h(v_1) = h(v_2)\} \geq p_1$ *when* $d(v_1, v_2) \leq r_1$,
- $Pr\{h(v_1) = h(v_2)\} \leq p_2$ *when* $d(v_1, v_2) > r_2$,

where $d(v_1, v_2)$ is the distance between data object v_1 and v_2. The hash family has to satisfy $p_1 > p_2$ and $r_1 < r_2$ to be useful.

The gap between p_1 and p_2 can be amplified by combining several hash functions from the given hash family with two types of constructions. We first discuss the AND-construction, which is described as follows. Given a (r_1, r_2, p_1, p_2)-sensitive hash family $\mathcal{H}$, we can construct a new hash family $\mathcal{G} = \{g : S \to U^k\}$ such that $g(v) = (h_1(v), \cdots, h_k(v))$ for a fixed k. In other words, Each member of $\mathcal{G}$ consists of k members of $\mathcal{H}$ which are independently chosen. We say $g(v_1) = g(v_2)$ if and only $h_i(v_1) = h_i(v_2)$ for all i = 1,2,...,k. Because the members from $\mathcal{H}$ are independently drawn to constitute a member of $\mathcal{G}$. Therefore, $\mathcal{G}$ is $\left(r_1, r_2, p_1^k, p_2^k\right)$-sensitive.

Another construction is called OR-construction, which converts a (r_1, r_2, p_1, p_2)-sensitive hash family $\mathcal{H}$ into a $\left(r_1, r_2, 1-(1-p_1)^L, 1-(1-p_2)^L\right)$-sensitive family $\mathcal{F}$. The OR-construction is defined as follows. Each member f of hash family $\mathcal{F}$ contains L members $h_1, h_2, h_3 \cdot \cdot, h_L$, which are independently chosen from $\mathcal{H}$. We define $f(v_1) = f(v_2)$ if and only if $h_i(v_1) = h_i(v_2)$ for one or more values of i.

If p is the probability of $h(v_1) = h(v_2)$, where h is a member of $\mathcal{H}$, then $1-p$ is the probability $h(v_1) \neq (v_2)$. $(1-p)^L$ is the probability none of $h_1, h_2, h_3 \cdot \cdot, h_L$ make $h(v_1) = h(v_2)$ happen, and $1-(1-p)^L$ is the probability that at least one h_i make $h(v_1) = h(v_2)$ happen.

A $\left(r_1, r_2, 1-\left(1-p_1^k\right)^L, 1-\left(1-p_2^k\right)^L\right)$-sensitive hash family can be created if we cascade AND-construction and OR-construction. Compared to the original hash family, the new one amplifies the gap between the the low probability (from p_1 to $1-\left(1-p_1^k\right)^L$) and high probability (from p_2 to $1-\left(1-p_2^k\right)^L$).

To demonstrate the cascading construction and the effect of amplified gap, let us look at an example. The orignal family is $\mathcal{H}$. We apply the AND-construction to $\mathcal{H}$

p	$1-(1-p^5)^1 0$
0.2	0.003
0.3	0.024
0.4	0.098
0.5	0.272
0.6	0.555
0.7	0.841
0.8	0.981

Table 2.1: Example: amplifying the probability gap of a LSH hash family

with $k = 5$ and create a family $\mathcal{H}_\infty$. Then we employ OR-construction with $L = 10$ on $\mathcal{H}_\infty$ to produce the thrid family $\mathcal{H}_\in$. If $\mathcal{H}$ is (r_1, r_2, p_1, p_2)-sensitive, then $\mathcal{H}_\in$ is $\left(r_1, r_2, 1-(1-p_1^5)^{10}, 1-(1-p_2^5)^{10}\right)$-sensitive. Table 2.1 shows how the gap is amplified. Suppose the original p_1 and p_2 are 0.4 and 0.7 respectively. After the amplification, they become 0.098 and 0.841, forming a wider gap.

2.2.2 A LSH Function For Euclidean Distance

A commonly used LSH family for Euclidean distance is proposed in [45]:

$$h(v) = \lfloor \frac{a \cdot v + b}{w} \rfloor, \tag{2.1}$$

where v is a d-dimensional data point, a is a d-dimensional random vector, each entry of which is independently selected from a p-stable distribution [52], b is a random real number chosen from the uniform distribution bounded by $[0, w]$, and w is a real number parameter, called the "width" of the function.

Now we compute the probability that two points v_1 and v_2 collide under a hash function drawn randomly from this family. According to the property of the p-stable distribution, $\sum_i a_{(i)} v_{(i)}$ has the same distribution as $\left(\sum_i |v_{(i)}|^p\right)^{1/p} X$, where X is the p-stable distribution from which the entries of a are randomly drawn and $v_{(i)}$ is the i^{th} entry in the data point. Let $f(x)$ be the probability density function of the absolute value of the p-stable distribution X. For instance, if X is the standard Gaussian distribution,

$$f(x) = \begin{cases} 0 & x < 0 \\ \frac{2}{\sqrt{2\pi}} e^{-x^2/2} & x \geq 0 \end{cases}. \tag{2.2}$$

Let v_1 and v_2 be two data instances and $c = \|v_1 - v_2\|_p$. Thus $(a \cdot v_1 - a \cdot v_2)$ has the same distribution as cX. Therefore, the probability density function of $|a \cdot v_1 - a \cdot v_2|$ is $\frac{1}{c} f\left(\frac{x}{c}\right)$. Since b is a random number uniformly drawn from $[0, w]$, we can conclude that

$$Pr\left\{h\left(v_1\right) = h\left(v_2\right)\right\} = \int_0^w \frac{1}{c} f\left(\frac{x}{c}\right)\left(1 - \frac{x}{w}\right) \mathrm{d}x. \tag{2.3}$$

If we take the derivative of the above formula in terms of c, the result can be proved to be smaller than 0. Thus the probability that v_1 and v_2 collide decreases monotonically with the distance between them.

2.2.3 Two-layered LSH

Haghani et al. formally proved in their work [53] that using a hash function drawn from the family as described in Eq. (3.14), two data instances having a shorter distance to each other enjoy a higher probability of resulting in two hash values with a smaller difference. This is formalized as 2.2.1.

Theorem 2.2.1. For two data points $v, q \in S$, $c = \|v, q\|_2$ and a fixed distance δ, $Pr\left[\left|h\left(v\right) - h\left(q\right)\right| \leq \delta\right]$ monotonically decreases in terms of c, i.e., a negative correlation exists between $Pr\left[\left|h\left(v\right) - h\left(q\right)\right| \leq \delta\right]$ and c.

Based on this theorem, Haghani et al. have proposed a two-layered hash strategy which takes the output of k LSH functions on the first layer as the input of the second layer LSH function. In other words, this approach maps the d-dimensional dataset to the k-dimensional p-stable LSH bucket space, then to the 1-dimensional machine identifier space, as illustrated in Figure 2.3. For the first layer mapping, k LSH hash functions are drawn from the family as illustrated by Equation 3.14, the results of which form a k-dimensional vector as the input of the second layer LSH function. Possible options for the the second layer hash function are simple summing [54], LSH functions based on 1-stable Cauchy distribution [53] and LSH functions based on 2-stable Gaussian distribution [55].

The benefit of the two-layered LSH approach is that data instances close to each other will have similar resultant hash values in the 1-dimensional space. With slight modification on the resultant hash values, we can obtain data partition identifiers for individual data instances so that the data instances in the same partition have comparatively shorter distances to each other than to data instances in different partitions.

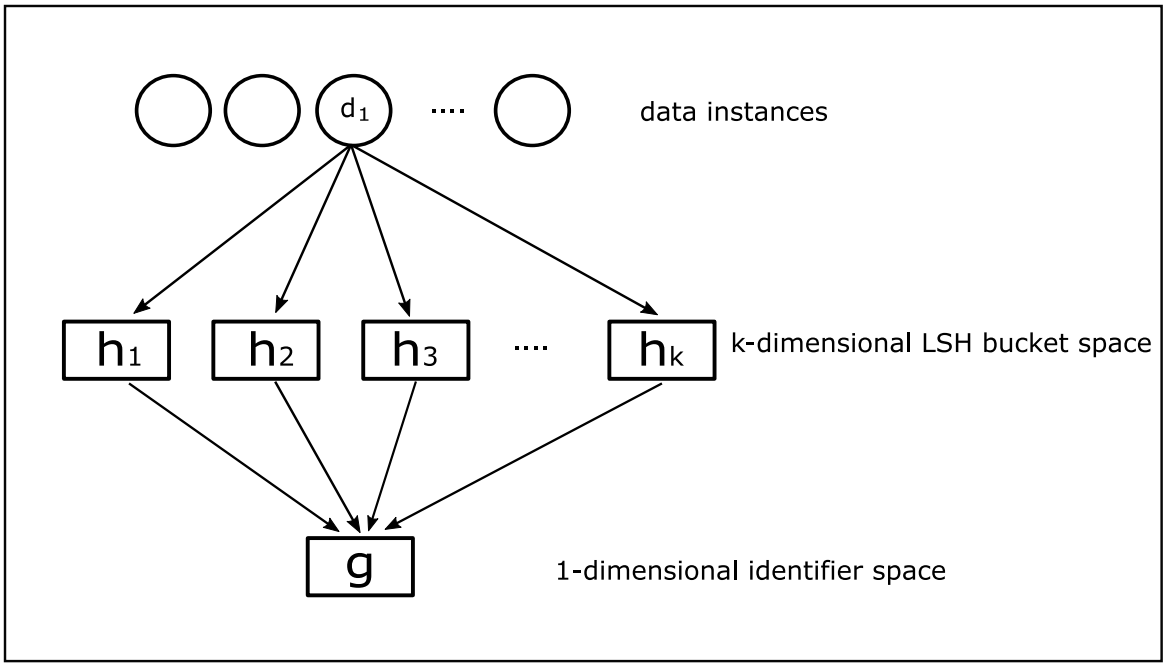


Figure 2.3: Illustration of two-layered LSH

Chapter 3

Literature Review

In this chapter, several definitions of outliers and different categories of outliers are first presented. Then we briefly discuss the supervised and semi-supervised outlier detection methods in literature, which both require labeled training data. Finally, the majority of our focus is laid upon the unsupervised outlier detection approaches proposed in the last few years. The unsupervised outlier detection algorithms are grouped into several categories based on the underlying techniques adopted and application scenarios. These categories are nearest neighbor based techniques, clustering-based techniques, projection-based techniques, outlier detection techniques for high-dimensional data and distributed outlier detection techniques.

3.1 Outlier Definition

The first definition of the outlier is likely attributable to Grubbs in 1969 [56], "an outlier is one that appears to deviate markedly from other members of the sample in which it occurs." Other apt definitions include "an observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data" [57], "a data point which is significantly different from other data points, or does not conform to the expected normal behaviour, or conforms well to a defined abnormal behaviour" [58], etc. Outliers are defined typically based on the following assumptions [59]: 1) outliers are different from the norm with respect to their features; 2) outliers are rare in a dataset compared to normal instances.

There are different ways to categorize outliers. First, based on the number of data

instances involved to comprise an outlying pattern, there are 1) point outliers and 2) collective outliers [58]. A point outlier is an individual data instance that deviates largely from the rest of the dataset. This is the simplest type of outlier to identify and is the major focus of the research on outlier detection [58]. Collective outliers are a collection of data instances that appear anomalous with respect to the rest of the entire dataset. However, each of the instances within the collection may not be outliers individually. An example of collective outliers is a specific sequence of access actions encountered in intrusion detection.

Based on the scope of comparison, point outliers can be further classified into 1) local outliers and 2) global outliers. The notion of local outliers was first introduced in LOF [30]. The detection of local outliers relies on the characteristic difference (e.g., neighborhood density) between the outlier and its nearest neighbors. Global outliers address the difference with the entire dataset.

Based on the type of data, outliers can be categorized into 1) vector outliers and 2) graph outliers [60]. Vector outliers are mentioned with vector-like multi-dimensional data, while graph outliers exist in graph data. A vector-like data point has multiple attributes, each of which has either a numeric value or a categorical value. The outlier detection methods rely on a distance definition between two vector-like data points (e.g., Euclidean distance and Cosine distance). Graph data consist of nodes and edges, whdich well represent the inter-dependencies among data objects. Outliers in graph data can be point outliers (e.g., node outliers and edge outliers) or collective outliers (e.g., sub-graph outliers) [60]. Readers can refer to [61] for a comprehensive survey of outlier detection in graph data.

3.2 Outlier Detection with Labeled Training Data

3.2.1 Supervised Outlier Detection

In the supervised model, training data are supposed to be labeled as outliers or non-outliers, based on which a predictive model (classifier) is built. The predictive model predicts which class the input new data belong to. Essentially the supervised outlier detection problem becomes a classification problem, where the training data is very imbalanced.

Note that two issues exist for supervised outlier detection [58]. The first issue is the imbalanced instances of outliers and non-outliers in the training set, which can greatly compromise the performance of conventional classification algorithms. The issue has been

addressed by the machine learning community with approaches such as resampling [62], boosting [63, 64], bagging [65, 66]. The second issue is the difficult access to a sufficient amount of accurate and representative labeled training data. For instance, new types of outliers may arise, to which the related training data may not exist. One possible solution to address these issues is to artificially create outliers into the training data [67, 68].

3.2.2 Semi-supervised Outlier Detection

Semi-supervised outlier detection can be viewed as a special case of semi-supervised learning [69], with plentiful unlabeled training and scarce labeled training data. This paradigm suits the property of most of the outlier detection problems that labeled training data are hard to acquire while provides better performance compared to unsupervised methods due to the label information as feedback.

However, it is worth noting that some, for instance [58], refer to the semi-supervised outlier detection model as a model built by using training data which only cover the normal instances. In this case, the built model only represents the normal data but can be used to identify outliers that deviate from it. This idea of semi-supervised outlier detection resembles the one-class classification [70], which builds a model to determine if an instance belongs to a learned class. A well-known example is One-class SVM [71]. However, a majority of the semi-supervised outlier detection methods use both labeled and unlabeled training data, where the labeled data consist of both normal instances and outliers.

A typical way to obtain and utilize the labeled instances is through active learning [72]: the initial model is built upon unlabeled data, based on which some data instances are selected by some query strategies to be labeled by a domain expert. Then the model is updated with the newly acquired label information. This type of feedback loop can be carried on iteratively until certain criteria are met. This section surveys some of the newest works in the literature that focus on incorporating human feedback into outlier detection to improve the detection accuracy. A summary of the approaches in this section can be found in Table 3.1.

In the work by Görnitz et al. [73], anomaly detection is regarded as an optimization problem named support vector data description (SVDD) [74]. SVDD computes a hypersphere to enclose the data, with radius r and center c. The hypersphere represents normality. The anomaly scores are based on the distances to the center c: data points found outside the hypersphere ball are considered outliers, whereas data points inside are

Table 3.1: Outlier detection with feedback

Algorithm	Features	Base Outlier Detector	Active Learning Strategy	Comments
Görnitz et al. [73]	Tailored active learning strategy	SVDD [74]	Clusters near decision boundary	Robust against obfuscation & quick accuracy boost over a few labels
Das et al. [75]	AATP [76] & adjusted ensemble weights	Loda [77]	The most anomalous instances	High accuracy & generalizable
Vercruyssen et al. [23]	Label propagation	Constrained k-means based	Decision boundary	High accuracy
Siddiqui et al. [78]	Online convex optimization [79]	Isolation Forest [80]	The most anomalous instances	High accuracy & high efficiency & generalizable

viewed inliers. They present a generalized support vector data description making use of labeled data: data points with inlier labels are required to reside within the hypersphere and vice versa. Thus it becomes a semi-supervised outlier detection problem. They show that the new optimization problem is unconvex but can be converted into a convex equivalent under mild assumptions. Additionally, different active learning strategies are introduced, which not only query the instances on the borderline but also those that could lead to the discovery of novel outlier categories.

Das et al. [75] proposed a semi-supervised approach that iteratively incorporates expert feedback into the model of an ensemble anomaly detection approach called Loda [77]. They aim at presenting the maximum number of anomalies to the expert. Thus, the instance with the highest anomaly score is selected for labeling in each iteration. The label information is then used to update the weights for the projections in Loda so that projections more effective at isolating anomalies are assigned higher weights. To achieve that effect, they devised an objective function modified from the accuracy at the top (AATP) approach [76]. The direct effect is that false positives are downgraded in the internal ranking based on the outlier scores produced by Loda, whereas true positives are pushed up in the ranking. The proposed framework can be generalized for many other methods based on random projections besides Loda.

Vercruyssen et al. [23] described a semi-supervised anomaly detection approach that employs constrained k-means clustering [81] to perform the initial unsupervised scoring and iteratively updates the anomaly scores by incorporating expert labeling. In the clustering phase, the scoring formula is based on several intuitions: anomalies tend to deviate from its cluster centroid; the centroid of an anomalous cluster tends to deviate from other centroids; smaller clusters are more likely to bear anomalies. Then whenever new expert

labels are available, the anomaly scores can be updated for unlabeled instances based on their distances to the labeled anomalies. This process is called label propagation. The underlying assumption is that unlabeled instances with shorter distances to the labeled anomalies should increase their scores compared to their peers. In label propagation, they introduce a weighting parameter to control the influence of the label information versus the score obtained from the clustering phase. To improve the detection accuracy, they used uncertainty sampling, which is choosing the unlabeled instances with a score closest to 0.5 for the expert to label.

Siddiqui et al. [78] proposed a general algorithm for anomaly detection that aims at aligning anomaly scores with the application-specific interestingness by incorporating expert feedback. They framed this anomaly detection problem with online convex optimization [79] and provided two loss functions that correspond to two different methods. The loss functions are associated with human expert feedback and promote the anomalies scores that are consistent with the feedback. A way to instantiate the algorithm with tree-based anomaly detection methods (e.g., isolation forest [80]) is described, which is achieved by adjusting the weights of the edges in the trees according to the feedback.

3.3 Unsupervised Outlier Detection

As the name suggests, unsupervised outlier detection does not require labeled training data. This property makes unsupervised outlier detection methods more preferable for many real-world problems due to the unavailability of labeled data. Next, we will look into some of the representative algorithms proposed recently.

3.3.1 Proximity-based Approaches

The proximity-based approaches identify outliers based on their relations with nearby data points. A common situation is that an outlier is located in a sparse area, with very few data points within a given distance or the nearest data points are very far away. The notion of proximity can be defined in various ways. In this section, we focus on the techniques that address the proximity with nearest neighbors and clusters.

3.3.1.1 Nearest-neighbor-based Approaches

Nearest-neighbor-based outlier detection approaches measure the degree of abnormality on the basis of a data point's relation to its nearest neighbors. There are two main ways to define the neighborhood: k nearest neighbors (k-NN) and a neighborhood within a pre-specified radius, centered by a data point. The underlying assumption is that normal data instances are closer to their neighbors, thus forming a dense neighborhood, whereas outliers are far from their neighbors, thus sparsely populated.

In this section, we investigate several classical outlier detection approaches based on the nearest neighbors, as well as more recent approaches taking advantage of subsampling and ensembling. Table 3.2 is a summary of the nearest-neighbor-based approaches introduced in this section.

Table 3.2: Nearest-neighbor-based outlier detection

Algorithm	Features	Time Complexity	Local Outlier	Comments
LOF [30]	N/A	$O(N^2)$	Yes	First to address local outliers
COF [82]	Shortest path to connect neighbors	$O(N^2)$	Yes	Address non-spherical distributions
LOCI [83]	Count-based neighborhood density	$O(N^3)$	Yes	Free of parameters but high time complexity
INFLO [84]	Reversed nearest neighbors	$O(N^2)$	Yes	Improvement on border points between two areas with different densities
LoOP [32]	Assuming Gaussian distribution of distances to k-NN	$O(N^2)$	Yes	Interpretability of output
iNNE [85]	Subsampling and ensembling	$O(N\psi t)$	Yes	Highly efficient
LeSiNN [27]	Subsampling and ensembling	$O(N\psi t)$	No	Intuitive & highly efficient

[1] t denotes the number of sample sets.
[2] ψ denotes the number of instances within each sample set.

Some of the primitive nearest-neighbor-based approaches are very straightforward and intuitive. For instance, the approach by Ramaswamy et.al [86] uses the distance to the kth nearest neighbor as the outlier score. The method by Angiulli et al. [86] uses the sum of distances to the k-NN [87]. Knorr et al. [88] rely on the number of neighbors within a pre-defined radius of a data point. Because the degree of abnormality is compared in the context of the entire dataset, these methods detect global outliers. They assume that the density across different regions of the dataset is homogeneous. However, this assumption may not hold for many real-life datasets. Thus they are often outperformed in terms of detection accuracy by approaches that take into consideration varied density [89]. The latter type of approach focuses on local outliers.

The Local Outlier Factor (LOF) [30] is a well-known approach that first introduced the

concept of local outliers, and has inspired many subsequent works on local outliers. Local outliers are significantly different with regard to its closeby data points. The LOF score for a data instance is based on the average ratio of the instance's neighbor's density to that instance's density. In other words, the outlier score is the density normalized by the density of the neighbors. The normalization by the neighbors is how LOF addresses the local outlier. The detailed procedure of calculating the LOF score is described as below.

First, the k-NN need to be obtained for each data instance p. Second, the local reachability density (LRD) is calculated based on the average reachability distance from p to its k-NN:

$$LRD\left(p\right) = \left(\frac{\sum\limits_{o \in N_k(p)} d_k\left(p, o\right)}{\mid N_k\left(p\right) \mid} \right)^{-1}, \tag{3.1}$$

where $N_k\left(p\right)$ is the k-nearest neighborhood of p and $d_k\left(p, o\right)$ is the reachability distance, which is defined as the larger value between the kth nearest neighbor distance to o (k-distance) and the distance between p and o, i.e.,

$$d_k\left(p, o\right) = \max\left\{k\text{-distance}(o), \text{distance}(p, o)\right\}. \tag{3.2}$$

The local reachability density is basically the reciprocal of the average distance to the neighbors unless there exist some neighbors that are "sufficiently close". The reason to introduce reachability distance other is to create a smoothing effect that reduces the statistical fluctuations of $d\left(p, o\right)$ for all the o's close to p [30]. Finally, the LOF score can be calculated by comparing the local reachability density (LRD) of p with all its k neighbors' LRDs:

$$LOF\left(p\right) = \frac{\sum\limits_{o \in N_k(p)} \frac{LRD_k(o)}{LRD_k(p)}}{\mid N_k\left(p\right) \mid}, \tag{3.3}$$

which equals to

$$LOF\left(p\right) = \frac{\sum\limits_{o \in N_k(p)} LRD_k\left(o\right)}{\mid N_k\left(p\right) \mid LRD_k\left(p\right)}. \tag{3.4}$$

Informally, the LOF score of p is the average ratio of p's neighbors' density to p's density. Usually outliers have neighbors with a higher density. Thus outliers have LOF scores higher than normal one, and a higher score indicates an instance is more likely to be an outlier.

The Connectivity-based Outlier Factor (COF) [82] addresses the shortcomings of LOF, which assumes that the outlier pattern is only low density in an Euclidean distance-based

spherical neighborhood. However, such a view of outliers is overly simplified, and outliers in other patterns of neighborhood relations may not be successfully identified. For example, the normal instances of a two-dimensional dataset distribute roughly along a straight line. An outlier lies astray from the straight line but still has a considerable density. This type of outlier will have a similar LOF score to the normal data points. To overcome this shortcoming of LOF, COF uses the notion of "isolativity", which is the degree that a data point is connected with others. To quantify the "isolativity", COF uses the "chaining distance", which can be viewed as the shortest path connecting the k neighbors and the data instance. Then the COF for a data point is its chaining distance normalized by the average of the chaining distance of its k-NN.

Papadimitriou et al. [83] proposed Local Correlation Integral (LOCI) based on the definition of local density, which is the count of neighbors within a radius r around a data point (r-neighborhood). They devised a related measure called Multi-Granularity Deviation Factor (MDEF). The MDEF with a given radius r for a data point p equals one minus the ratio of the local density of p to the average local density of the points in the r-neighborhood of p. MDEF represents the degree that the data point deviates from its neighbors in terms of local density. Another related measure is δ_{MDEF}, which is the standard deviation of the local density of the points in the r-neighborhood normalized by the average local density in the r-neighborhood. To determine whether a data instance is an outlier, with the radius r increasing in each iteration, the MDEF and δ_{MDEF} of the data point are calculated, and if MDEF is larger than three times of δ_{MDEF} in any iteration, the data point is labeled as an outlier. An advantage of LOCI is that it does not require parameters, for instance, k in k-NN, which is a crucial and difficult choice. Instead, it expands the radius of the r-neighborhood and derives a binary outlier label on the basis of the standard deviation of the MDEF. Thus, another advantage of LOCI is that it is free of outlier cutoff threshold that must be specified by users in other approaches. However, due to the iteration for the radius expansion, the time complexity is $O(N^3)$. Aware of the high complexity of LOCI, the authors have proposed an approximate method named aLOCI [83]. aLOCI approximates the neighborhood using a space partitioning grid, resulting in practically linear performance.

Influenced Outlierness (INFLO) [84] uses a reverse nearest neighborhood set (k-RNN) combined with the k-NN to compute the outlier score. The k-RNN of a data point p is the set of other instances whose k-nearest neighborhood includes p. Thus, the size of a k-RNN set is not necessarily k. The rest of the computation is similar to LOF: the outlier score

is derived by dividing the local density of p by the average density of p's neighborhood. The incentive of incorporating k-RNN for outlier analysis is to address the limitation of LOF that LOF fails to appropriately score the instances on the borders of clusters with significantly different densities. As depicted in Figure 3.1, data point p is on the board of a dense region (right) and a sparser region (left). Most of the members of p's k-NN would be from the dense region, resulting in a high LOF score because the neighbors from the dense region have higher density. However, p is not supposed to be deemed as anomalous considering the sparser region. On the other hand, if we take into account the k-RNN as INFLO describes, the extended neighborhood set would also contain many members from the sparser region. Thus, a more reasonable outlier score will be assigned to p, and p will not be viewed as an outlier.

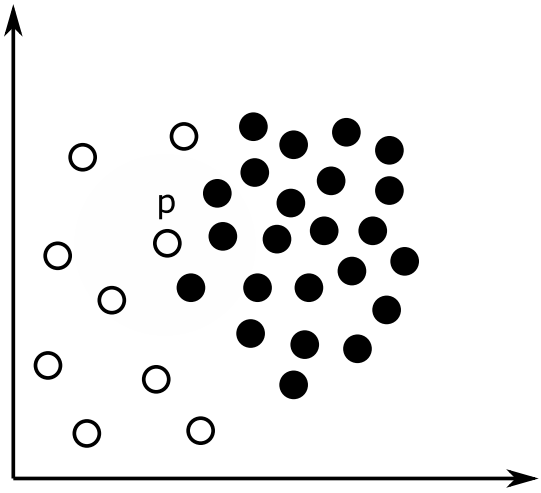


Figure 3.1: INFLO addressing the limitation of LOF: a 2-dimensional example

Kriegel et al. [32] proposed the Local Outlier Probability (LoOP), which outputs a probability that indicates the likelihood of a data point being an outlier. LoOP attempts to tackle the dilemma other methods face: how to choose the suitable cut-off threshold for outlier scores to distinguish between outliers and inliers. The formulated LoOP ranges from 0 to 1 with an interpretable meaning and thus can be more useful in practical scenarios. The computation framework of LoOP is similar to LOF: compute local density and normalize it with neighborhood average. However, LoOP differs in the way it calculates the local density for a data point. It is assumed that a data point p is at the center of its neighborhood, and the distances to its k-NN follow a half-Gaussian distribution (distance is always non-

negative). Accordingly, a quantity named standard distance is defined:

$$\sigma\left(p, N_k\left(p\right)\right) = \sqrt{\frac{\sum\limits_{o \in N_k(p)} \operatorname{dist}\left(p, o\right)^2}{\left|N_k\left(p\right)\right|}}, \tag{3.5}$$

where $N_k\left(p\right)$ is the k-NN of p. The standard distance resembles the deviation of $\operatorname{dist}(p, o)$, where $o \in N_k\left(p\right)$. However, The standard distance uses 0 as the mean. Then the probabilistic set distance is used as the estimated density, which is defined as:

$$\operatorname{pdist}\left(\lambda, p\right) = \lambda \cdot \sigma\left(p, N_k\left(p\right)\right) \tag{3.6}$$

where λ is merely a parameter controlling the contrast in the output scores without affecting the ranking. To normalize the density with regard to the average of the k-NN, Probabilistic Local Outlier Factor (PLOF) is defined as:

$$PLOF\left(p\right) = \frac{\operatorname{pdist}\left(\lambda, p\right) \cdot \left|N_k\left(p\right)\right|}{\sum\limits_{o \in N_k(p)} \operatorname{pdist}\left(\lambda, o\right)} - 1 \tag{3.7}$$

Finally, to convert PLOF into a probability, normalization by deviation and a Gaussian error function is used.

Ting et al. [90] pointed out in their work that nearest-neighbor-based outlier detection approaches are contrary to the conventional belief that more training data produce better results. Instead, using only samples from the original dataset gives rise to better performance for nearest-neighbor-based approaches. They argued that there exists an optimal sample size for an individual dataset. When the actual sample used is smaller than the optimal size, the data distribution is not well represented. But when the actual sample size increases above the optimal size, the resultant accuracy tends to decrease because the separation between normal data points and outliers diminishes. Put in another way, using small samples reduces the masking effect where outlier instances forming clusters are mistakenly regarded as normal instances [27, 80].

Based on subsampling, iNNE (isolation using Nearest Neighbour Ensemble) [85] creates isolation regions to determine outlier scores. An isolation model is built for each sample set. For each sample instance c within a sample set $\mathcal{S}$, a hypersphere $B(c)$ is built with the sample instance at the center and the radius $r(c)$ as the distance between the sample instance and its nearest neighbor within the sample set. The isolation score for a data

point p with regard to a sample $\mathcal{S}$ is defined as

$$I(p) = \begin{cases} 1 - \frac{r(nn(minS(x)))}{r(minS(x))} & x \in \bigcup_{c \in \mathcal{S}} B(c) \\ 1 & \text{otherwise} \end{cases}, \tag{3.8}$$

where $minS(x)$ is the sample instance with the minimal hypersphere that x falls in, and $nn(c)$ is the nearest neighbor of c in the sample set. According to the equation, if the data point falls within the isolation hypersphere of any sample instance, the isolation score will be less than 1. The sample instance with the smallest radius is picked as the proxy of x. The score is then calculated as the radius ratio between the sample instance and the sample instance's nearest neighbor in the sample set. The comparative ratio of the radiuses is to address local outliers.

LeSiNN [27] is another outlier detection method that also builds models with subsampling. The outlier score for a data point p with regard to a sample set $\mathcal{S}$ is simply defined as the distance between p and p's nearest neighbor in $\mathcal{S}$. Note that both iNNE and LeSiNN have a linear time complexity because the k-NN search for a data point is limited within a sample set, and the sample size is constant. Besides, both iNNE and LeSiNN use an ensemble to ensure the stability of the outlier detector. The final outlier score with the ensemble is the average score over multiple sets of samples.

Nearest-neighbor-based methods have the advantage of a more refined granularity on the outlier analysis over clustering-based approaches. This enables nearest-neighbor-based methods to differentiate between strong outliers and weak outliers that are more likely to be considered as noise [89]. However, high computation complexity usually comes as a cost, due to the expensive computation of the pairwise distances. Moreover, the choice of k has a significant impact on the performance. But the optimal choice of k varies for different approaches and datasets. An overly large k results in a weak distinction between outliers and normal points. An overly small k results in an unreliable estimation of the proximity density.

Using subsampling is a good way to reduce the time complexity to linear. Subsampling also helps with the aforementioned masking effect. Coupled with ensembling, subsampling-based methods can also deliver promising and reliable performance. However, the new problem is to decide the suitable sample size and ensemble size. Typically, when dealing with large datasets, a large ensemble size is desired for good performance. This could, however, cause a considerable increase in execution time.

3.3.1.2 Clustering-based Approaches

Clustering is an extensively studied data mining technique that groups data into multiple clusters with similar data instances ending up in the same cluster. Outlier detection algorithms based on clustering usually take a two-step procedure: grouping the data with clustering algorithms and analyze the degree of deviation based on the clustering results. As pointed as by Aggarwal [89], there is a complementary relationship between clusters and outliers, which can be simplistically put as that a data point not belonging to any clusters is considered an outlier. Aside from the cluster membership (whether or not in a cluster), there are two other commonly used cluster-related quantities to construct an outlier score. The first is the distance to the cluster center, the assumption being that normal data points are close to the cluster centers, whereas the outliers are far from them. The second is the cardinality of a cluster, the assumption being that the cluster of normal data points is dense and large, whereas the cluster of outliers is sparse and small.

Compared with nearest-neighbor-based approaches, a major advantage of clustering-based outlier detection is its efficiency. For instance, the time complexity for k-means clustering is $O(Nkt)$, with N data instances, k cluster centers and t iterations. Usually k and t are far smaller than n. In contrast, nearest-neighbor-based approaches typically induce quadratic time complexity due to the pair-wise distance computations. However, nearest-neighbor-based approaches depending on the point-to-point distance provide more refined granularity compared to clustering-based approaches, which employ simplified representations for aggregation of data points, e.g., cluster centers. In this section, we introduce some representative outlier analysis approaches based on clustering. They are summarized in Table 3.3.

Table 3.3: Clustering-based outlier detection

Algorithm	Clustering Algorithm	Features	Score Based on	Predefined Cluster #	Comments
Jiang et al. [91]	k-means [92]	Minimal spanning tree of cluster centers	cluster cadinality	Yes	Time efficient but coarse granularity & spherical clusters
CBLOF [93]	Arbitrary	Heuristic small and large clusters	Cluster cadinality & distance to cluster center	N/A	Too many parameters & misuse of cluster cardinality
LDCOF [94]	Arbitrary	Local normalization	Cluster cadinality & distance to cluster center	N/A	Detection of local outliers but too many parameters
Du et al. [28]	Density peak clustering [95]	Chebyshev's theorem for statistical threshold determination	Standard deviations of δ	No	Intuitive & arbitrary-shaped clusters

Jiang et al. [91] presented an outlier detection approach based on a modified version of k-means clustering and a minimum spanning tree constructed from the cluster centers. The modified k-means clustering has an initial value and an upper bound for the number of clusters. If an encountered data point is far from all of the existing cluster centers, this data point will be assigned the center of a new cluster, which means the number of clusters increases by one. To determine how far is enough for the creation of a new cluster, two distances are involved. The first one is the shortest distance between any two cluster centers, which is maintained and updated when there are changes to the clusters. The second one is the distance between the data point and its nearest cluster center. A new cluster will be created if the first distance is no less than the second distance. When the actual number of clusters exceeds the upper bound, two clusters whose centers have the shortest distance will be merged into a single cluster. Similar to k-means, the modified version also iterates through the entire dataset for a number of times, with the goal of minimizing the sum of the data point distance to its cluster center. As for the outlier detection phase, a minimum spanning tree is first created with the cluster centers as the nodes and their distance between one another as the edge weight. Then longest edges are repeatedly removed until the number of subtrees becomes k. The data points in the subtrees with the smallest cardinality are regarded as outliers.

The Cluster-based Local Outlier Factor (CBLOF) [93] is a clustering-based outlier detection approach that distinguishes small and large clusters by a quantitative measure. Given a set of k clusters $\{C_1, C_2, ..., C_k\}$, sorted by the decreasing order of the cluster cardinality, and two numeric parameters α, β, a boundary cluster C_b has at least one of the following two conditions hold: (1) $\sum_{i=1}^{b} |C_i| \geq \alpha |D|$; (2) $|C_b|/|C_{b+1}| \geq \beta$. Accordingly, the clusters after C_b in the sorted sequence are defined as small clusters, whereas the rest are large clusters. The intuition behind the first condition is that outliers account for only a small portion of the entire dataset. The second condition is due to the consideration that clusters with a high possibility of being outliers should be significantly smaller in size. Then the outlier score for data point p is defined on the basis of small clusters and large clusters:

$$\text{CBLOF}(p) = \begin{cases} |C_i| \cdot \min(\text{dist}(p, C_j)) & C_i \text{ is a small cluster} \\ |C_i| \cdot \text{dist}(p, C_i) & C_i \text{ is a large cluster} \end{cases}, \tag{3.9}$$

where $p \in C_i$, and C_j is a large cluster that does not include p. The cluster cardinality used as the scaling factor is intended to make the algorithm able to detect local outliers. The assumption is that a larger cardinality is associated with a lower density. However, this

does not hold in most cases. On the contrary, a large cardinality is supposed to indicate normality.

Later in the work by Amer et al. [94], it is demonstrated that simply removing the cluster cardinality of CBLOF can produce better results, which is named the unweighted-CBLOF:

$$\text{unweighted-CBLOF}(p) = \begin{cases} \min(\text{dist}(p, C_j)) & C_i \text{ is a small cluster} \\ \text{dist}(p, C_i) & C_i \text{ is a large cluster} \end{cases}. \tag{3.10}$$

This modification also makes unweighted-CBLOF a global outlier detector since the outlierness is evaluated with regard to the whole dataset. In order to introduce the local density characteristic, the authors of [94] proposed Local Density Cluster-Based Outlier Factor (LDCOF), which uses the average distance of the data points within a cluster to the cluster center to normalize the outlier score:

$$\text{LDCOF}(p) = \begin{cases} \frac{\min(\text{dist}(p, C_j))}{\text{avg-dist}(C_j)} & C_i \text{ is a small cluster} \\ \frac{\text{dist}(p, C_i)}{\text{avg-dist}(C_i)} & C_i \text{ is a large cluster} \end{cases}, \tag{3.11}$$

where $p \in C_i$, and C_j is a large cluster that does not include p. The average distance of cluster members to the cluster center is defined as:

$$\text{avg-dist}(C) = \frac{\sum_{i \in C} \text{dist}(i, C)}{|C|}. \tag{3.12}$$

Note that both CBLOF and LDCOF have the incorporated clustering algorithm independent of the framework. But as suggested by [94], algorithms with a fixed number of clusters such as k-means are advantageous in performance and an overestimated number of clusters is recommended due to the potential non-spherical distributions.

Du et al. [28] devised a local outlier detection approach building upon the density peak clustering algorithm [95], which is a simple but effective density-based approach that can detect clusters of arbitrary shapes. The density peak clustering relies on two assumptions: (1) cluster centers have higher local density than surrounding data points; (2) cluster centers have a comparatively large distance to other data points with higher local density. The first assumption represents the concentration effect of a cluster, whereas the second assumption differentiates a cluster center and a nearby member in the same cluster. Two quantities are designed according to the two assumptions. The local density ρ for a data

point is defined as the number of neighbor data points within a cutoff radius. The δ for a data point is its minimum distance to another data point with higher local density. In the clustering process, data points with high δ and high ρ are first assigned cluster centers, then each of the remaining data points belongs to the same cluster where its nearest data point with higher local density is assigned. After the clustering phase, the outlier detection approach herein calculates the mean and the standard deviation of δ within each cluster. Moreover, Chebyshev's theorem [96] is used to decide the deviation threshold for outliers.

3.3.2 Projection-based Approaches

Many popular outlier detection techniques mentioned previously require the pairwise distance computation for the data points or the search for k-NN, which often incurs quadratic time complexity and makes those techniques hard to scale to very large datasets. In this section, we present approaches that use various projection techniques (e.g. random projection [97], LSH [45], etc.) to convert the original data into a new space with reduced dimensionality or complexity, while still preserving the proximity information (e.g., pairwise Euclidean distance, nearest-neighbor relations, etc.) of the original dataset to some degree. Then the outlier detection can be performed in the projected space with much-improved execution time.

Table 3.4 is a summary of the approaches introduced in this section. Many of them are extremely efficient and also applicable to high dimensional data. It is noteworthy that subspace techniques are also a type of straightforward projection. They have been widely used to address the challenges with high-dimensional data. Related techniques will be discussed further in Section 3.3.3.

Projection-indexed Nearest-neighbours (PINN) [98] is based on a random projection scheme to reduce the data dimensionality, and thus decrease the computation cost of determining the k-NN relations. The random projection scheme they adopted was developed by Achlioptas [97], and can approximately preserve the Euclidean distances for pairs of data points with high probability. Each of the randomly and independently generated entries from the projection matrix is defined as:

$$a_{ij} = \sqrt{s} \begin{cases} 1 & \text{with probability } \frac{1}{2s} \\ 0 & \text{with probability } 1 - \frac{1}{s} \\ -1 & \text{with probability } \frac{1}{2s} \end{cases}, \tag{3.13}$$

Table 3.4: Projection-based outlier detection

Algorithm	Projection Technique	Base Outlier Detector	Features	Scalability to High Dimensionality	Comments
PINN [98]	Random projection [97]	LOF [30]	Approximate k-NN	Good	Improved time efficiency & high accuracy
LSOD [49]	LSH [41, 45]	kth-NN distance	LSH-based ranking & pruning	Medium	Early detection of top outliers
Schubert et al. [99]	Space-filling curve [100]	LOF [30]	Approximate k-NN & ensemble	Poor	Near-linear complexity & easy distributed deployment
Loda [77]	Sparse random projection	Histogram-based outlier detector	One-dimensional histogram ensemble	Good	Linear complexity & high accuracy & handling missing values
Isolation Forest [80]	Binary tree	N/A	Ensemble & subsampling	Good	Linear complexity & high accuracy
Extended iForest [101]	Binary tree	N/A	Random hyperplane cuts	Good	Improved accuracy

where s is a parameter creating the effect that the random projection samples approximately $\frac{1}{s}$ of the entire feature space for each resulting projected feature. The advantage of random projections over other dimension reduction techniques such as PCA [102] is its efficiency. The authors of PINN further proved that the employed random projection could also preserve the k-distance of a data point and subsequently the neighborhood. These properties provide justification for their k-NN search in the projected space. The k-NN search is the most time-consuming component for many k-NN based outlier detection algorithms. With the dimensionality decreased, not only are less data involved in the computation, but also efficient indexing structures (e.g., [103, 104]) can be used to reduce the time complexity of k-NN search from $O(N^2)$ to $O(N \log N)$. Those indexing structures are not applicable in the case of high-dimensional data. After the approximate k-NN relations are determined, the data points are mapped back to the original space where the rest of the computation for LOF is conducted. To enhance the quality of the result, they maintain more than k nearest neighbors in the projected space, which are truncated to k for the computation in the original space.

Locality Sensitive Outlier Detection (LSOD) [49] leverages locality-sensitive hashing (LSH) [41, 45] to create an initial ranking of outliers. Locality-sensitive hashing (LSH) was first proposed by Indyk et al. [41] for the approximate nearest neighbors problem in the Euclidean space. The property of LSH functions is that they map similar data points to the same hash buckets with higher probability compared to those data points that are less

similar to each other. The LSH function adopted by LSOD was introduced by [45]:

$$h(v) = \lfloor \frac{a \cdot v + b}{w} \rfloor, \tag{3.14}$$

where v is a d-dimensional data point, a is a d-dimensional random vector, each entry of which is independently selected from a p-stable distribution [52], b is a random real number chosen from the uniform distribution bounded by $[0, w]$, and w is a real number parameter, called the "width" of the function. LSOD uses LSH to project the original data into one-dimensional hash values. These hash values are then segmented into multiple LSH buckets. Then LSOD generates the ranking of outlierness for a data point based merely on the number of points that are mapped into the same bucket. The assumption behind is that outliers tend to have less similar data points and thus end up in buckets with a small number of data points. To efficiently identify the top outliers, LSOD integrates a number of pruning strategies for distance-based outlier detection, including PPSN [86], ANNS [86] and PPSO [105]. Data points with a higher ranking are processed first, which results in high thresholds for these pruning strategies and thus greatly improves the efficiency. The final outlier score is the distance to the kth-NN.

Another outlier detection algorithm based on projection was proposed by Schubert et al. [99]. To tackle the approximate nearest neighbor search problem, they employed an ensemble of space-filling curves [100]. A space-filling curve maps a multi-dimension space into a single-dimension space. It has been widely used to develop indexing schemes for multi-dimensional data [106] and to perform similarity search in multi-dimensional space [107], etc. Based on the idea that diversity improves the accuracy of outlier ensembles, the proposed algorithm herein creates numerous space-filling curves by varying the characteristics of the space-filling curve, such as employing different curve families, using different sets of subspaces and shifting offsets. Then all the data points are projected to each of the created space-filling curves and the resulting one-dimensional values are sorted on each of the space-filling curves respectively. Based on the sorted sequence, a sliding window with a user specified width is used to produce candidates for each data instance on each individual curve. Finally, the candidates for each data point are merged together, and the k nearest ones are kept as the result. The authors argue that the space-filling curve is more suitable for k-NN search than other techniques, e.g., LSH [45] and random projection [108], due to the space-filling curve's preservation of closeness instead of distance or regions of a fixed size. Besides, they provided a distributed framework to scale the algorithm, where worker

nodes perform the space-filling curve projecting and send samples to the master node for distribution estimation. Also, note that the proposed approximate k-NN search scheme can be used to accelerate outlier detection that is based on k-NN and reverse k-NN in a general sense. They chose to instantiate it with LOF [30] in the experimentation, which relies on k-NN search to estimate proximity density.

Loda [77] employs a line of sparse random projections. Each of the projection maps data points to a one-dimensional space, based on which histograms are generated to estimate the probability for each data point. It is important to know that Loda follows the spirit of ensembling and demonstrates how multiple weak outlier detectors combined together into an ensemble can produce very good results. More specifically, each sparse random projection is performed by calculating the dot products of the data instances and a random vector of dimension $\sqrt{d}$, where d is the dimension of the input data space. This means only a randomly selected portion of the features are involved for each projection. The elements of the projection vector are independently and randomly drawn from $N(0,1)$. The rationale comes from the Johnson-Lindenstrauss lemma [109], which shows that such projection approximately preserves the pairwise L^2 norm distance (Euclidean distance) in the projected space. The histogram approximates the probability density of the projected one-dimensional data by discretizing the projected data into equal-width bins. The number of data points residing in a bin leads to the estimation of the probability of the bin. Sampling is often used to construct the histograms. The output of Loda for a data instance p is an average of the logarithm of the estimated probabilities on the projection vectors:

$$S\left(p\right) = -\frac{1}{k}\sum_{i=1}^{k}\log\left(f_i\left(p^T\ v_i\right)\right). \tag{3.15}$$

where f_i is the probability estimator of the i^{th} histogram and v_i is the corresponding projection vector. Loda can also handle missing variables for a data instance by taking into account only the histograms whose projection vector has a zero item on the place of that missing variable. Loda is not only very efficient but is also able to deliver high accuracy, thanks to the ensemble.

$$S\left(p\right) = -\frac{1}{k}\sum_{i=1}^{k}\log\left(f_i\left(p^T\ v_i\right)\right). \tag{3.16}$$

where f_i is the probability estimator of the i^{th} histogram and v_i is the corresponding projection vector. Loda can also handle missing variables for a data instance by taking

into account only the histograms whose projection vector has a zero item on the place of that missing variable. Loda is not only very efficient but is also able to deliver high accuracy, thanks to the ensemble.

At the end of this section, we introduce some tree-based approaches. In a broad sense, the construction of the tree models can also be viewed as a type of projection, where the original data points are mapped to specific tree nodes, and those tree nodes contain proximity information about the original data.

Liu et al. [80] developed Isolation Forest, which is an unsupervised tree ensemble that intuitively resembles the random forest for classification problems. The Isolation Forest consists of multiple Isolation Trees (iTrees), which can be viewed as the unsupervised counterpart of decision trees. An iTree model is generated with a given sample set by recursively choosing one random attribute and one random split value of the data on every tree node until the height limit is reached or the terminal leaf contains one distinct data instance. The intuition behind is that outliers have a higher chance of being isolated on an earlier stage than normal data instances. Therefore outliers are expected to have a shorter height in the isolation trees. Based on this idea, the outlier score of point p is defined as

$$Score(p) = 2^{-\left(\frac{\overline{d}(p)}{Ed(p)}\right)}, \tag{3.17}$$

where $\overline{d}\,(p)$ is the average depth of p in all the iTrees, and $Ed\,(p)$ is the expected length of the tree path for p. The latter is estimated based on the average length of the unsuccessful searches in the binary search tree. Isolation Forest is supposed to be constructed with small subsamples from the dataset rather than the entire dataset. Subsampling increases the diversity for the tree ensemble, which is beneficial for the accuracy of the result. Subsampling also helps alleviate or avoid the swamping (mistakenly identifying normal instances as outliers) and the masking (closely clustered outliers making themselves hard to be detected) issues. Another benefit of subsampling is the gain in efficiency since only a small portion of data is processed to build the model. After all, without having to deal with the pairwise distances, Isolation Forest is extremely efficient, with linear time complexity. Moreover, Isolation Forest also exhibits high detection accuracy, over a variety of datasets.

Hariri et al. [101] proposed Extended Isolation Forest to address the drawbacks of Isolation Forest. They provided an in-depth discussion about the limitations of axis-parallel cuts used in the original Isolation Forest, as well as on why the random hyperplanes benefit the algorithm. Extended Isolation Forest differs from Isolation Forest in that it

uses randomly generated hyperplanes involving multiple attributes to split the data and construct binary tree nodes, instead of using only one feature for each split. For each split, to determine whether a d-dimensional data point p should go to the left subtree or the right, the following equation is used:

$$(p - b) \cdot a \leq 0, \tag{3.18}$$

where b is the random intercept, each entry of which is drawn from a uniform distributed, bounded by the range of the corresponding attribute values of the data points in the tree node; a is a random vector deciding the slope of splitting, with each entry drawn from a normal distribution. Imagine the 2-dimension case where the separation can be visualized by lines. The splitting lines for Isolation Forest are all parallel to the coordinate axes, whereas those for Extended Isolation Forest have different angles. This flexibility in the slope makes Extended Isolation Forest capture the distribution and shapes better than Isolation Forest. Consider a specific 2-dimension example with two dense clusters: one on the top left corner and the other on the bottom right corner. Dense cuts will be made over the clusters. Since the Isolation Forest uses cuts that are parallel to the axes, this could easily create "densely cut areas" on the top right corner and the bottom left corner, which are unwanted artifacts. These two artifact areas will make the algorithm mistakenly assign low outlier scores for outliers within them. In contrast, Extended Isolation Forest is less likely to create such artifacts due to the variety of splitting slopes for separating the data.

3.3.3 High-dimensional Outlier Detection

As summarized by Zimek et al. [110], the challenges for outlier detection in high-dimensional data are twofold: the efficiency aspect and the effectiveness aspect. The difficulty in achieving efficiency with high dimensional data is mainly attributed to two reasons. First, the similarity search such as k-NN search becomes more expensive in terms of computation cost because of the increased dimensions. Second, some techniques used to accelerate the outlier detection such as sampling [111, 112], pruning [113], ranking strategies [84, 114] and efficient indexing structures (R-trees [103], X-trees [104], etc.) degrade significantly or even introduce almost no improvement with high-dimensional data.

For the effectiveness aspect, the concern is whether the outlier detection method can identify meaningful outliers. A frequently used term related to this problem is the "curse

of dimensionality" [110, 115–117]. It refers to the dilemma that in the high-dimensional space, the detection of outliers based on deviation tends to be interfered by a phenomenon called "distance concentration": the distances for all pairs of data points tend to become almost uniform. Thus all the regions in the dataset become almost equally sparse, and the distinction between outliers and normal instances is hard to capture. This phenomenon is caused by the dilution effects of a large number of "normally noisy" irrelevant dimensions/attributes [89]. In other words, these irrelevant dimensions conceal the information that can be used to identify outliers. This section focuses on approaches that are designed to tackle one or both of the challenging aspects of outlier detection in high-dimensional data. A summary of these approaches is presented in Table 3.5.

Table 3.5: Outlier detection for high-dimensional data

Algorithm	Features	Improve efficiency/effectiveness	Subspace	Comments
RBRP [118]	Recursive partitioning & approximate k-NN	Efficiency	No	Fast search of approximate k-NN
PINN [98]	Random projection & dimension reduction	Efficiency	No	Improved efficiency but approximate results
ABOD [2]	Angle variance	Effectiveness	No	High time complexity
Kriegel et al. [119]	Axis-parallel hyperplane	Effectiveness	Yes	Intepretability of result & High accuracy
HiCS [120]	Statistical test & ensemble	Effectiveness	Yes	Improved accuracy & generalized pre-processing method
RS-Hash [121]	Randomized hashing & ensemble	Efficiency & Effectiveness	Yes	Linear complexity & high accuracy & intepretability of results

To improve the efficiency of outlier detection for high dimensional data, Ghoting et al. [118] proposed Recursive Binning and Re-projection (RBRP). RBRP is inspired by ORCA [113], a nested loop outlier detection approach whose outlier score is based on the distance to the kth nearest neighbor. In order to take advantage of the pruning scheme by ORCA, k approximate nearest neighbors need to be found. RBRP uses a recursive binning process to accelerate the search for such approximate k-NN. First, the data points are recursively partitioned into bins until the size of an individual bin is smaller than a pre-defined threshold. This recursive partitioning strategy resembles divisive hierarchical clustering. More specifically, for each recursion of the partitioning, k-means is adopted to create k partitions so that data points closer to each other in distance have a high probability of being assigned to the same bin. After the recursive partitioning phase, RBRP searches for k approximate nearest neighbors in the generated bins, where the data points are ordered as per their projection along the principle component to accelerate the

search.

Note that the Projection-indexed Nearest-neighbours (PINN) [98] algorithm previously mentioned in Section 3.3.2 also aims at improving the efficiency of high-dimensional outlier detection. PINN leverages random projection for dimension reduction and uses approximate k-NN to deliver efficient performance.

Many more works in the literature focus on the effectiveness aspect of the high-dimensional outlier detection problem.

Kriegel et al. [2] introduced an angle-based outlier detection method (ABOD) to address the issue of deteriorating quality encountered by Euclidean-distance-based algorithms in the face of high-dimensional datasets. As illustrated by Figure 3.2, the intuition behind ABOD is that if a data point is far from the rest of the data points (e.g., o), the angles having such a data point as the vertex (e.g., $\angle poq$ and $\angle jok$) tend to be small and vary slightly. In contrast, if a data point (e.g., i) is closely surrounded by others or is within a cluster, such angles (e.g., $\angle piq$ and $\angle jik$) usually have a high variance. Therefore, the outlier score for a data point relies on the variance of the angles having that data point as the vertex, weighted by the distances to the pair of other data points. The authors stress the importance of the distance weighting because naturally the angle to two data points varies more widely with a bigger distance. More specifically, the proposed angle-based outlier factor (ABOF) for data point i is defined as

$$
\begin{aligned}
ABOF(i) &= VAR_{p,q \in D} \left(\frac{\overrightarrow{ip} \cdot \overrightarrow{iq}}{||\overrightarrow{ip}||^2 ||\overrightarrow{iq}||^2} \right) \\
&= \frac{\sum_{p \in D} \sum_{q \in D} \frac{1}{||\overrightarrow{ip}|| ||\overrightarrow{iq}||} \left(\frac{\overrightarrow{ip} \cdot \overrightarrow{iq}}{||\overrightarrow{ip}||^2 ||\overrightarrow{iq}||^2} \right)^2}{\sum_{p \in D} \sum_{q \in D} \frac{1}{||\overrightarrow{ip}|| ||\overrightarrow{iq}||}} - \left(\frac{\sum_{p \in D} \sum_{q \in D} \frac{1}{||\overrightarrow{ip}|| ||\overrightarrow{iq}||} \frac{\overrightarrow{ip} \cdot \overrightarrow{iq}}{||\overrightarrow{ip}||^2 ||\overrightarrow{iq}||^2}}{\sum_{p \in D} \sum_{q \in D} \frac{1}{||\overrightarrow{ip}|| ||\overrightarrow{iq}||}} \right)^2,
\end{aligned}
$$

where D is the dataset, the dot represents the dot product between two vectors. Since the outlier score for each data instance involves all the pairwise combinations of other data instances, this incurs the expensive $O(n^3)$ time complexity. To reduce the time complexity, two approximate variants were introduced: FastABOD and LB-ABOD. FastABOD constricts the selection of the pairs of data points for the outlier score computation to the k-NN of the data point. LB-ABOD is presented as a lower bound approximation of ABOD, which is designed to obtain the top outliers with the highest scores efficiently.

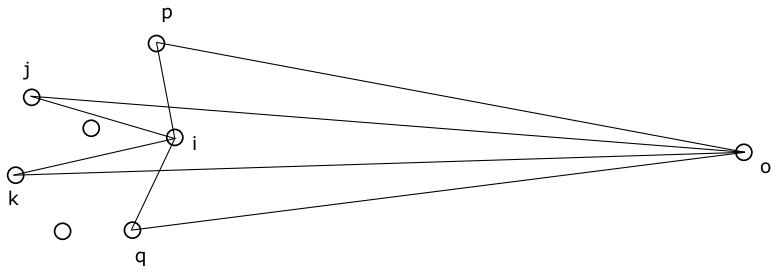


Figure 3.2: The intuition of ABOD [2]

In addition, many works explore solutions on subspaces to handle the effect of the "curse of dimensionality", assuming that only a subset of the attributes is relevant for the discovery of meaningful outliers despite the rest of the attributes being noise. Kriegel et al. [119] developed an outlier detection schema which evaluates outlierness based on the deviation of an individual data point from the axis-parallel hyperplane spanned by a set of reference points. The hyperplane spanned by a set of points is associated with the subspace where these data points have high variance. The reference set for a data point is selected by ranking the number of shared nearest neighbors with the data point, based on the assumption that even though the traditional concept of k-NN loses its meaning in high-dimensional data, two data points generated by a similar mechanism still share a considerable number of nearest neighbors despite the irrelevant attributes. Note that the proposed method herein customizes the subspace for each data point because of the way how the hyperplane is created. Thus, the explanation for reasons of outlierness can be provided according to the related subspace.

Based on the assumption that outliers in high dimensional data are hidden in multiple subspaces that exhibit non-uniformity and high contrast, Keller et al. [120] proposed a way of measuring the contrast of subspaces and accordingly a subspace search method called High Contrast Subspaces (HiCS). The contrast quantification for a candidate subspace is performed by sampling and statistical tests in an iterative way. In each iteration, a random attribute is chosen, and a random rectangular region in the subspace is generated. Then the deviation value is computed by comparing the marginal probability and the conditional probability. The final contrast value for a subspace is obtained by combining the individual deviation values. Based on the contrast quantification method, high-contrast subspace

candidates are produced in a fashion that resembles the famous Apriori algorithm [122]. Starting from 2 dimensions, subspaces with a contrast value over a pre-defined threshold will be used for the candidate generation of the current dimension plus one. Then a pruning step that removes redundant subspaces ensues. As for the final outlier score, the results over different subspaces are combined, which resembles the feature bagging technique [31]. However, HiCS discriminatively selects subspaces, whereas such selection in feature bagging is random. In the paper, LOF is used as the actual outlier detector. However, the choice of the outlier detector is independent, and thus HiCS can be viewed as a generalized pre-processing technique for high-dimensional outlier detection.

Sathe et al. [121] proposed RS-Hash, an extremely efficient and accurate subspace outlier detection approach based on randomized hashing. RS-Hash follows the spirit of ensembling and averages the scores of all the ensemble component as the final score. Each component is essentially a set of models based on the closed hash function. Those models are trained by a sample of the original dataset, through a variety of randomized transformations and normalizations, coupled with randomized selections of the subspace. The outlier score for a data point output by an individual ensemble component is based on the number of sampled data points falling in the same hash bin during the training phase. Naturally, a low count in such bins indicates outlierness. Intuitively, RS-Hash estimates the density of the rectangular regions for a given data point, over different subspaces. Due to the randomization, the rectangular regions evaluated for a data point in different ensemble components are of different sizes, which is important to the accuracy of the ensemble. Similar to the approach proposed by Kriegel et al. [119], RS-Hash also provides insights for the reason of a data point being an outlier, by analyzing the related subspaces that result in low scores. With linear time complexity, RS-Hash is considered a very efficient algorithm. Moreover, due to the use of subspace in the models, RS-Hash is also effective at handling the "curse of dimensionality".

Outlier detection for high-dimensional data is still a challenging problem due to the concerns about efficiency and effectiveness. Plenty of methods address either aspect or both with techniques such as approximate k-NN, subspace, and ensemble. Subspace-based approaches have recently received much attention in the research community. An inevitable problem to consider is how to identify the most meaningful and useful subspaces while minimizing the associated computational cost, given that the number of possible combinations of different attributes can be enormous.

In addition to the aforementioned methods, other interesting works in recent years

include: HighDOD [123], using a dynamic subspace search method with a sample-based learning process; LODES [124], which relies on a novel local-density-based spectral embedding to identify outliers in nonlinear subspaces; RAMODO [125], which uses representation learning to reduce the dimensionality, and combines it with the random distance-based approach [27].

3.3.4 Outlier Detection in Data Streams

A data stream is a continuous and unbounded sequence of data in large volumes. Outlier detection in the context of data streams faces two major challenges. The first one is the storage memory challenge. As the data points continuously arrive and the sequence is theoretically endless, it is often not feasible to store the entire stream in the memory from the very beginning. Besides, the on-the-fly property of many outlier detection applications (e.g., intrusion detection in a computer network, suspicious behavior detection in wireless sensor network) imposes efficiency requirements.

To address these challenges, a commonly used technique is windowing: using a segment of the data stream, usually the newest one, to build incremental models and update the models in response to the change of involved data points. As summarized in [126], there are four types of windowing techniques:

- Landmark window: A specific point in the data stream is fixed as the landmark. The outlier detection algorithm takes into account the sequence of data between the landmark and the current data point. Since the size of the data involved in the processing increases over time, memory storage becomes a major issue.

- Sliding window: A window of a fixed width w is sliding over the data stream. In other words, only the latest w data points are used as the context of outlier detection. Based on the definition of the window width, there are two types of sliding-window: count-based window and time-based window. The count-based window uses a fixed number of data points as the window width whereas the time-based window uses a fixed time duration.

- Damped window: A weight is assigned to each data point depending on the timing or order of its arrival. Usually, newer data points have higher weights so that the detection results can reflect the most recent trends.

- Adaptive window: Adaptive window is similar to the sliding window except that the window width w varies according to the rate of change from the data within the current window. The window expands when the data remain stationary and constricts when a change of the data is observed [127].

3.3.4.1 Distance-based Outlier Detection in Data Streams

A number of works in the literature have applied the distance-based outlier detection algorithm proposed by Knorr et al. [88] to the data stream scenarios. They adopt the same criterion for determining outliers as in [88]: a data instance has less than p neighbors within a radius of r. This definition allows for unsupervised outlier detection without any assumptions on the distribution of the dataset. Table 3.6 summarizes the distance-based outlier detection approaches mentioned above.

Table 3.6: Distance-based outlier detection in data streams

Algorithm	Features	Memory Complexity	Outlier Storage	Count-based /Time-based Sliding Window	Comments
STORM [128]	Safe inliers	$O(Wk)$	None	Count-based	High memory usage & high time complexity
Abstract-C [129]	Pre-handling neighbor counts	$O(W^2/S + W)$	None	Both	Imrpoved execution time
DUE [130]	Event queue	$O(Wk)$	Outlier list	Both	Efficient at re-evaluating outlier at runtime
MCOD [130]	Event queue & Micro-cluster	$O(cW + (1-c)kW)$	Outlier list	Both	Improved execution time & improved memory complexity
Thresh_LEAP [131]	Seperate index per slide & Minimal probing	$O(W^2/S)$	Outlier candidate list per slide	Both	Improved execution time but potentially more memory usuage

[1] k denotes the number of neighbors (as in k-NN).
[2] W denotes the size of the window.
[3] S denotes the slide size (the number of data points inserted and deleted for each time of sliding).
[4] c denotes the fraction of data points that are included in the micro-clusters.

Angiulli et al. [128] proposed a sliding-window-based approach called STORM to respond to outlier queries regarding the current window of data objects. STORM maintains a data structure called Indexed Stream Buffer to store the data instances in the current window and the related information of their neighbors. A neighbor of data instance i here is defined as another data instance whose distance to i is no bigger than the radius r. In Indexed Stream Buffer, each data point is associated with a list of neighbors that arrive before that data point, as well as the count of neighbors that arrive after that data point.

An important property of this problem is identified that if a data instance has more than p neighbors succeeding it, it is guaranteed to be an inlier during its presence in the window. This property is used to develop two approximations in case of limited memory that is not capable of holding the entire window. The first approximation is to only keep a fraction of such guaranteed inliers in the window. The second approximation consists in reducing the size of the preceding neighbors for each data instance in the window. They conducted formal analysis on the approximation error bounds and proved the statistical guarantees of the approximations.

Yang et al. [129] developed another outlier detection approach named Abstract-C for data streams. They identified the challenge of pattern detection for data streams: the expiration of data points gives rise to complex changes in the existing patterns. More specifically, for the neighbor-based outlier detection, the problem becomes how to update the neighbor counts when data points expire due to the window sliding, without maintaining an exact neighbor list for each data point but only counts of neighbors. The provided solution takes advantage of the "predictability" of the neighbor counts for the succeeding windows. Abstract-C calculates the lifespan of each data point and preserves that information when updating the neighbor counts for that data point. Each data point is associated with its future neighbor counts for the next few window slides. In other words, the expiration of the data points is pre-handled, and no updating related to neighbor counts is required when they are eliminated from the sliding window. Abstract-C is more efficient and takes less space compared to STORM.

Kontaki et.al [130] introduced DUE and MCOD, event-based outlier detection approaches for data streams. DUE reacts to the expiration events to update the neighbor lists of only the related data points. DUE maintains a priority queue, named event queue, to store the unsafe inliers, which are those with less than k succeeding neighbors. The priority in the queue is determined by the earliest expiration time of the preceding neighbors. Also, an outlier list is maintained to keep track of all the current outliers. When the window slides, expired data instances trigger events to update the neighbor information of the unsafe inliers in the event queue. Some of them may become outliers and be moved to the outlier list. For newly added data points, a neighbor search is performed, and the resulting preceding and succeeding neighbor lists are created for each of them. Then they may be put into the outlier list or inlier list according to the number of their neighbors. The neighbors of the newly added data points also update their neighbor list, and they may be moved from the current queue depending on the neighbor counts. DUE is efficient

at adjusting the outlier evaluation in response to the window sliding due to the event-based mechanism. However, the event queue takes extra time and memory to maintain the desired sorted order.

The specialty of MCOD [130] is its employment of micro-clusters to reduce the number of range query searches, i.e., searching for the neighbors of a data point within a radius. A micro-cluster is centered by one data point with a radius of $r/2$, comprising at least $k+1$ member points. The data instances belonging to such micro-clusters are guaranteed to be inliers because every pair of data points has a distance of less than r, due to the triangular inequality of the distance metric. When the window slides, expired data points in the micro-clusters are removed. This can cause the dismission of a micro-cluster if it has less than $k+1$ members after the removal of the expired data points. In such cases, the remaining data points are processed as newly added ones. New data points can be added to existing micro-clusters if they are within the distance. New points can also form a new micro-cluster after range query searches among the "free" points that are not included in any micro-clusters if there are at least k neighbors within the $r/2$-radius area of the new point. Aside from the points in micro-clusters, those "free" points are treated differently because they may have outliers and unsafe inliers. An event queue as in DUE [130] is maintained to manage the unsafe inliers. MCOD is advantageous in execution time thanks to the reduction of the neighbor searches. MCOD also needs less memory space since the points inside a micro-cluster do not need an extra neighbor list.

Cao et al. [131] proposed an approach called Thresh_LEAP to tackle the problem of distance-based outlier detection for data streams. Thresh_LEAP takes advantage of the temporal relationships among the individual slides and treats a slide as a unit for the neighbor counting. To be more specific, each data point maintains a list keeping track of the number of neighbors in each slide. Reciprocally, each slide has a trigger list storing the data points that will be affected when the slide expires. A strategy called "minimal probing principle" is adopted, through which the neighbors in the same slide are searched first, then the neighbors in the preceding slides are explored slide by slide from newest to oldest. The probing stops as soon as more than k neighbors are found. When a slide expires, the data points in the trigger are re-evaluated. If a data point has less than k neighbors due to the expiration, succeeding slides will be probed for the data point while preceding slides must have been probed already. The "minimal probing principle" as well as indexing each slide individually give Thresh_LEAP an advantage in CPU time, over other distance-based outlier detection methods except MCOD, which uses micro-clusters.

However, considerable memory cost will be incurred, especially when small slide size creates a huge number of slides per window.

Even though the distance-based techniques are easy to understand and computationally efficient for data streams, they also have limitations. First, it is tricky to find the appropriate values of parameters r and p for different datasets. Also, it assumes homogeneous densities in the entire dataset. However, for real-life datasets, approaches like LOF [30] that address local outliers may produce better results.

3.3.4.2 Density-based Outlier Detection in Data Streams

In this section, we introduce several outlier detection algorithms in data streams that are based on the density of the data points with regard to the k-NN. All of these approaches are extended from the LOF algorithm [30]. The distance-based approaches introduced in the previous section are considered to be able to detect global outliers since they assume homogeneous densities across the dataset. In contrast, as previously discussed in Section 3.3.1.1, LOF usually achieves good performance in datasets with non-homogeneous densities. This property also holds when it is used for data streams.Table 3.7 summarizes the density-based approaches for data streams introduced in this section.

Table 3.7: Density-based outlier detection in data streams

Algorithm	Features	Window Type	Time Complexity	Memory Complexity	Comments
Incremental LOF [132]	Selectively updating records	Landmark window	$O(N \log N)$	$O(Nk)$	High memory comlexity & high time complexity
MiLOF [133]	c-means-based summarization	Sliding window	$O(N \log W)$	$O(Wk)$	Low memory comlexity & time complexity but compromised accuracy
DILOF [134]	Density-based summarization	Sliding window	$O(NW)$	$O(Wk)$	Low memory comlexity & time complexity & high accuracy

[1] N denotes the size of the overall data stream.
[2] W denotes the width of the window.

Pokrajac et al. [132] proposed the first incremental version of LOF [30] for data streams. The Incremental LOF aims at delivering equivalent performance as applying the original LOF repeatedly on the data streams every time when a new data instance is received but with significantly less execution time. Inserting new data points and deleting obsolete data points (due to memory constraints or particular outdated behaviors) are followed by updating the records (k-distance, LRD, LOF score, etc.) of existing data points. Incre-

mental LOF is based on an important observation that the insertion and deletion can only potentially affect a limited number of data points. To be more specific, the insertion or deletion of a data instance i influences the k-distances of the k-reverse-nearest-neighbors (k-RNN) of i. k-RNN of i is defined as the set of data points that have i as one of their k-NN. The change of the k-distances leads to the change of reachability distances and thus the LRDs of i's k-RNN's k-RNN, whose LOF scores need to be modified accordingly. They proved that the maximal number of k-RNN of a data point is proportional to k and exponentially proportional to the number of dimensions. Thus, if efficient approaches for k-NN and k-RNN queries (with time complexity $O(\log N)$) are applied, the overall time complexity of incremental LOF for the entire data stream of size N is merely $O(N \log N)$, if k and the dimensionality are treated as constants.

Salehi et al. [133] proposed MiLOF to overcome the unbounded memory issue of Incremental LOF by creating summarizations of previous data points, which leads to a fixed memory bound. MiLOF divides the available memory storage into two parts: one part for the newest b data points in the original form and the rest for c summaries of obsolete data points. Whenever the memory is running out, the older half of the b data instances are summarized and then removed from the memory. The summarization is performed using c-means clustering [92]. The cluster centers are chosen as the representatives for the clusters they belong to. These cluster centers also participate in the LOF score calculation for the incoming data points as regular data points. However, the LOF-related records (k-distance, reachability distance, LRD, and LOF score) associated with these cluster centers are not computed based on their k-NN but based on the clusters they represent. To produce more accurate results, they introduced a flexible c-means which selectively summarizes the regions that are less likely to contain outliers, therefore, the regions with a higher probability of containing outliers are preserved in the original form. In order to fix the memory bound, the summaries in the form of cluster centers are merged in the same frequency of summarization so that only one single set of cluster centers exists. The merging is carried out with a clustering algorithm which weights the cluster centers according to the number of data points of the cluster. Subsequently, the updating of the LOF-related records for the merged cluster centers ensues. The insertion of incoming data points of MiLOF is similar to Incremental LOF but with modifications that handle the cases when the cluster centers appear in the k-NN of the incoming data points. MiLOF successfully reduces the memory consumption to a user-specified bound and decreases the time complexity accordingly. However, the accuracy is inevitably compromised due to the

summarization, which may not effectively preserve the density of the data instances.

Another LOF-based outlier detection algorithm for data streams called DILOF was proposed by Na et al. [134]. DILOF also addresses the enormous memory consumption issue of Incremental LOF and additionally provides a solution for detecting a long sequence of outliers. Different from MiLOF, which uses c-means clustering for summarization, DILOF adopts a novel density-based sampling scheme to summarize the data, without prior assumptions on the data distribution. Thanks to this summarization technique, DILOF is shown to outperform MiLOF in detection accuracy. Similar to MiLOF, DILOF has a detection phase and a summarization phase. In the summarization phase, a window of size W is maintained. As soon as the window is filled with data, $W/4$ out of the oldest $W/2$ data points are selected according to a proposed density-based sampling algorithm while the unselected points are removed to free up space of $W/4$. The goal of the summarization phase is to select the data points whose density resembles the original data to the highest possible degree. To achieve this end, they defined this task as a combinatorial optimization problem by extending a non-parametric Renyi divergence estimator [135] and converted the problem into a solvable binary constrained optimization problem. Then they introduced a new component for the objective function in order to preserve the data distribution. Furthermore, they developed a heuristic distance approximation technique, which was shown to greatly accelerate the summarization phase while still preserving the detection accuracy. As for the detection phase, they adopted the same method by Incremental LOF, which only updates the data points that are influenced when insertion or deletion happens. Moreover, they introduced a strategy called the skipping scheme to detect a group of outliers that comes in the form of long sequences. The skipping scheme shortcuts the detection process when a data point is found to be part of an outlier sequence. The assumption underlying the skipping scheme is that sequence outliers arrive consecutively, whereas the members of an emerging new class (considered to be normal data points) come in alternately with inliers.

MiLOF and DILOF have both managed to overcome the limitations of memory and execution time in Incremental LOF by summarizing a portion of the data points, which allows for keeping only a limited number of data points in the memory. Thanks to a better summarization technique, DILOF tends to outperform MiLOF in terms of accuracy measured by AUC. On the other hand, MiLOF seems to beat DILOF in time complexity. However, in the experiments of [134], DILOF outperforms MiLOF in terms of execution time. This is because we treat the parameters related to c-means (the maximum number of

iterations, number of cluster centers, etc.) as constants. In practice, when window width W is comparatively small to the c-means-related parameters, DILOF tends to outperform MiLOF.

3.3.4.3 Clustering-based Outlier Detection in Data Streams

As mentioned in Section 3.3.1.2, clustering-based approaches are advantageous over distance-based and density-based outlier detection in terms of time complexity. However, the granularity of outlier analysis is sacrificed. It is also noteworthy that the performance and the property of the outlier detection techniques depend heavily on the underlying clustering algorithms. For instance, a k-means-based approach may not be able to identify outliers in the context of arbitrary-shaped clusters. In the setting of data streams, new challenges include ensuring scalability, devising incremental strategies, etc. In this section, we discuss outlier detection approaches for data streams that are based on clustering. We summarize the characteristics of these approaches in Table 3.8.

Table 3.8: Clustering-based outlier detection in data streams

Algorithm	Features	Window Type	Score Based on	Predefined Cluster Number	Comments
D-Stream [136]	Grid-based clustering	Damped window	Density of a grid	No	Arbitrary-shaped clusters but poor scalability to high dimension
Elahi et al. [137]	k-means & delayed determination of outliers	Non-overlapping sliding window	Distance to cluster centers	Yes	Assuming spherical clusters & too many parameters
AnyOut [138]	Hierarchical clustering & ClusTree [139]	Damped window	Distance to cluster centers & Gaussian probability density	No	Real-time & varying granularity but assuming spherical clusters
Salehi et al. [140]	Hyperellipsoidal clustering	Non-overlapping sliding window	Whether belongs to a cluster	No	Time efficient & addressing switching data streams
Chenaghlou et al. [141]	Hyperellipsoidal clustering & Gaussian clusters	Non-overlapping sliding window	Gaussian probability density	No	Time efficient & addressing emerging distributions

Chen et al. [136] introduced a grid-based approach called D-Stream for data stream clustering, which can also be used to detect and remove outliers. On the one hand, D-Stream maps the incoming data points to grids in an online fashion. On the other hand, D-Stream periodically forms clusters from the grids and eliminates the so-called "sporadic grids" that are considered to be outliers. Concretely, each existing data point is associated with a density coefficient, which decays with the elapse of time, in order to capture the

dynamic changes of the data stream. The density of a grid is defined as the sum of the density coefficients of the data points residing in the grid. The grid density is updated when new data points are mapped to it. Due to the property of the density coefficient's decay factor, it is adequate to just maintain a characteristic vector for each grid rather than keeping track of all the density coefficients of the data points. In order to group the grids into clusters, the grids are classified into dense grids, sparse grids, and transitional grids according to the grid density. Based on the classification of the grids, a cluster is formed by connecting a group of dense grids that are surrounded by sparser grids. Due to the decaying factor, the class of a grid may change over time, which leads to the dynamic changes of the clusters. Therefore, the cluster structure needs to be periodically adjusted. With the assumption that outliers are mapped to grids with very few data points, they developed a threshold function to detect the grids with a density under a certain value. Note that the decaying factor can also result in low grid density even if the grid has a decent number of data points. This type of grid is not expected to be removed as outliers. Therefore, the threshold function is designed to distinguish this case from a grid having very few data points.

Elahi et al. [137] proposed an outlier detection algorithm based on k-means clustering for data streams. The data stream to be processed is treated as chunks of data, which is essentially the non-overlapping sliding window model. There are two sets of cluster centers maintained throughout the stream processing. One set is called the actual cluster centers, which are the output of k-means clustering based on merely the current chunk of data. The other set is called the updated cluster centers, which are initiated as the average of previous updated cluster centers and current actual cluster centers, then run through k-means using both the current data chunk and candidate outliers. The candidate outliers are determined based on the distance of a data point to its updated cluster center. In order to address the scenario of pioneering members of an emerging cluster being falsely treated as outliers, the algorithm withholds the candidate outliers for L (user-specified parameter) chunks of data processing. The outlier scores for a candidate are accumulated during the course of L chunks, after which the candidate is finally judged to be an outlier or not. Therefore, only the cluster centers and candidate outliers are being held in memory while the data points considered to be safe are discarded. This strategy greatly reduces memory consumption. Even though this approach is intuitive and efficient, it has limitations such as requiring the determination of multiple parameters (k, L, outlier threshold, etc.) and assuming the spherical shape of clusters due to the use of k-means clustering.

Assent et al. [138] proposed an anytime outlier detection method called AnyOut, which leverages a tree structure built as the result of hierarchical clustering to represent the data and determine the outlier scores for the incoming data points in new window slides. AnyOut establishes a tree structure called ClusTree, which was developed by Kranen et al. [139] originally for parameter-free hierarchical clustering of data streams. Each tree node in ClusTree compactly represents a cluster by use of cluster features, which is a tuple composed of the number of the data points in the cluster, the linear sum of these points, and the squared sum of them. ClusTree is capable of updating the cluster features when new data points join the model. ClusTree also has additional buffer entries that allow for the anytime insertion of clusters. When it comes to building a ClusTree, they adopted a top-down bulk loading method [142], uses the expectation maximization [143] algorithm. AnyOut emphasizes the real-time characteristic of itself. With more time given, it outputs a finer-grained and more accurate score. This is achieved by a top-down outlier assessment strategy, in which the data point finds its closest cluster at each level. When the next data point arrives, and the result for the current data point must be returned, the outlier score is computed based on its relation with the most recently found cluster in the tree. Two ways to calculate the outlier score are provided. The first is called mean outlier score, defined as the distance between the data point and the mean of the entries in the cluster (resembling the centroids in k-means [92]). The second way is based on the Gaussian probability density of the data point.

Salehi et al. [140] introduced a weighted ensemble framework to detect outliers in data streams, which uses a clustering algorithm to model the normal patterns of data instances in previous windows. Their approach addresses the scenario where a data stream alternates between different states, each of which potentially consists of multiple distributions. The proposed framework comprises three components. First, all data points in the current window are clustered by the HyCARCE clustering algorithm [144]. HyCARCE is a density-based hyperellipsoidal clustering algorithm without predefined cluster numbers. HyCARCE outputs a set of cluster boundaries, which can be viewed as the built "clustering model". Every window is clustered, and their clustering models are kept in memory for the computation of the ensemble score for the incoming data points. The second component of the framework is to define the similarity between two clustering models, from which the ensemble weight is derived. To this end, the focal distance [145] between two hyperellipsoids is adopted. To be more specific, for two clustering models, the distance of every pair of hyperellipsoid boundaries, each from a different clustering model, is first

computed. Then pairs of boundaries are selected out in a greedy fashion, starting from the shortest distance. In the end, the reciprocal of the sum of the resulting pairs' distances is used as the similarity between two clustering models. The third component of the framework is to calculate the outlier score for a data point in the new window with the ensemble model, based on the relationship between the data point and previous clustering models. Specifically speaking, the algorithm checks if that data point belongs to any cluster for each clustering model in the history, based on whether the Mahalanobis distance between the data point and the cluster hyperellipsoid is beyond a threshold. The check produces a binary score for each previous clustering model. Then the final outlier score is the weighted sum of those binary scores, the weight being the similarity between the current clustering model and the corresponding former clustering model.

Another data stream outlier detection approach based on the HyCARCE clustering algorithm [144] was proposed by Chenaghlou et al. [141]. Different from the method presented in [140], their approach models normal data patterns using Gaussian clusters and derives the outlier score based on the Gaussian probability density function of a cluster. Besides, the proposed approach is aware of and handles the newly emerging clusters. To process the data points in a new window, the first stage of the proposed approach is to find out if existing Gaussian clusters can explain the underlying distribution of some of the data points. To this end, they created two criteria: 1) the number of data points in the new window fitting the cluster must not be too few, which is tested by Cumulative Binomial Probability (CBP) function; 2) the data points must spread out the cluster, which is tested by transforming the data points into standard Gaussian distributions [146], then into a spherical coordinate system [147]. The second stage is to detect potential emerging Gaussian clusters by using CBP, after removing the data points that can be explained by existing models in the first stage. If the result is positive, HyCARCE clustering is employed to cluster these data points and save the new model. Finally, the score of a data point is the maximum value among the probabilities of the data point being observed under each of the Gaussian clusters.

3.3.5 Distributed Outlier Detection

Distributed systems and algorithms (e.g., [148]) are not a completely new topic. In a variety of domains, there have been a considerable amount of research on distributed systems, e.g., distributed simulation [149–154], wireless ad-hoc networks [155–166], vehicular ad-

hoc networks [167–172], mobile distributed multimedia systems [173, 174], etc.

With the advent of the big data era, distributed algorithms for data mining and machine learning are especially in high demand. This is because traditional centralized data mining and machine learning methods fall short for a few reasons. First, the resources of an individual computer may not be sufficient to perform the computation tasks due to limitations in disk storage, memory, CPU, etc. Second, the centralized algorithms may not be able to satisfy the rigid time constraints required by many modern applications, e.g., real-time big data analytic applications. Moreover, the datasets themselves are tending to become more and more distributed.

In this section, we talk about recently proposed outlier detection approaches that function in a distributed manner to address the big data challenge. Table 3.9 summarizes the approaches discussed in this section. A challenging task for extending outlier detection to the distributed setting is minimizing the communication overhead while still guaranteeing the accuracy. This task is especially difficult for methods that require the computation of pairwise distances between data points. It is worth noting that in addition to the algorithms to be introduced subsequently, some works focusing on distributed k-NN search [175–179] can be inspirational for the development of distributed k-NN-based outlier detection algorithms.

Table 3.9: Distributed outlier detection

Algorithm	Base Outlier Detector	Distributed Infrastructure	Features	Comments
Bhaduri et al. [180]	ORCA [113]	Network of ring topology	Top-N pruning	High communication overhead
Angiulli et al. [181]	Solvingset [182]	Ethernet network with TCP sockets	Top-N pruning	Impaired scalability due to broadcasting
DLOF [36]	LOF [30]	Hadoop MapReduce	Grid-based partitioning & duplication reduction	Reduced comminication overhead but not scalable to high dimensionality
DTOLF [183]	LOF [30]	Hadoop MapReduce	Grid-based partitioning & top-N pruning	Reduced comminication overhead but not scalable to high dimensionality
OW-OCRF [24]	One-class random forest [184]	Wireless sensor network	Weighted ensemble	Real-time response & high accuracy

Bhaduri et al. [180] developed DOoR, a distributed solution for the ORCA method [113], which uses the kth nearest neighbor distance as the outlier score and has a simple pruning rule. DOoR operates in a cluster of machines connected with a ring overlay topology, with an additional central machine that connects to all the machines in the ring. The central node maintains a list of current top-N points and the largest kth nearest neighbor distances

as the cutoff threshold for pruning. Whenever the threshold information is updated in the central node, it will be broadcast to every machine in the ring topology for the most effective pruning. Each worker node in the ring contains a partition of the data. A worker node receives data partition from its previous node and validates the outlierness against its own local data partition, then passes it to the next node. After being passed around a circle, those data points not pruned will be sent to the central node for further evaluation. Essentially, all the data points not pruned are broadcast across the cluster, which incurs possibly high communication cost in the network and thus is not scalable to a very large scale of datasets.

Angiulli et al. [181] extended the SolvingSet algorithm [182] to the distributed environment. The solving set is an iteratively expanding sample set, based on which every data instance outside the set estimates their approximate k-NN in each iteration. A top-N outlier score is maintained and updated so that non-outliers can be pruned in advance. In the distributed setting, the solving set is broadcast across the cluster. The method also consists of the parallel computation and the synchronization of the partial results. This approach falls short in case of big datasets due to the correspondingly increasing size of the solving set to be broadcast.

Bai et al. [185] proposed a distributed solution for LOF. A grid-based partitioning method that tries to balance the workload and allocate the adjacent grids to the same machines is adopted. Based on the relation between the local k-distance and the grid borders, data instances whose k-NN all reside in the same partition can be identified. They are named "grid-local tuples". For other data instances, which are named "cross-grid tuples", they have devised a way to construct a minimum rectangle extension in every possible adjacent grid, which covers all the potential neighbors. Yan et al. [36] proposed a similar distributed version of Local Outlier Factor (LOF), named DLOF. DLOF greatly resembles the approach presented by Bai [185], but some extra optimizations are also introduced. DLOF uses grid-based data partitioning, with which each partition works in a fully distributed fashion. To ensure its compliance with the popular shared-nothing distributed infrastructures, the notion of supporting area is created so that every data instance p is distributed to other partitions where p is a k-nearest neighbor to some instances in those partitions. Moreover, some optimizations are introduced based on the observations that the necessary extending area for a data instance p cannot exceed the sphere with a radius of the local k-distance of p as well as that a data instance whose LOF score is computed and is not needed in any of the supporting areas can be eliminated.

The Distributed Top-N LOF (DTOLF) [183] provided a distributed solution for the Top-N LOF approach proposed in [186]. DTOLF also utilizes grid-based data partitioning as DLOF [36]. It features a pruning strategy that eliminates data instances that are guaranteed not to be Top-N outliers and that are not needed by the computation of other machines. The pruning strategy takes into account the distances between the data points inside a partition and the boundaries of the partitions. Because the pruning strategy merely relies on the local data characteristics of a specific partition itself, it enables the reduction of communication cost among the machines in a cluster. Additionally, this elimination strategy reduces the data duplication rated compared to DLOF. To mitigate the problem of higher-dimensional data, they have introduced a correlation-aware partitioning, which is based on the observation that real-world datasets usually have strongly correlated dimensions, and thus data partitioning can be carried out merely on independent dimensions.

A major limitation of the methods relying on grid-based data partitioning is that they do not scale well with the dimensions of the data. The number of grid cells grows exponentially with the increase of data dimensions. In the case of k-NN-based algorithms, each data instance may be needed by a great number of other grids or partitions in order to determine the k-NN neighborhoods. This usually incurs high communication cost across the cluster.

Moreover, Tsou et al. [24] presented a distributed unsupervised anomaly detection framework to address the challenges in anomalous behavior detection of wireless sensor network devices. A wireless sensor network [187–193] comprises a collection of geographically dispersed devices that can measure and record environmental conditions (e.g., temperature, air poluttion level, etc.). The proposed anomaly detection framework approach relies on the one-class random forest [184], and the devices collaborate by sharing their models instead of data. To discriminate decision tree models according to their effectiveness, they developed an unsupervised ensembling algorithm to optimize the weights of the decision trees. The weights are learned by minimizing the uncertainty of the predictions for data points in an unsupervised fashion.

Chapter 4

Distributed Local Outlier Factor in MapReduce

Local Outlier Factor [30] has been a very popular outlier detection method over the past few years. LOF is based on the relatvie densities of data instances to their nearest neighbors instead of absolute density adopted by many of the previous methods [88] [113]. Thus LOF is more suitable to be applied to datasets with regions of different densities and outperforms other algorithms in a wide range of applications [89].

However, the traditional centralized LOF is limited by its computational cost mainly due to the k-NN search, especially when the datasets increase rapidly in size. Additionally, the datasets of nowadays are becoming more and more distributed. Therefore, scalable distributed LOF algorithms are highly desirable.

We start by introducing a baseline MapReduce solution for LOF in Spark, which is named MR-LOF. We show that it has comparatively high communication and computation overhead. To decrease the execution time, we propose an approximate algorithm that takes advantage of a two-layered LSH for data partitioning and computes approximate LOFs locally in a self-sufficient way. We name this method MR-LOF-LSH. To provide a more accurate approximation, we have developed a strategy called cross-partition updating that recalculates the LOFs of the top candidates based on the candidates' actual global k-nearest neighborhoods.

4.1 MR-LOF: A Baseline Distributed LOF Approach in MapReduce

This section gives a detailed description of MR-LOF, the baseline distributed LOF approach in the MapReduce paradigm. On a high level, we first broadcast the dataset in partitions so that each data instance can get in touch with every other data instance to form the k-nearest neighborhood. The k-NN information is then used in successive steps to compute the k-distance, local reachability density (LRD) and finally the LOF score for each data instance step by step. In the end, we analyze the induced shuffle cost and time complexity of MR-LOF in the distributed setting.

We make use of several concepts and functions from MapReduce and Apache Spark to better explain various stages in this approach. They are:

- **RDD**: as mentioned in Section 2.1.2.2, an RDD is a distributed immutable collection of objects and all the objects in an RDD should be of the same type. For example, each object in *neighborhoodRDD*, as shown in Figure 4.1, is a key-value pair. The key is the data instance ID, and the value is an array of tuples, each of which consists of a neighbor's ID and the corresponding distance from that neighbor to the data instance. RDD is the core abstraction in Spark and data processing in Spark heavily relies on RDDs.

- **Map**: the *map* function applies another function on each element of a RDD. In the pseudocode of this thesis, the function being applied to each element is described in the function body of the *map* function, whose input is one of those elements in the collection. Often the elements are in key-value pairs.

- **ReduceByKey** and **GroupByKey**: *reduceByKey* merges together the key-value pairs with the same key and applies another function on each key and the corresponding list of merged values, denoted as *valList* in the psuedocode. To be more accurate, the function being applied is performed on an accumulator, one value each time, with the output as the new accumulator for the next value. By contrast, *groupByKey* only performs the merging.

- **Join** and **LeftJoin**: *join* combines two RDDs and merges two key-value pairs with the same key, each from one of the RDDs. The resulting key-value-value tuples are

the elements in the output RDD. The difference between *leftJoin* and *join* is similar to that in SQL.

- **Emit**: *emit* is not a function used in the MapReduce programming model. It is used here to declare the ending of a *map* or *reduceByKey* function and specify the format of the objects in the output RDD on a per-key basis.

Besides, the symbols used in the psyeudocode are summarized in Table 4.1.

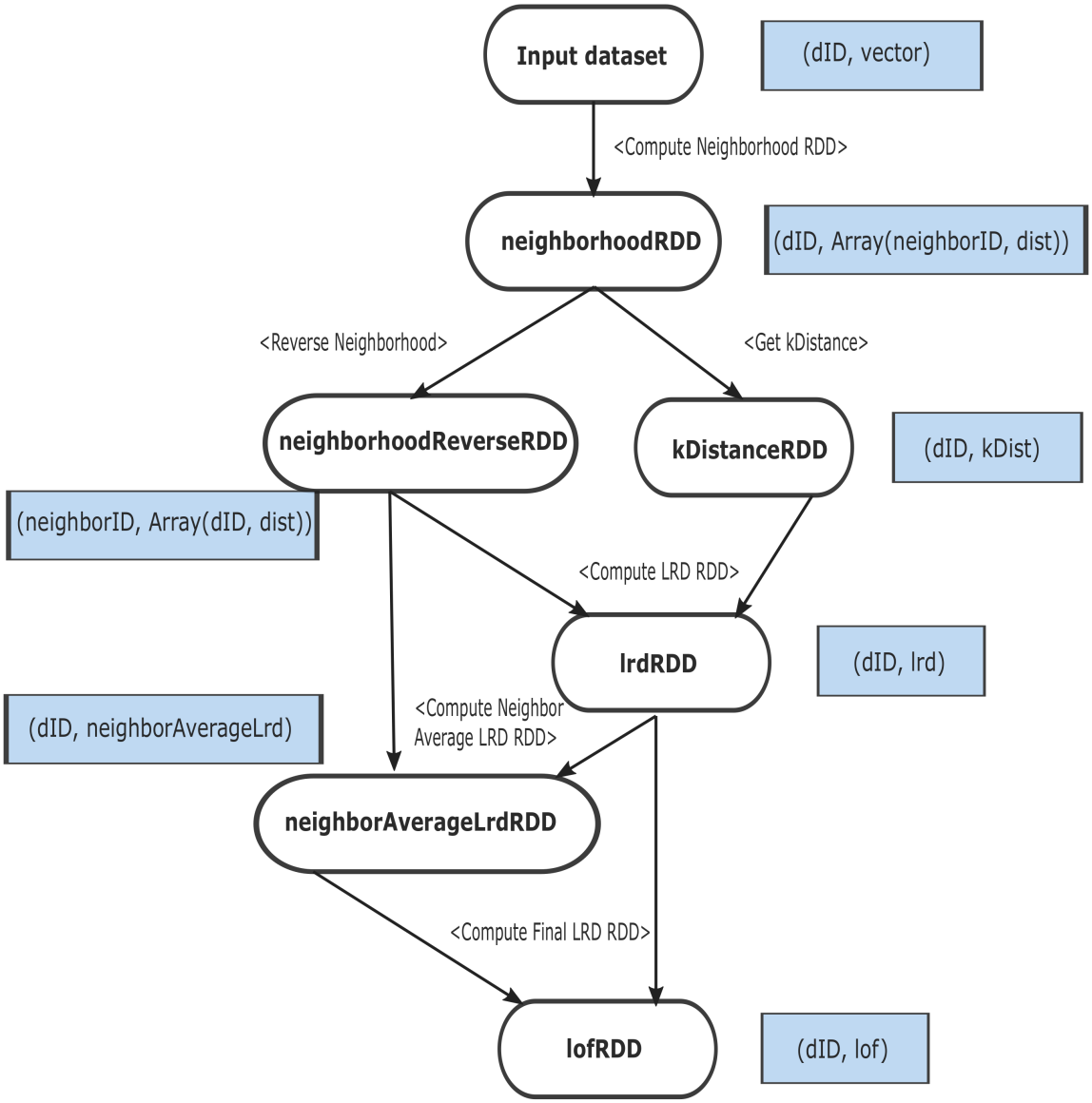


Figure 4.1: Overview of RDD transformations

MR-LOF can be roughly outlined by three stages, computing k-nearest neighborhood, computing local reachability density (LRD) and computing final LOF. Figure 4.1 presents the overview of MR-LOF by illustrating the transformations among the RDDs. What an individual element of a RDD contains is denoted by the blue square box beside the RDD box, and the arrow represents the transformation.

Symbol	Description
Partitions	The input dataset in the form of multiple partitions distributed on different physical machines
d	A data instance
dID	The index of a data instance d
dist	The distance (squared Euclidean distance is used in this work) between two data instances whose indices are involved

Table 4.1: Symbols in the pseudocode and their descriptions

4.1.1 Compute K-Nearest Neighborhood

In this stage, the original dataset is processed and the k-NN of each data instance are found, which is as shown in Algorithm 1. In order to make every data instance appear in every other instance's search scope for the nearest neighborhood, each partition is broadcasted across the cluster. For readers who are interested, this is implemented using RDD's *mapPartitions* method and Spark's *broadcast variable*. *computeNeighborhood*(line 6) finds the k-NN of d from partition P_j by maintaining a priority queue of size k. The record emitted by the *map* function is a key-value pair containing the ID of a data instance and its neighbors found on a particular partition.

In typical cases where there are more than one partition, we can expect multiple records exist for the same data point ID in the output of the *map* function. *reduceByKey* collects the records with the same key and merges these neighborhoods into one final neighborhood of size k. This procedure is performed by applying *mergeNeiborhood* (line 4) on two neighborhoods each time, resulting in a new neighborhood of size k (denoted as *accumulator*) to be merged with the next neighborhood. Since the neighbor array generated from *map* function is sorted (by dequeuing the priority queue), *mergeNeiborhood* is simply merging two sorted arrays, keeping only the top k neighbors with the smallest distances.

Algorithm 1 Neighborhood RDD

Input *Partitions*
Output *neibhorhoodRDD*

function MAP($P_i \in Partitions$)
    broadcast P_i across the cluster
    for all $P_j \in Partitions$ where $i \neq j$ **do**
        obtain broadcasted P_i on P_j
        for all $d \in P_i$ **do**
            $(dID, \texttt{Array}(neighborID, dist)) \leftarrow$ COMPUTENEIGHBORHOOD(d, P_j)
            EMIT($dID, \texttt{Array}(neighborID, dist)$)
        end for
    end for
end function

function REDUCEBYKEY(dID, $valList = [\texttt{Array}(neighborID, dist), ...]$)
    $accumulator(neighborID, dist) \leftarrow empty$
    for all $array \in valList$ **do**
        $accumulator \leftarrow$ MERGENEIGHBORHOOD($accumulator, array$)
        EMIT(dID, $accumulator$)
    end for
end function

4.1.2 Compute Local Reachability Density

The computation of local reachability density RDD is illustrated in Algorithm 3. The definition of local reachability density can be found in Eq. (3.1). It requires two auxiliary RDDs, namely the k-distance RDD and the neighborhood-reverse RDD. As the names indicate, those RDDs contain related information for each data point.

We obtain the k-distance RDD by getting the last element in the neighbor array for each data instance. Neighborhood-reverse RDD is generated with Algorithm 2, which reverses the key-value relationship between a data point and its neighbors, resulting in an RDD containing information about in which data points' neighbhorhood the key data point resides.

The *map* function in Algorithm 3 works on an RDD joining k-distance RDD and neighborhood-reverse RDD, which allows us to find the reachability distance between a data point and its neighbors. Finally, the local reachability density is the reciprocal of the average reachability distance between a data instance and its k-NN.

Algorithm 2 Reverse Neighborhood RDD

Input $neighborhoodRDD$
Output $neighborhoodReverseRDD$

function MAP($(k : dID, v : \texttt{Array}(neighborID, dist)) \in neighborhoodRDD$)
    for all $(neighborID, dist) \in v$ **do**
        EMIT($neighborID, (dID, dist)$)
    end for
end function
function GROUPBYKEY($neighborID$, $valList = [(dID, dist), ...]$)
    EMIT($neighborID, valList$)
end function

Algorithm 3 Local Reachability Density RDD

Input $neighborhoodRDD$
Output $lrdRDD$ ▷
/* Preparatory RDD transformations */
$kDistanceRDD \leftarrow$ GETKDISTANCE($neighborhoodRDD$)
$neighborhoodReverseRDD \leftarrow$ REVERSENEIGHBORHOOD($neighborhoodRDD$)
$joinedRDD \leftarrow$ JOINRDD($neighborhoodReverseRDD, kDistanceRDD$)
function MAP($(k : neighborID, v) \in joinedRDD$) ▷
/* $v : (\texttt{Array}(dID, dist), NeighborKDistance)$ */
    for all $(dID, dist) \in v.array$ **do**
        $reachDistance \leftarrow$MAX($v.NeighborKDistance, dist$)
        EMIT($dID, (reachDistance, 1)$)
    end for
end function
function REDUCEBYKEY(dID, $valList = [(reachDistance, count), ...]$)
    $(distanceSum, countSum) \leftarrow (0, 0)$
    for all $(reachDistance, count) \in valList$ **do**
        $distanceSum \leftarrow distanceSum + reachDistance$
        $countSum \leftarrow countSum + count$
    end for
    $localReachabilityDensity \leftarrow countSum/distanceSum$
    EMIT($dID, localReachabilityDensity$)
end function

4.1.3 Compute Final LOF RDD

The final stage consists of two steps: attaining the average local reachability density of the neighborhood and computing the LOF score, which are detailed in Algorithm 4 and Algorithm 5.

In Algorithm 4, the reverse neighbor relationship is used to produce the average local reachability density in the neighborhood of each data instance, by joining neighorhood-reverse RDD and LRD RDD. In Algorithm 5, by joining neighbor-average-LRD RDD and LRD-RDD, the final LOF score for individual data instances can be obtained.

Algorithm 4 Neighbor Average Local Reachability Density(LRD) RDD

Input $neighborhoodReverseRDD, lrdRDD$
Output $neighborAverageLrdRDD$ ▷
/* Preparatory RDD transformations */
$joinedRDD \leftarrow$ LeftJoinRDD$(neighborhoodReverseRDD, lrdRDD)$
function Map$((k : neighborID, v) \in joinedRDD)$ ▷
/* $v : (\texttt{Array}(dID, dist), neighborLrd)$ */
  for all $(dID, dist) \in v.array$ **do**
    Emit$(dID, (neighborLrd, 1))$
  end for
end function
function ReduceByKey$(dID,\ valList = [(neighborLrd, count), ...])$
  $(neighborLrdSum, countSum) \leftarrow (0, 0)$
  for all $(neighborLrd, count) \in valList$ **do**
    $neighborLrdSum \leftarrow neighborLrdSum + neighborLrd$
    $countSum \leftarrow countSum + count$
  end for
  $neighborAverageLrd \leftarrow neighborLrdSum/countSum$
  Emit$(dID, neighborAverageLrd)$
end function

4.1.4 Complexity Analysis

In this section, we analyze the shuffle cost and the time complexity for computation. The meaning of various symbols used for complexity analysis can be found at Table 4.2. To simplify the analysis, we only take into account the major shuffle cost and computational cost. Additionally, some assumptions are made to idealize the experimental situations:

- Each node in the cluster contains exactly one partition.

Algorithm 5 Final LOF RDD

Input $neighborAverageLrdRDD, lrdRDD$
Output $lofRDD$ ▷
/* Preparatory RDD transformations */
1: $joinedRDD \leftarrow$ JoinRDD($neighborAverageLrdRDD, lrdRDD$)
1: **function** Map($(k : dID, v : (neighborAverageLrd, lrd)) \in joinedRDD$)
2: **for all** $(neighborAverageLrd, lrd) \in v$ **do**
3: $lof \leftarrow neighborAverageLrd/lrd$
4: Emit(dID, lof)
5: **end for**
6: **end function**

Symbol	Description
N	The total number of data instances in the dataset
N_i	The number of data instances in the i^{th} partition
p	The number of partitions
m	The number of dimensions (columns) of a data instance
k	The number of neighbors that form the k-NN neighborhood
c	The number of outlier candidates (for MR-LOF-LSH only)
j	The number of LSH functions (for MR-LOF-LSH only)

Table 4.2: Symbols in the complexity analysis and their descriptions

- The partitions are evenly divided.
- The parallel stages on different nodes begin and end synchronously.

4.1.4.1 Shuffle Cost

Many distributed data processing jobs inevitably involve communication and data transmission among different nodes. The shuffle cost is the size of data to be transferred over the cluster, although the entry size of the data (e.g., a double number has 8 bytes) is omitted in our analysis for simplicity. The shuffle data must be written to the disk, transferred over the network then loaded into memory again. Therefore, shuffling can be very time-consuming when the data size is considerably big.

In computing the neighborhood RDD, the shuffle cost is induced by broadcasting individual partitions to every other node as well as performing *reduceByKey* in order to merge

the intermediate k-NN information. Note that each data instance's ID corresponds to p records as emitted by the *map* function in Algorithm 1. Thus, this stage has a shuffle cost of

$$p \sum_{i=1}^{p} N_i m + p \sum_{i=1}^{p} N_i \; k = Np\,(m + k)\,.$$

As for computing neighborhood-reverse RDD, *groupByKey* incurs a shuffle cost of

$$p \sum_{i=1}^{p} N_i \cdot k = Npk.$$

The shuffle cost in computing local reachability density RDD is incurred by the $joinRDD$ and $reduceByKey$, leading to a cost of

$$N \cdot k + N \cdot k + N = N\,(2k + 1)\,.$$

Note that each data instance is linked with k records in the result of *map* function of Algorithm 3.

The processing described in Algorithm 4 and Algorithm 5 has a shuffle cost of

$$N \cdot k + N + N \cdot k + N + N = N\,(2k + 3)\,.$$

caused by $join$ and $reduceByKey$.

In summary, the shuffle cost is:

$$N\,(pm + 2pk + 4k + 4)\,.$$

4.1.4.2 Time Complexity for Computation

The time complexity for computation described here is used to analyze the time to be spent on parallel computation, excluding the time cost for data transferring (shuffling).

In the phase of computing neighborhood RDD, each invocation of *computeNeighborhood* contributes $\frac{N}{p}k$ because of the sequential scan for neighbors and the computation of Euclidean distances. Considering the partition is evenly divided, and the broadcasting of single partitions happens sequentially, the time complexity for the *map* stage is

$$p \cdot \left(\frac{N}{p}\right)^2 \cdot mk = O\left(\frac{N^2}{p}\right),$$

where we treat m and k as constants. We keep p considering the assumption that each node has a maximum computing capacity for a certain number of data points due to the limitation of memory, storage, etc. Therefore, p is expected to increase as the N goes up, in order to prevent system failures.

As for the *reduceByKey* stage, each data instance has p records and the merging occurs in parallel on p nodes. Also, each merging scans k elements in two sorted arrays. Therefore, the *reduceByKey* stage has a time complexity of

$$\frac{p \cdot N}{p} \cdot k = O(N).$$

In later phases of the data processing, the time complexity for computation is bounded by $O(N)$ since it only involves simple arithmetic computation.

In summary, the time complexity for parallel computation is given by

$$O\left(\frac{N^2}{p}\right).$$

4.2 MR-LOF-LSH: A Distributed LOF Approach in MapReduce with Locality-Sensitive Hashing

As we can observe from the complexity analysis in last section, the baseline MR-LOF incurs high time complexity for computation and extremely expensive shuffle cost. The high overhead is primarily attributed to the fact that every data point is shuffled to every other machine to compute the LOF for every data instance in a shared-nothing architecture like MapReduce. Besides, the computation of LOF is complex. As pointed out in [36], computing the LOF of a data instance involves its k-NN, its k-NN's k-NN and its k-NN's k-NN's k-NN, all together $k + k^2 + k^3$ points. This leads to the complicated interactions of the intermediate RDDs.

In order to reduce the computation cost and shuffle cost, we propose an approximate method by exploiting LSH for data partitioning. We name it MR-LOF-LSH. MR-LOF-LSH consists of three stages: LSH partitioning, local computation and cross-partition updating,

the overview of which is depicted in Figure 4.2. The black circles and white circles represent two different types of normal data instances while the circles with black stripes represent outliers.

We use LSH to partition the dataset so that closer data points have higher chances of falling into the same partition. As the example in Figure 4.2 shows, the white circles fall in different partitions from the black circles. This property is highly desirable for the distributed LOF method because we want most of the k-NN to be found in local LOF computations. However, it is still possible that some data instances have some of their k-NN separated from the partition they belong to, leading to a poor-quality approximation of LOF scores. To tackle this issue, we have developed a strategy called cross-partition updating, which updates the LOF of the top-N outlier candidates based on the actual global k-NN of these candidates.

4.2.1 LSH Data Partitioning

In distributed file systems such as HDFS, data are divided into a number of partitions, each partition stored on a different physical computer. How the dataset is partitioned has a significant impact on the quality of the outlier detection result for our method. We make use of a two-layered LSH scheme to map individual data instances to a 1-dimensional space. The LSH value determines which partition a data instance should be placed in.

For the first layer, we take

$$H(v) = [h_1(v), h_2(v), ...h_k(v)]\,. \tag{4.1}$$

Each of the k values $h_i(v)$ ... is drawn from the hash family mentioned earlier as Eq. (3.14), where the entries of a are randomly sampled from the standard Gaussian distribution, which is a 2-stable distribution.

For the second layer, we simply take

$$g(x) = a' \cdot x, \tag{4.2}$$

with a' being a k-dimensional vector, entries of which are randomly selected from the standard Gaussian distribution. Note that the input vector x for the second-layer function is the output of the first-layer function, consisting of k elements. Thus, for a data instance

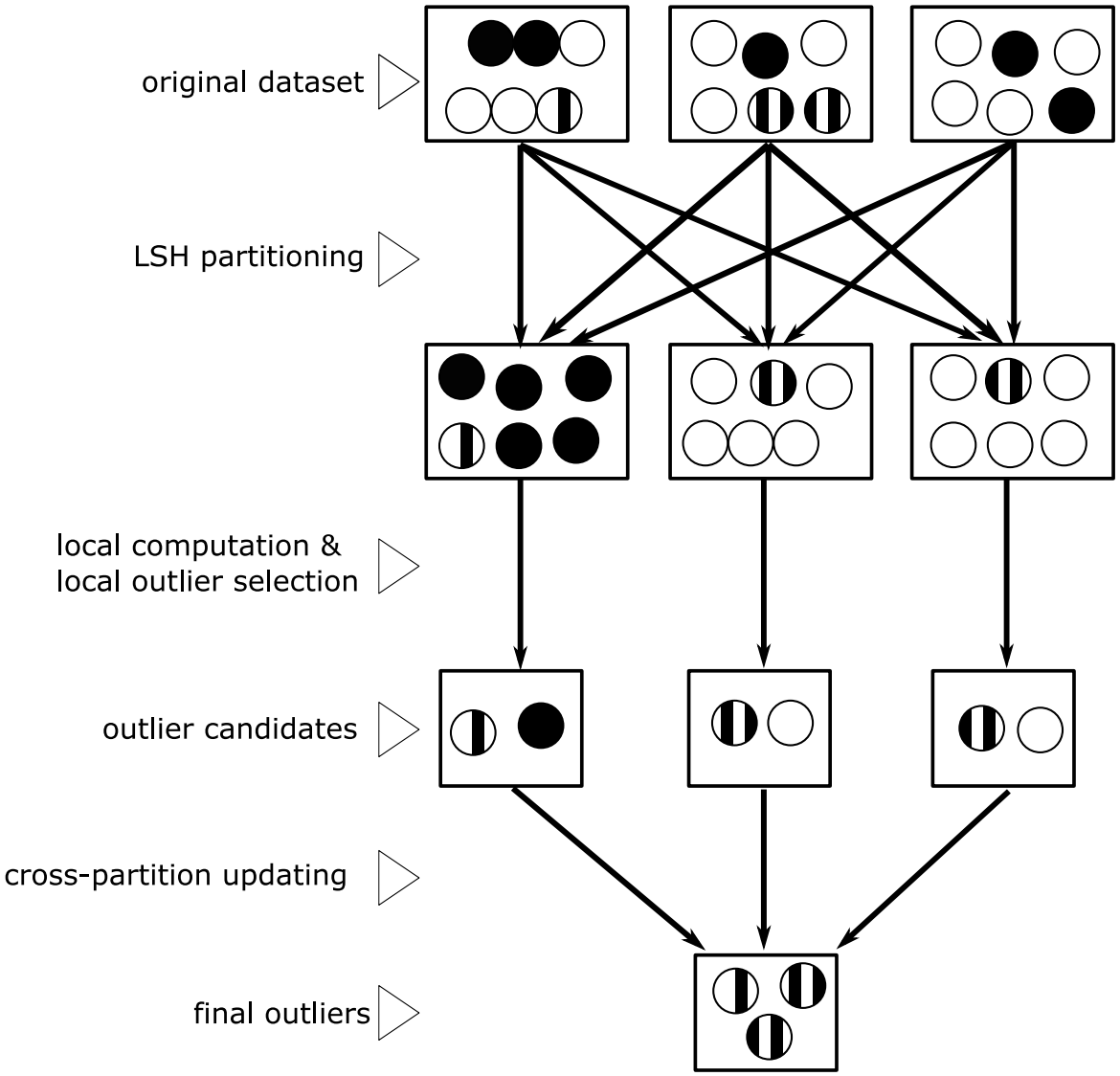


Figure 4.2: The overview of MR-LOF-LSH

v, the final output hash value is $g\left(H\left(v\right)\right)$.

Then we sort the hash values of the data points and divide them into as many segments as the given number of partitions. Each segment represents an actual partition. The data instances are to be transferred to specific partitions based on which segment the hash value lies in. The segmentation of the hash value space is ideally expected to result in each partition containing roughly the same number of data points, in consideration of load balancing. Figure 4.3 illustrates an overview of the mapping from d-dimensional data to

k-dimensional space of first-layer hash values, and finally to 1-dimensional space of second-layer hash values.

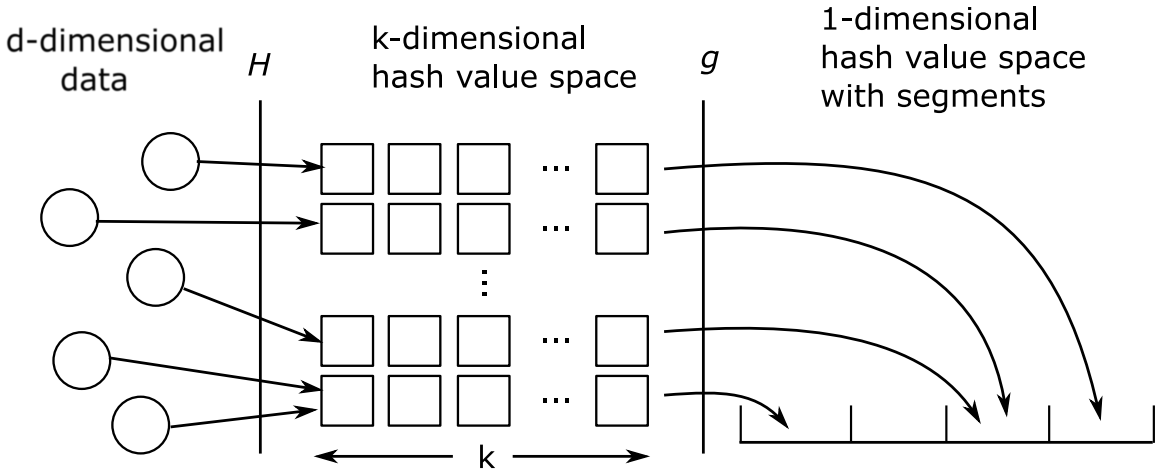


Figure 4.3: Illustration of the two-layered mapping from d-dimensional data to 1-dimensional hash value space with segments

To avoid the excessive shuffling induced by sorting the entire RDD, an approach similar to Hadoop TeraSort [194] can be exploited here. The actual implementation used in our experiments is RDD's *sortBy* function. In short, we grasp the approximate data distribution in individual partitions via reservoir sampling. The sample size for each partition is 60, and the overall sampling size is bounded by 6 minions. With the approximate distribution information, the hash value ranges for individual partitions can be determined so that each range would predictably contain a roughly equal number of elements. Therefore, the partitioning can be performed merely by mapping the data instances to the corresponding ranges their hash values fall in.

Next, we analyze the two-layered LSH scheme. As indicated in [53], two data instances with a larger Euclidean distance have a higher probability of resulting in a larger difference between hash values computed from Eq. (3.14). In other words, the difference between the hash values preserves the Euclidean distance in the original data space. Similarly, the second layer LSH function as described by Eq. (4.2) preserves the distance between the vectors of hash values computed from the first layer hash function, which will be proven at the end of this section. Therefore, the output of the two-layered LSH scheme contains information about the Euclidean distance in the original data space. If we segment those hash values taking into account their orders and differences, we will have a higher chance to gather data points close to each other in the same partition.

Here are the reasons why we have used Eq. (4.2) as the second layer LSH function instead of applying Eq. (3.14) again. First of all, the floor operation in Eq. (3.14) results in integers as the bin identifiers. This makes the segmentation of the hash values less convenient for the sake of load balancing. Consider several scenarios: there can be more partitions than bins, which means we need to devise a strategy to decide which bins to split up and how to do that. Another scenario is that the sizes of the bins can be very different and it is difficult to make sure the resulting partitions are roughly equal. Therefore, using float hash values enables a fine granular partitioning with load balancing. If we remove the floor operation in Eq. (3.14), b and w would have no impact on the partitioning results. Thus it is simplified into Eq. (4.2), with the additional advantage of having fewer parameters.

Theorem 4.2.1. For two data points $v, q \in S$, $c = \|v, q\|_2$ and a distance δ, a negative correlation exists between $Pr\{|g(v) - g(q)| \leq \delta\}$ and c; a positive correlation exists between $Pr\{|g(v) - g(q)| \leq \delta\}$ and δ.

Proof. $|g(v) - g(q)| = |av - aq| = |a(v - q)|$. According to the property of p-stable distribution, $a(v - q)$ has the same distribution as cX, where X is a standard Gaussian distribution in this case. Let $f(x)$ be the probability density function of the absolute value of the standard Gaussian distribution, as described in Eq. (2.2). Therefore, the probability density function of $|a \cdot (v - q)|$ is $\frac{1}{c} f\left(\frac{x}{c}\right)$. Thus,

$$Pr\{|g(v) - g(q)| \leq \delta\} = \int_0^{\delta} \frac{1}{c} f\left(\frac{x}{c}\right) \mathrm{d}x = \int_0^{\frac{\delta}{c}} f(x) \, \mathrm{d}x.$$

Considering the properties of $f(x)$, we can easily draw a conclusion that a negative correlation exists between $Pr\{|g(v) - g(q)| \leq \delta\}$ and c; a positive correlation exists between $Pr\{|g(v) - g(q)| \leq \delta\}$ and δ.

4.2.2 Parallel Computation of LOF

Our goal is to find top-N outliers, which are the top-N data instances with the highest LOF scores. After the LSH data partitioning, we assume that each data instance has a high chance to find most of its k-NN locally. Thus, if we compute the LOF scores based on the data points in the same partition, the result would very likely be close to the scores computed based on the entire dataset. Moreover, the actual top-N outliers tend to have relatively high LOF scores on every partition.

Therefore, a good idea would be to globally sort the locally computed LOF scores and pick the top-N points. But to avoid the shuffling cost induced by sorting the entire dataset, our strategy is to collect top outlier candidates on each partition then merge the collections to the driver (master node) to find the final top outliers.

However, some normal data instances could turn out high in LOF locally due to some of its k-NN ending up in other partitions. If there is a considerable number of such normal data instances, real outliers can be pushed out of the top-N list. To handle this situation, we could expand the candidate size in order to cover more actual outliers and also verify the candidates across the entire dataset in order to filter out some inliers having high LOF scores locally. This strategy is to be detailed in next subsection.

Algorithm 6 Parallel Computation of LOF

Input *Partitions*: dataset in partitions, *candiSize*: candidate size
Output *topCandidates*

1: **for all** $P_i \in Partitions$ **do**
2: $localLOF \leftarrow$ComputeLOF(P_i)
3: $localTopN \leftarrow$GetTopN($localLOF$, $candiSize$)
4: $topNList$.Add($localTopN$)
5: $globalThreshList$.Add($localTopN.min$)
6: **end for**
7: $topNList$.FilterOut($globalThreshList.max$)
8: $topCandidates \leftarrow$GetTopN($topNList$, $candiSize$)
9: Return(topCandidates)

The details of parallel computing of LOF are presented in Algorithm 6. The *ComputeLOF* function is the implementation of the centralized LOF algorithm. We have a few methods to choose for k-NN search. Utilizing an indexing technique such as KD-tree [195] leads to a time complexity of $O(N \log N)$. However, they are usually not applicable to high dimensional datasets. Without loss of generality, we only make the sequential scans aided with a priority queue, having a complexity of $O(N^2)$. However, the k-NN search technique is a completely pluggable component in our framework, which can be altered to suit specific practical scenarios.

Note that $LOFPartitions$ and $topNList$ are actually distributed collections (RDD). They contain the partition information of the input data. $localLOF$ is computed according to Eq. (3.1) (3.2) (3.3). $globalThreshList$ is a global accumulator, the corresponding implementation in Spark being *collectionAccumulator*. It collects the smallest LOF score among the local candidates (the n^{th} candidate) for each partition and the pick the maximum

among them (line 10) as the threshold to prune those candidates impossible to be in the final top-N list thus reducing the number of data points to be shuffled across the network. *GetTopN* is implemented based on a priority queue (heap).

This parallel computation scheme can significantly expedite the outlier detection process because in addition to taking advantage of parallelism, there is much less communication over the network compared to MR-LOF.

Algorithm 7 Compute Candidates' Global Neighborhoods

Input $LOFPartitions$, $candidates$
Output $candidateNeibhorhoodRDD$ ▷
/* Preparatory processing */
broadcast $candidates$ across the cluster
1: **function** MAP($P_i \in LOFPartitions$)
2: obtain broadcasted $candidates$ on P_i
3: **for all** $d \in candidates$ **do**
4: $(dID, \texttt{Array}(neighborID, neighborInfo)) \leftarrow$ COMPUTENEIGHBORHOOD(d, P_i)
5: EMIT($dID, \texttt{Array}(neighborID, neighborInfo)$) ▷
/* $neighborInfo$ contains information: k-distance, distance and density */
6: **end for**
7: **end function**
1: **function** REDUCEBYKEY(dID, $valList = [\texttt{Array}(neighborID, neighborInfo), ...]$)
2: $accumulator(neighborID, neighborInfo) \leftarrow empty$
3: **for all** $neighborhood \in valList$ **do**
4: $accumulator \leftarrow$ MERGENEIGHBORHOOD($accumulator, neighborhood$)
5: EMIT(dID, $accumulator$)
6: **end for**
7: **end function**

4.2.3 Cross-partition Updating

Since the LOFs output by Algorithm 6 are approximate scores, the resultant top-N list can be different from that based on the actual LOF scores generated by MR-LOF. Consider a specific scenario: some data instances are assigned high LOFs because most of their original neighbors are distributed to other partitions. These data points become false positives and can push real outliers out of the top-N list.

In order to filter out the false positive cases mentioned above and obtain a more accurate top-N list of outliers based on LOF, we suggest a process called *cross-partition updating*,

Algorithm 8 Update Candidate LOF Score

Input $candidateNeibhorhoodRDD$
Output $candidateLRDRDD$

1: **function** MAP($(k : dID, v) \in candidateNeibhorhoodRDD$) ▷
/* v : `Array`$(neighborID, neighborInfo)$ */
2: $nborDensitySum \leftarrow 0$
3: $nborReachDistSum \leftarrow 0$
4: **for all** $(neighborID, neighborInfo) \in v$ **do**
5: $nborDensitySum \mathrel{+}= neighborInfo.density$
6: $nborReachDistSum \mathrel{+}= \text{MAX}(neighborInfo.kDistance, neighborInfo.dist)$
7: **end for**
8: $nborAverDensity \leftarrow nborDensitySum/v.size$
9: $density \leftarrow v.size/nborReachDistSum$
10: $lof \leftarrow nborAverDensity/density$
11: EMIT(dID, lof)
12: **end function**

which attempts to find more accurate LOF approximations for the outlier candidates. Algorithm 7 and Algorithm 8 illustrate this process.

At a high level, the idea is to broadcast the entire set of candidates across the cluster in order to find their precise neighborhoods in a global sense. Then the new LOFs are computed by using the information associated with the exact neighbors. The information includes the distance between the data instance and its neighbor, the k-distance of the neighbor and the reachability density (LRD) of the neighbor. The latter two are approximate results computed as the intermediate products in parallel computing of LOF (Algorithm 6), based on the data instances of merely one partition. The information can be stored along with LOF scores in an RDD, which is referred to as *LOFPartitions* in Algorithm 6 and Algorithm 7.

4.2.4 Complexity Analysis

In this section, we analyze the shuffle cost and the time complexity for computation in the same manner as in Section 4.1.4, with the same assumptions and symbols shown in Table 4.2.

4.2.4.1 Shuffle Cost

The major shuffle cost comes from LSH-partitioning, with the entire dataset and the LSH value shuffled, leading to a shuffle cost of $N \cdot m$.

For the parallel computation of LOF, only the candidates are shuffled and sorted according to the LOF. Each partition contributes c candidates. Therefore, the shuffle cost is $c \cdot p$.

Finally, during cross-partition updating, c candidates are broadcast in the cluster. The neighborhood merging also induces shuffle cost. Thus, the shuffle cost for cross-partition updating is $c \cdot p + c \cdot k$.

Altogether, the shuffle cost is:

$$N \cdot m + 2 \cdot c \cdot p + c \cdot k.$$

However, note that the candidate size c is usually sufficiently smaller than N, making the terms associated with c negligible.

4.2.4.2 Time Complexity for Computation

LSH-partitioning computes a hash value for each data instance, using j hash functions, thus the time complexity is

$$\frac{N}{p} \cdot j \cdot m = O\left(\frac{N}{p}\right).$$

As for parallel computation of LOF, the time complexity is bounded by

$$\left(\frac{N}{p}\right)^2 \cdot m \cdot k = O\left(\left(\frac{N}{p}\right)^2\right),$$

because of the sequential k-NN search.

During cross-partition updating, the time complexity incurred by searching for and merging the candidates' neighborhoods is

$$\frac{N}{p} \cdot ck + ck = O\left(\frac{N}{p}\right).$$

The total time complexity is bounded by

$$O\left(\left(\frac{N}{p}\right)^2\right).$$

4.2.4.3 In Comparison with MR-LOF

Let us assume that N is sufficiently large to a degree where m, k, j, c are negligible in terms of complexity analysis, and we only look at the major terms. Hence the shuffle cost comparison between MR-LOF and MR-LOF-LSH is:

$$Np \cdot (m + 2k) \quad \texttt{versus} \quad N \cdot m.$$

This indicates that the shuffle cost of MR-LOF increases p times as fast as that of MR-LOF-LSH. Since p is the number of partitions (slave nodes), MR-LOF would predictably scale worse than MR-LOF-LSH.

If we look at the comparison in time complexity for computation, then

$$O\left(\left(\frac{N}{p}\right)^2\right) \quad \texttt{versus} \quad O\left(\frac{N^2}{p}\right).$$

We can see that their complexity both decreases with p, MR-LOF linearly and MR-LOF-LSH quadratically. This also indicates that MR-LOF-LSH would scale better than MR-LOF, to a growing size of the cluster.

Chapter 5

Experimental Evaluation

In this chapter, we conduct experiments to evaluate the proposed distributed LOF methods with different parameter settings on real-life and synthetic datasets. We first describe the experimental infrastructure and environment. Then we introduce the datasets that our methods are applied to, including where they come from and how they are preprocessed. The strategies for data normalization and duplicate handling are also presented. To evaluate the efficiency and accuracy of the proposed methods, we use the elapsed execution time and recall as the metrics. As for the actual experiments, we compare the execution time of the baseline MR-LOF method and MR-LOF-LSH-CU. We also study the scalability of MR-LOF-LSH-CU against different cluster sizes. To evaluate the accuracy, we vary the number of partitions and the candidate size. Finally, we test the impact of the LSH function parameter w and the number of LSH functions on the recall of MR-LOF-LSH. Every experiment is performed on each of the three datasets.

For clarity, let us first look at the names for the methods evaluated in the experiments.

- **MR-LOF**: the method introduced in Section 4.1.
- **MR-LOF-LSH**: the method detailed in Section 4.2 without the cross-partition updating.
- **MR-LOF-LSH-CU**: MR-LOF-LSH plus performing cross-partition updating in the end.
- **MR-LOF-RD**: similar to MR-LOF-LSH in that LOFs are computed locally within each partition, the difference being that MR-LOF-RD partitions the dataset ran-

domly instead of using LSH functions for projection. MR-LOF-RD is used to contrast MR-LOF-LSH in order to see how much the LSH-partitioning contributes to picking out the right top outliers.

5.1 Experimental Infrastructure

We conduct our experiments on Google Cloud Platform's Dataproc clusters. A cluster is established with one master node with 8 vCPUs, 30 GB memory and 200 GB disk storage and multiple slave nodes each with 4 vCPUs, 15 GB memory and 100 GB disk storage. The number of slave nodes is set the same as the number of data partitions. The Spark version is 2.2.1 and Hadoop version is 2.8.4. All the Spark jobs run on YARN, the resource management and job scheduling module of Hadoop. The Spark driver memory is configured as 16 GB and Spark executor memory 4 GB. The number of Spark executor cores is set to 2.

Table 5.1 presents the default values for the algorithm-related parameters in most experiments. Besides, we have used a different w as in Eq. (3.14) for each dataset: Synthetic 0.2, CoverType 0.1 and KDDCup99 0.8, in order to deliver good performance. Those default values do not apply when otherwise specified or when the parameter is used as a varying parameter.

Name	Description	Default Value
nPartitions	Number of data partitions	10
nNodes	Number of slave nodes in the cluster	10
nLSHFunctions	Number of LSH functions	15
nNeighbors	Same as k in k-nearest neighbors	30
candidateTimes	Ratio of candidate size to outlier size	2

Table 5.1: Default values for the parameters

5.2 Datasets

We evaluate our proposed methods on a synthetic dataset and two real-world datasets: CoverType and KDDCup99, obtained from the UCI machine learning repository [196].

The synthetic dataset is generated from 5 10-dimensional multivariate Gaussian distributions, each contributing 200,000 instances, resulting in 1,000,000 records altogether. The multivariate Gaussian distributions have different mean vectors. Three of them have a unit matrix as their covariance matrix while the other two use a diagonal matrix with 4 in each diagonal entry. The number of outliers is set as 1000.

CoverType contains cartographic attributes originally used to predict the type of forest covers for 30X30 meter cells in the Rocky Mountain region. The dataset comprises 581,012 instances with 54 attributes. We remove the cover type labels to suit the unsupervised outlier detection problem. The number of outliers is set as 1000.

The KDDCup99 dataset is created by simulating normal connections and intrusions or attacks in a military network environment. It was initially used to evaluate supervised network intrusion detectors (classifiers) in the Third International Knowledge Discovery and Data Mining Tools Competition. For our experiments, data instances with labels "normal", "pod", "guess_passwd" and all other categories with less than 50 instances are kept. This results in a dataset of 972,781 instances, among which 443 instances are marked as outliers. We also made some modifications to the attributes. Categorical attribute "service" is removed, and another categorical attribute "protocol_type" is converted into 3 binary attributes with the one-hot encoding scheme.

Name	Size	Dimensions	Outliers	Percentage
synthetic	1,000,000	10	1000	0.1000
CoverType	581,012	54	1000	0.1720
KDDCup99	972,781	42	443	0.0455

Table 5.2: Overview of the datasets used for evaluation

5.3 Notable Implementation Details

5.3.1 Normalization

Normalization, also known as feature scaling, is generally used as a data preprocessing method to standardize the range of the individual features of data. The motivation of data normalization is to avoid the situation where the attributes of data contribute highly disproportionately to the objective functions. For example, many classifiers and outlier

detection methods use the Euclidean distance between instances. One of the attributes will dominate the distance if it has a much broader range compared to others. In outlier detection, distance plays a significant role thus normalization is usually necessary.

Some methods for normalization are listed as below. In our experiments, min-max normalization is adopted.

Min-max normalization:

$$v' = \frac{v - \min(v)}{\max(v) - \min(v)},$$

where v is the original value of an attribute of an instance and v' is the normalized value. $\texttt{min}(v)$ and $\texttt{max}(v)$ are the minimum value and maximum value of the attribute in the entire dataset.

Mean normalization:

$$v' = \frac{v - \texttt{average}(v)}{\max(v) - \min(v)},$$

where $\texttt{average}(v)$ is the mean value of the attribute.

Standardization:

$$v' = \frac{v - \texttt{average}(v)}{\sigma},$$

where σ is the standard deviation. The normalized data by standardization have zero-mean and unit-variance for each feature.

5.3.2 Duplicate Handling

Duplicates are instances in a dataset that have exactly the same values on every attribute. It can cause issues for methods based on nearest-neighbors. Take LOF for instance. The local reachability density of an instance becomes infinite if there are more than k duplicates sharing the same spatial coordinate as that instance.

There are a few solutions being used in the past. The first and most simple solution is removing the duplicates. However, the drawback is that important information may also be removed and normal instances with many duplicates in the original dataset may be identified as outliers as a result. The second solution is to define the density based on the number of instances as introduced in [197]. Another solution is described in [30] as using the k-distinct-distance, which is to find the neighborhood that contains instances of

k distinct spatial coordinates. Goldstein et al. [59] have used a similar approach in which the duplicates are removed but the number of these duplicates are kept as a weight, which is later used in computing the densities. The approach we adopted to handle duplicates is similar to k-distinct-distance in which we count the nearest neighbors with zero-distance only once if they exist and find other $k-1$ neighbors regardless of whether they are duplicates or not. For example, we set k to 20 and there are 10 other instances sharing the same spatial coordinate with the target instance. We include all of the 10 instances in the k-nearest-neighborhood and also search for the rest of the 19 neighbors. There can be duplicates in those 19 neighbors but no special treatment is performed on them. With this approach, we have managed to avoid infinite densities in a very simple way.

5.4 Evaluation Metrics

To evaluate the efficiency of the proposed methods, we use the elapsed execution time as the metric. We want to compare the execution time of MR-LOF and MR-LOF-LSH-CU as well as to figure out how much time the cross-partition updating process costs. We also want to know how the execution time diminishes on different datasets as the scale of parallelism builds up.

We use recall as the metric to evaluate the accuracy of MR-LOF-LSH and MR-LOF-LSH-CU. The recall is defined as the number of correctly detected outliers divided by the number of total outliers in the dataset. As stated previously, we have set different numbers of outliers for the three datasets. The outliers are selected by picking the top LOF scores among the entire dataset, which are generated by the original centralized version of the LOF algorithm. Since the result of MR-LOF is the same as the centralized LOF, we only apply the accuracy evaluation on MR-LOF-LSH and MR-LOF-LSH-CU, to test how well they approximate the original results.

5.5 Experimental Results

5.5.1 Elapsed Execution Time

In this subsection, we first contrast the execution time of MR-LOF-LSH-CU and MR-LOF against centralized LOF on a cluster of 10 nodes. Then we compare the execution time of

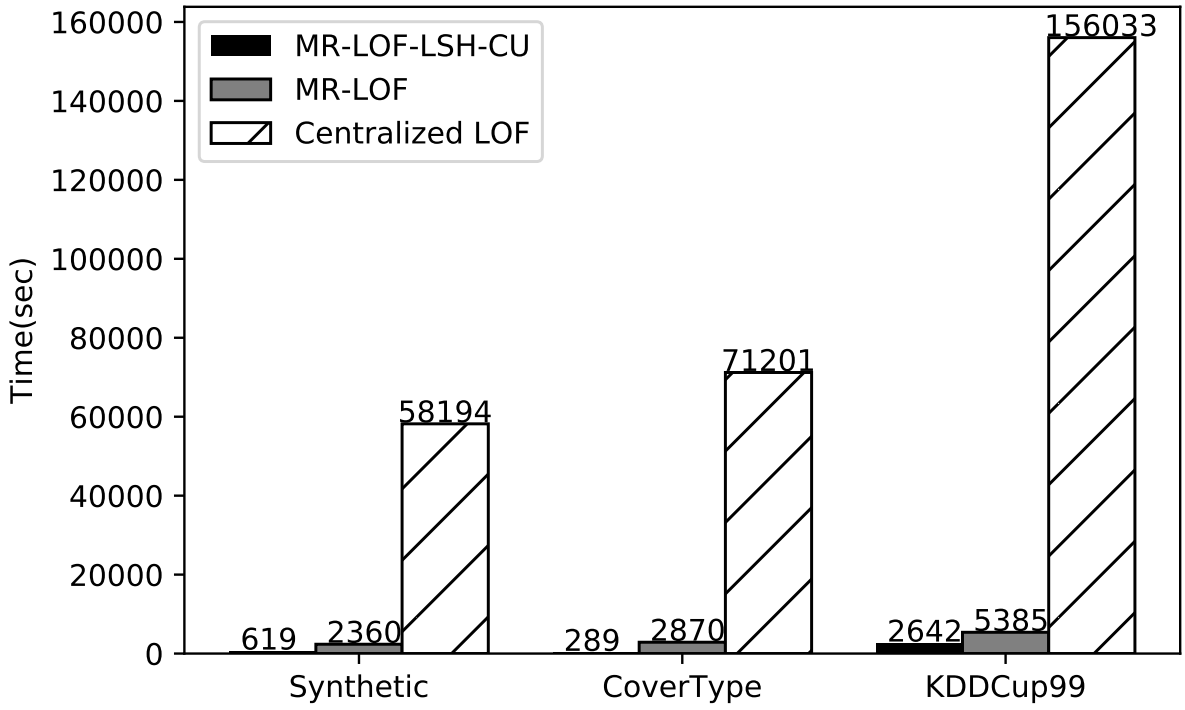
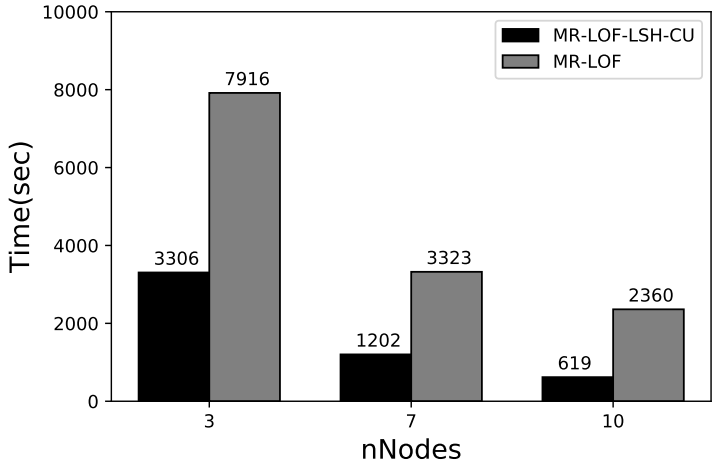


Figure 5.1: Execution time comparison on a cluster of 10 nodes

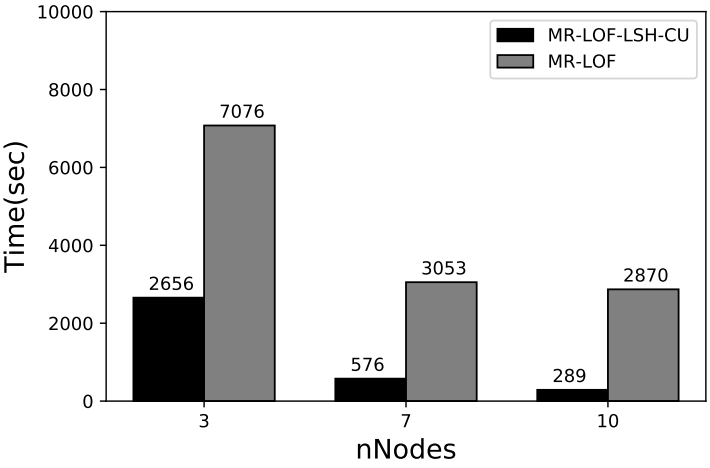
MR-LOF-LSH-CU and MR-LOF on various cluster sizes as well as evaluate the scalability of MR-LOF-LSH-CU. To ensure decent usage of the cluster resources such as virtual CPUs and memory, we set $nPartition$ to $2 \times nNodes$.

Figure 5.1 illustrates the elapsed execution time difference between MR-LOF-LSH-CU, MR-LOF and the centralized LOF in a scenario where the two distributed methods MR-LOF-LSH-CU and MR-LOF, are tested on a cluster of 10 nodes. The gain in runtime is obvious. MR-LOF is shown to reduce the execution time by a factor of 24 to 29 times, compared to the centralized LOF. MR-LOF-LSH further minimizes the execution time by a factor of 2.4 to 9.9 times compared to MR-LOF.

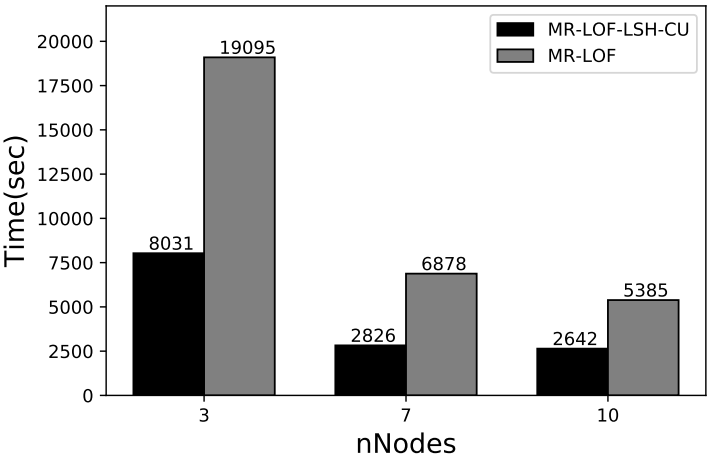
Figure 5.2 demonstrates the execution time difference between MR-LOF-LSH-CU and MR-LOF with 3, 7 and 10 nodes. It shows that MR-LOF-LSH-CU guarantees at least half of execution time to be reduced compared to MR-LOF. The most highly contrasting example is CoverType with 10 nodes, where MR-LOF takes nearly 10-folds of the time MR-LOF-LSH-CU needs. The performance gain results from the reduced shuffle cost and computation complexity. Instead of broadcasting every partition to every other node, MR-LOF-LSH-CU computes its LOF scores locally and only broadcasts a small number of candidates.



(a) Synthetic



(b) CoverType



(c) KDDCup99

Figure 5.2: Execution time comparison varying the cluster size

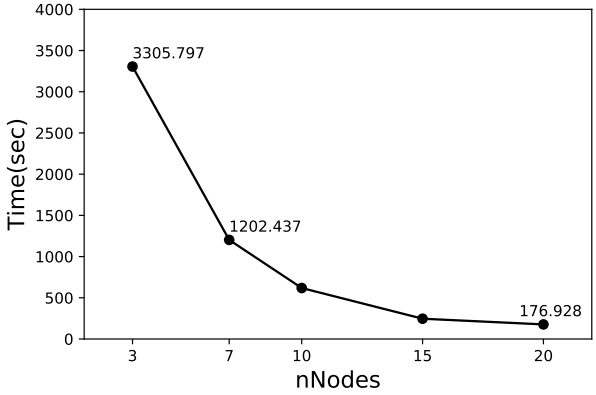
Figure 5.2 also demonstrates that MR-LOF enjoys less execution time improvement with an increasing size of the cluster. Take the CoverType dataset for example, from 3 nodes to 7 nodes, the execution time of MR-LOF-LSH-CU decreases by 78% while that of MR-LOF is only 56%. When the cluster scale increases to 10 nodes, the execution time reduction ratios become 50% versus 6%.

As mentioned in Section 4.1.4, although MR-LOF's time complexity for computation decreases linearly with p, the number of partitions, its shuffle cost increases linearly with p. This means that with a fixed size of the dataset, using more slave nodes gives rise to linearly more data to be transferred across the network. Data transferring (shuffling) is expensive for distributed computing because it involves data serialization, loading data to disk, transmitting over the network, loading the data back to memory and serialization. Therefore, MR-LOF can have difficulty scaling to high parallelism.

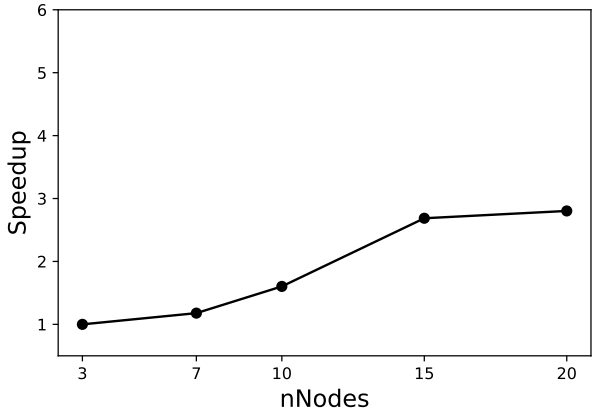
We have also studied the scalability of MR-LOF-LSH-CU. The left column of Figure 5.3 demonstrates the execution times concerning an increasing number of slave nodes in a cluster. As expected, the execution time drops sharply as the number of nodes increases.

To evaluate how well the algorithm scales, we have designed the speedup graph as presented in the right column of Figure 5.3. For an cluster with n nodes employed, the speedup is defined as $S_n = \frac{T_b}{T_n} \cdot \frac{b}{n}$, where T_n is the execution time of n nodes and T_b is the time of b nodes as a baseline. We set b to 3 for our experiments. Semantically, S_n represents the "average gain" of the nodes in terms of execution time reduction, against the baseline case of b nodes. For instance, suppose a cluster of 3 nodes needs 20s to run the algorithm on a dataset. If the time for 6 nodes is reduced by half, the speedup for 6 nodes is $20/10 \times 3/6 = 1$, which means in the case of 6 nodes, the "average gain" of the nodes in terms of execution time reduction is the same as when there are only 3 nodes. However, if the execution time in the case of 6 nodes is one-fourth of 10 nodes, the speedup becomes $20/5 \times 3/6 = 2$, which means the "average gain" of 6 nodes is twice as much as when there are 3 nodes.

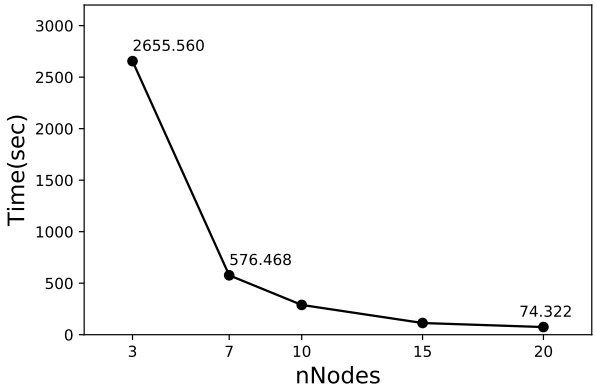
According to our previous conclusions on complexity analysis, the major portions of the shuffle cost and time complexity for MR-LOF-LSH-CU are $N \cdot m$ and $O\left(\left(\frac{N}{p}\right)^2\right)$ respectively. The shuffle cost does not involve p while the time complexity for computation involves p^{-2}. Hence we could conjecture that the speedup should exhibit a linear-like relationship with the number of nodes. As a result, Synthetic dataset basically conforms to this conjecture and CoverType dataset demonstrates a nice linear relationship.



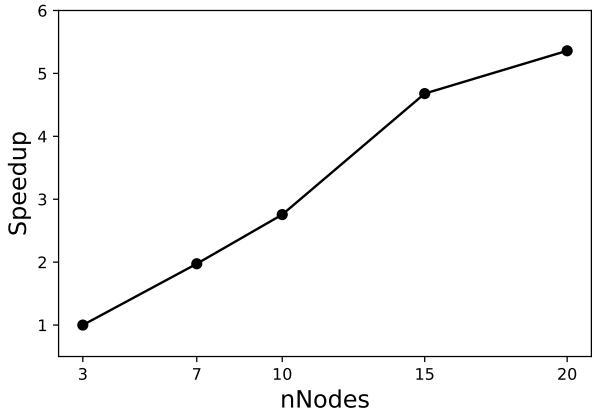
(a) Synthetic



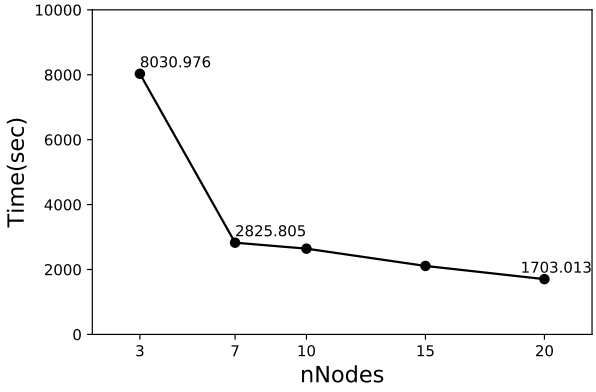
(b) Synthetic Speedup



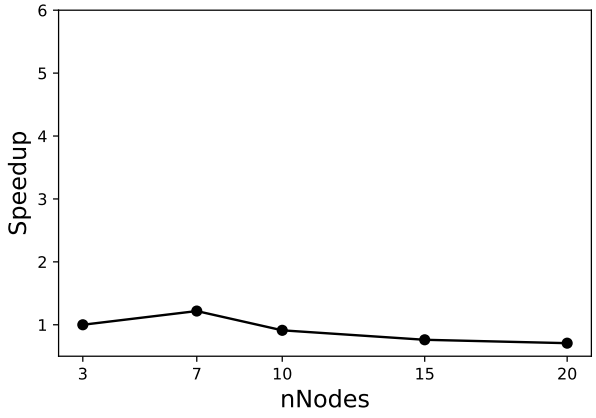
(c) CoverType



(d) CoverType Speedup



(e) KDDCup99



(f) KDDCup99 Speedup

Figure 5.3: Test of Scalability of MR-LOF-LSH-CU

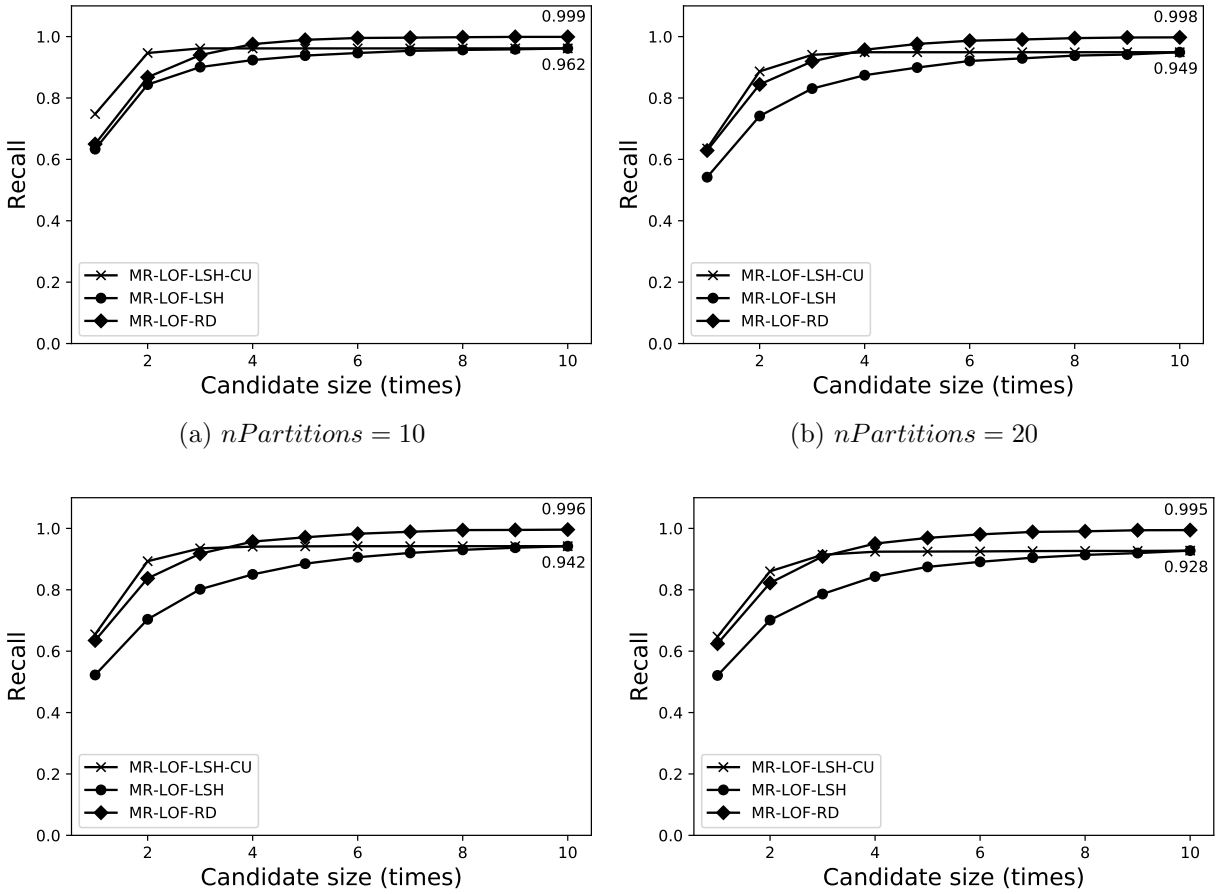
On the contrary, the speedup of KDDCup99 dataset deteriorates slightly as the scale of the cluster increases. One of the possible factors contributing to this result might be the imbalanced partitions. We have discovered in our experiments that the data partitions generated with LSH-partitioning can sometimes be very imbalanced, making the significantly larger partitions take much longer time to be processed thus elongating the overall execution time.

This problem is called the straggler tasks. Some methods in literature have been proposed to mitigate this issue [198]. In our LSH-partitioning method, we have made use of TeraSort to sort the data and balance the loads, based on the approximate data distribution information obtained through sampling. However, data imbalance still happens and can lead to very different results of elapsed execution time. As the size of the dataset escalates, the imbalance can get more dramatic. Therefore, a better load balancing technique is needed for LSH data partitioning, which will be investigated in our future work. A straightforward solution would be to infer the boundaries in the exactly even segementation based on the result of TeraSort, then perform repartitioning according to these boundaries. However the repartitioning incurs the shuffling of the entire dataset. Another way of optimization would be to improve the quality of the distribution information by using a larger sample size or a better sampling method, etc.

5.5.2 Evaluation of Recall with Different Numbers of Partitions and Candidate Sizes

To evaluate the accuracy on different datasets, we vary the number of partitions from 10 to 40 and extend the candidate size from 1-fold of the outlier size to 10-fold. The results are detailed by Figure 5.4, Figure 5.5 and Figure 5.6. Note that in our actual experiments, 10-fold candidates are selected as the output of MR-LOF-LSH, which are sorted in descending order by the LOF. The LOFs of these 10-fold candidates are then updated by cross-partition updating. We take as many as $candidateTimes$-fold top outliers from the output 10-fold outliers, with $candidateTimes$ from 1 to 10, to get the recall ratios. That is why MR-LOF-LSH and MR-LOF-LSH-CU always reach the same point when $candidateTimes$ turns 10 in these figures.

As we can see, in general, the patterns of the curves for the 3 datasets remain basically the same, regardless of $nPartitions$. What usually changes is the values of the starting point and the ending point. For instance, with regard to CoverType, MR-LOF-LSH-CU



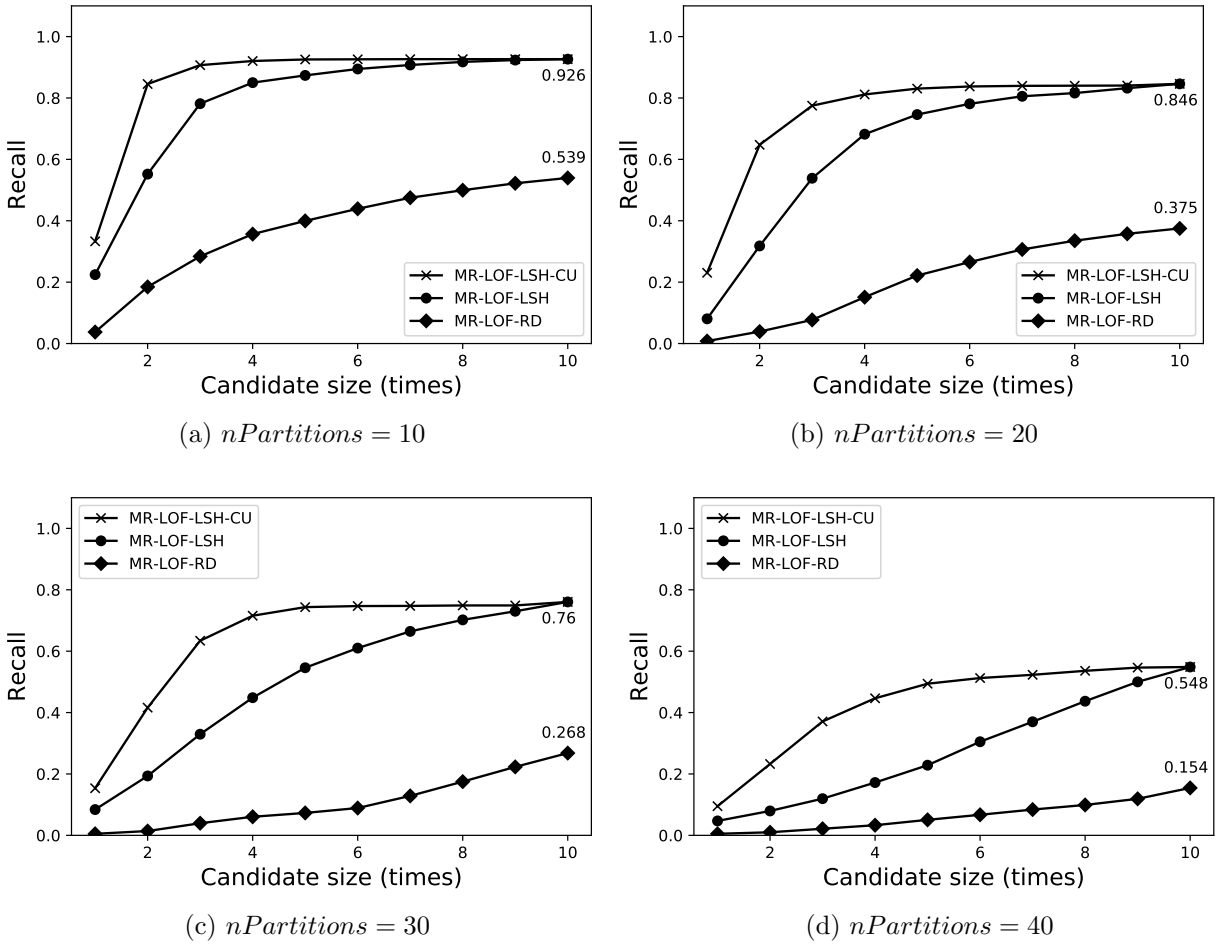
(a) $nPartitions = 10$

(b) $nPartitions = 20$

(c) $nPartitions = 30$

(d) $nPartitions = 40$

Figure 5.4: Test of recall on Synthetic dataset against different settings of $nPartitions$ and $candidateTimes$



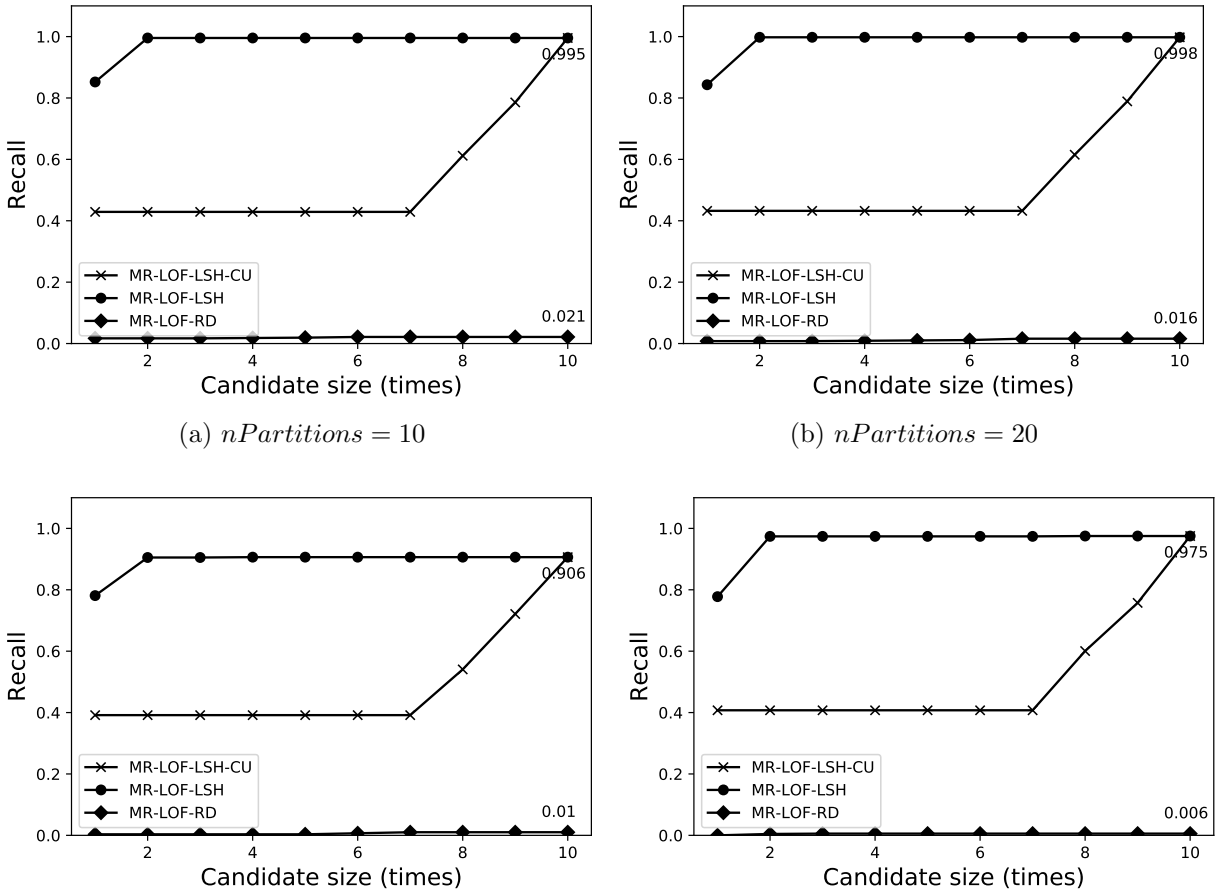
(a) $nPartitions = 10$

(b) $nPartitions = 20$

(c) $nPartitions = 30$

(d) $nPartitions = 40$

Figure 5.5: Test of recall on CoverType dataset against different settings of $nPartitions$ and $candidateTimes$



(a) $nPartitions = 10$

(b) $nPartitions = 20$

(c) $nPartitions = 30$

(d) $nPartitions = 40$

Figure 5.6: Test of recall on KDDCup99 dataset against different settings of $nPartitions$ and $candidateTimes$

constantly outperforms MR-LOF-LSH, which always stays ahead of MR-LOF-RD but their recall ratios all decrease with an increasing number of partitions. This means that we can reasonably predict the patterns of the results as the cluster escalates in size.

The results of CoverType dataset exhibit a clear trend that the recall of all the three algorithms gradually drops as the number of partitions rises. The recall for 10-fold candidate drops from 0.926 in 10 partitions to 0.548 in 40 partitions. Also, the candidate size needed to reach the peak or approximate the peak increases as the $nPartitions$ grow, from 2 in 10 partitions to around 5 in 40 partitions. This should be the expected trend for most datasets because generally speaking, the more partitions the original dataset is divided into, the more likely instances are separated from their actual k-nearest neighbors, thus leading to a less accurate result of LOF scores. However, the benefit of raising the number of the partitions is obvious: increase the level of parallelism and reduce the overall execution time. Interestingly, KDDCup99 and Synthetic do not very much conform to this trend. The recall of KDDCup99 only drops slightly with 1-fold candidate size but is not affected with larger candidate size, against the escalation of $nPartitions$, while the Synthetic dataset does not seem to be affected at all. We conjecture that this is due to the dense distribution of inliers for KDDCup99 and Synthetic, which makes data points able to find most of their actual neighbors locally or the local neighbors are in resemblant positions as the actual neighbors.

MR-LOF-RD performs poorly on CoverType and especially on KDDCup99, compared to MR-LOF-LSH. As for CoverType, with 10 partitions, the recall of MR-LOF-RD is 0.539 at maximum and it drops to 0.154 with 40 partitions while the recall of MR-LOF-LSH-CU is almost always several times higher than that. As for KDDCup99, the recall of MR-LOF-RD is steadily around 0.01. This indicates that the LSH-partitioning is very effective at mapping similar data instances into close hash values for those two datasets thus leads to considerably accurate approximations of LOF. However, this does not hold for the Synthetic dataset, according to Figure 5.4. Instead, the MR-LOF-RD constantly stays ahead of MR-LOF-LSH and surpasses MR-LOF-LSH-CU at a point. We believe that this is due to the highly dense distributions of the inliers instances in the Synthetic dataset, leaving outliers easy to detect in any way of partitioning.

Finally, we look at how MR-LOF-LSH-CU improves the recall. For CoverType and Synthetic, MR-LOF-LSH-CU performs better when the candidate size is small. It makes the recall reach the peak or draw close to the peak value with fewer candidates. This means that when MR-LOF-LSH-CU is applied, we can have more confidence in the tip-top

candidates. This property comes useful in cases where a very limited number of output outliers is required due to the expensive analysis cost on potential outliers and limited resources. However, MR-LOF-LSH-CU does not perform well with the KDDCup99 dataset, where MR-LOF-LSH's recall reaches nearly 100% with only 2-fold candidates in all the cases of different $nPartitions$. Perhaps for some datasets we can consider using the union of the tip-top candidates from both MR-LOF-LSH-CU and MR-LOF-LSH as the final outlier output in practical scenarios.

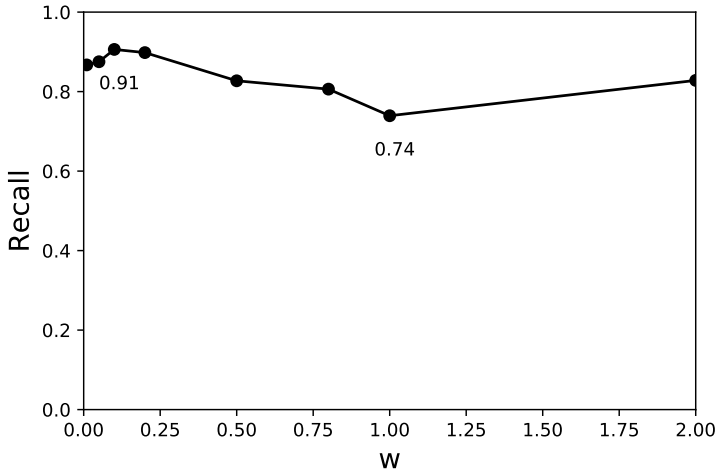
5.5.3 Impact of Varying LSH-related Parameters on Recall

Previously we have seen the effect of varying candidate size and the number of partitions on the recall. Here we also evaluate MR-LOF-LSH by changing w, also known as the "width" of the LSH function, and the number of LSH functions. The candidate size is set to only 2-fold of the outlier size. 10 partitions are used for each experiment.

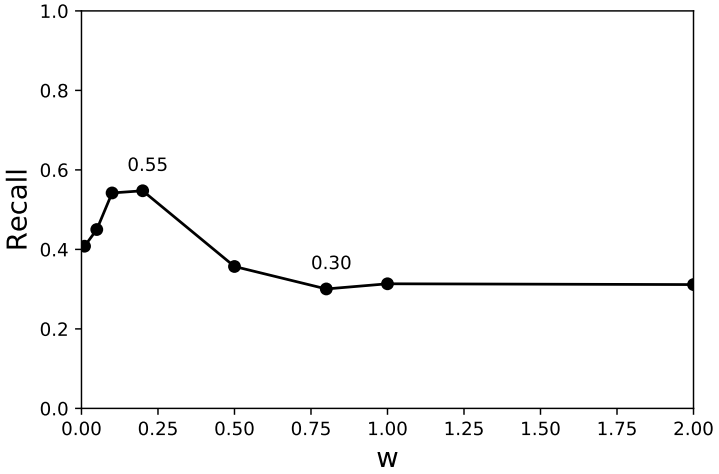
Note that choosing the best w is not an easy problem [199]. With an inappropriately large w, the number of resultant LSH bins can be too small, and thus the LSH bins are unable to represent the dissimilarities of the data instances effectively. An extreme case is that when w is large enough, all the output hash values are zero. On the other hand, if w is overly small, the consequence could be that some data instances with minor Euclidean distance to each other end up with very different hash values. Our choice of w in previous experiments is based on the empirical results of this subsection.

It is generally believed that the effect of different w is highly correlated with the data distribution. Hence we can expect the changes on w affect each dataset differently. Figure 5.7 illustrates the effect of different choices of w on the recall. The recall for each dataset fluctuates between a range: 0.74 to 0.91 for Synthetic, 0.30 to 0.55 for CoverType and 0.81 to 0.99 for KDDCup99. Empirically speaking, the optimal w for Synthetic resides around 0.1 and 0.2. It is also around 0.1 and 02 for CoverType and 0.8 to 1.0 with regard to KDDCup99.

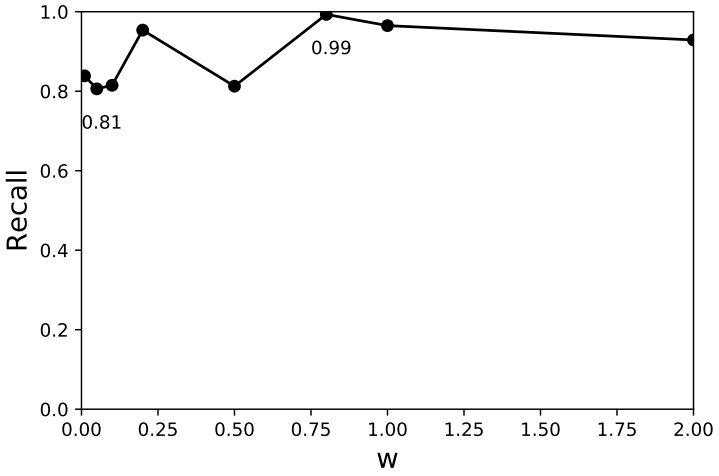
Figure 5.8 shows the effect of varying $nLSHFunctions$ on the recall. To our surprise, different values of $nLSHFunctions$ do not seem to have a big impact on the recall of outliers detection. Additionally, there are seemingly no conspicuous patterns that can be induced from the results: using one LSH function for the first layer does not necessarily outperforms having 25 LSH functions. More investigation on how to obtain the optimal value of $nLSHFunctions$ will be performed in the future work.



(a) Synthetic

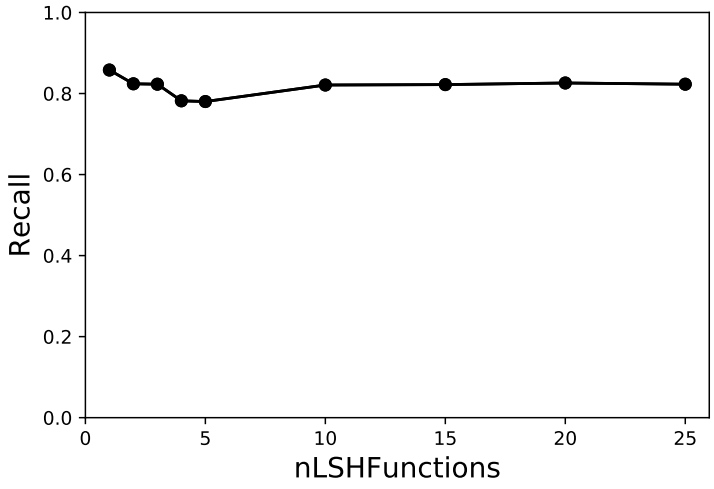


(b) CoverType

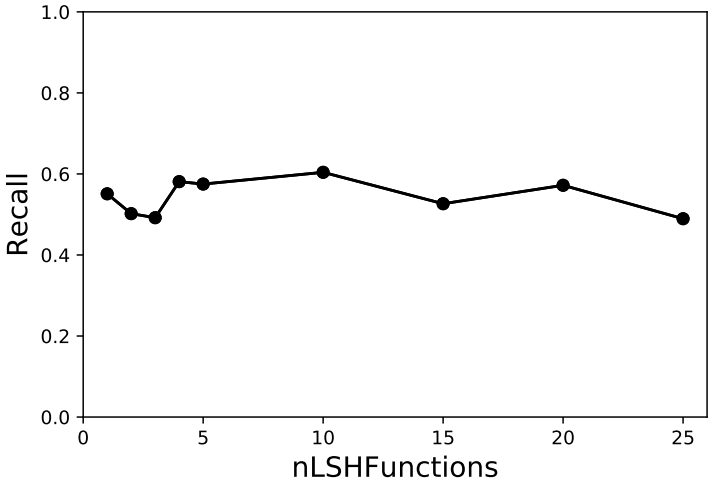


(c) KDDCup99

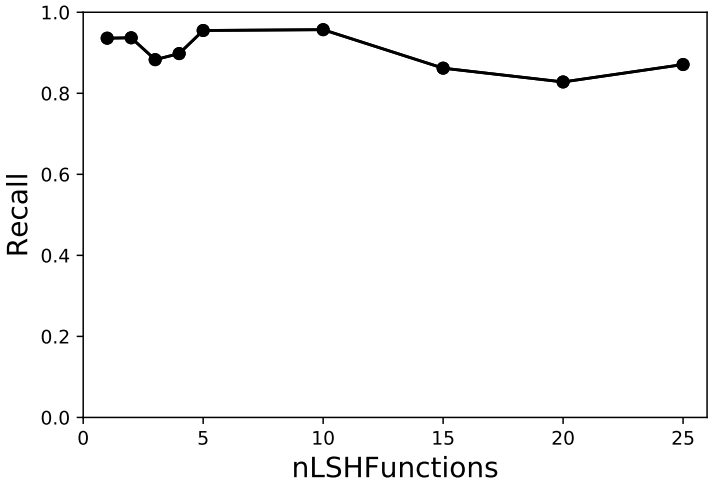
Figure 5.7: Varying parameter w



(a) Synthetic



(b) CoverType



(c) KDDCup99

Figure 5.8: Varying parameter $nLSHFunctions$

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we have presented a baseline distributed solution for LOF in Spark: MR-LOF and then an efficient distributed algorithm in Spark: MR-LOF-LSH that can detect top-N outliers with high confidence.

MR-LOF, implemented with Apache Spark, produces exactly the same LOF result as the centralized LOF algorithm. In order to do so, the dataset in numerous partitions is broadcast across the cluster and with the MapReduce paradigm, the exact k-NN are discovered for each data instance. Built upon the k-NN, the k-distance, local reachability density (LRD) and the final LOF for each data instance are computed in a step-by-step fashion. However, one machine only contains a portion of the dataset while each step requires global information. Hence, high shuffle cost is inevitable.

MR-LOF-LSH is an approximate method expediting the data processing while still producing promising results on the top outliers. MR-LOF-LSH utilizes a two-layered LSH strategy to partition the dataset so that data instances closer to each other in terms of Euclidean distance are more likely to be distributed to the same partition. Based on this property of the partitions, the approximate LOF for each data instance is computed locally without communicating with other machines in the cluster. To provide more reliable results, we verify the top candidates selected based on the locally computed LOF through a process named cross-partition updating, in which the actual k-NN of the top candidates are found in the entire dataset.

We have also conducted experiments to evaluate the efficiency and effectiveness of the proposed algorithms. We have measured the elapsed execution time for the centralized LOF as well as MR-LOF and MR-LOF-LSH on different cluster sizes. The results reveal that with a cluster of only 3 nodes, MR-LOF can achieve a factor of 7 to 10 time speedup compared to the centralized LOF and 24 to 29 time speedup when we raise the cluster size to 10 nodes. MR-LOF-LSH further reduces the execution time by a factor of 2.4 to 9.9 times compared to MR-LOF, thanks to the fully distributed computation of local LOF. The results also highlight that MR-LOF-LSH scales well with the cluster size escalating on most of the datasets we have used.

Additionally, we use recall as the metric to evaluate how well MR-LOF-LSH approximates the original LOF algorithm in terms of identified top outliers, namely the ratio of correctly identified top outliers to the number of top outliers. The results show that even though the recall drops as the number of partitions increases, MR-LOF-LSH can achieve a recall of around 0.9 on all the datasets with 20 partitions. The recall remains as high as over 0.9 on CoverType dataset and KDDCup99 dataset even with 40 partitions. To highlight the benefit of LSH data partitioning, we compared MR-LOF-LSH with MR-LOF-RD, the latter using randomly partitioned dataset. The results show that MR-LOF-LSH significantly outperforms MR-LOF-RD in terms of the recall against different parameter settings, except on the Synthetic dataset, where the random partitioning performs almost equally with LSH partitioning. We believe this is due to the highly dense distributions of inliers in the Synthetic dataset. The tests also show that on most of the datasets, the cross-partition updating strategy can effectively improve the recall with a limited candidate size.

In conclusion, MR-LOF-LSH along with cross-partition updating is a highly scalable and reliable solution for distributed LOF which enables outlier detection in a large scale of datasets.

6.2 Future Work

Before concluding this report, we would like to list some ideas and directions on how this work can be extended in the future.

- **Leveraging pruning strategies to accelerate the discovery of top-N outliers** For many applications, the most interest is laid on the most extreme outliers, those with the highest LOF scores. Our experiments conform to this setting, and a fixed

number of outliers are made the target. In MR-LOF-LSH-CU, the top candidates with the highest approximate LOF scores are picked out and sent for cross-partition updating. However, it is not necessary to compute all the LOF scores of each individual instances in order to single out the top candidates. A number of pruning strategies [114, 186] have been proposed in the past to accelerate the top-N outlier detection, with which some data instances can be eliminated before their LOF scores are computed. It would be very useful to integrate pruning methods into our framework. For example, the initial boundaries related to pruning can be obtained through global computation then are shared among every machine in the cluster. With these boundaries used in pruning, the process of local computation of LOF on each machine can be more efficient.

- **Applying more efficient k-NN search techniques to accelerate the local computation of LOF**

  In our implementation, the basic nested loop method is used to search for k-NN in a way of sequential scanning. As mentioned before, the k-NN search technique is a pluggable component of the MR-LOF-LSH framework and thus can be replaced by more efficient techniques in the hope of higher efficiency. For datasets of low to medium dimensionality, an efficient indexing structure, such as R-trees [103] and X-trees [104], would be useful. Moreover, approximate nearest neighbor techniques [98, 177] can be employed in the case of high-dimensional datasets, despite the fact that the approximate k-NN techniques make the result more "approximate". In order to produce more accurate and reliable results, a potentially feasible direction is to communicate the intermediate results with other machines in the cluster to obtain a "global" version of some variables.

- **Improving LSH data partitioning**

  The first issue that needs more investigation is how to determine the optimal parameters for LSH, mainly w and $nLSHFunctions$. Our experimental results show that the w producing the best performance varies with different datasets. But is there a way to automatically find such a w based on the available dataset? In addition to that, we have not yet drawn a conclusion on the pattern of how the number of LSH functions affect the resulting recall and how to choose the best $nLSHFunctions$ accordingly. The second issue about our LSH partitioning method is that a better load-balancing strategy is required. The current partitioning method uses TeraSort to sort the data instances based on the second-layer LSH value and conduct the partitioning

accordingly. This may have led to the performance deterioration in the experiments conducted on KDDCup99 dataset as mentioned previously in Section 5.5.1.

- **Developing a distributed outlier detection ensemble**
  Ensembling is a way of optimizing the trade-off between bias and variance by combining the results of multiple models. Ensemble analysis is well studied in supervised machine learning [200, 201]. In the unsupervised context, ensembling can also be useful, based on the fact that there exists unknown ground truth even though the training data is unlablled [202]. Ensembling is very compatible with the distributed setting because different models can be distributedly trained in parallel. Therefore, it would be interesting to develop a distributed outlier detection ensemble hierarchy, combining multiple outlier detectors of different types (e.g., k-NN based, clustering-based, projection-based, one-class random forest, etc.). Moreover, some recent works [27, 80, 90, 203] have demonstrated the advantage of training the outlier detectors with subsamples of the available data. The distributed data storage is very suitable for subsampling. Thus it would be beneficial and convenient to incorporate the subsampling technique into the distributed outlier detection ensemble architecture.

- **Applying effective dimension reduction techniques to speed up the outlier detection**
  When the number of dataset dimensions becomes high, the extended runtime becomes a problem. Thus it is very useful to reduce the data dimensions while still preserving important information in preprocessing. Some well-known dimension reduction methods include principal component analysis (PCA) [204], autoencoder neural networks [205, 206], locally linear embedding [207], sparse random projection [208], etc. An interesting question is how to tailor the dimension-reduction specifically for a given outlier detection method.

References

[1] "Apache spark cluster mode overview," [Online]. Available: https://spark.apache.org/docs/latest/cluster-overview.html, 2018, accessed on: Octobor, 2018.

[2] H.-P. Kriegel, A. Zimek *et al.*, "Angle-based outlier detection in high-dimensional data," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2008, pp. 444–452.

[3] H. A. Oliveira, E. F. Nakamura, A. A. Loureiro, and A. Boukerche, "Error analysis of localization systems for sensor networks," in *Proceedings of the 13th annual ACM international workshop on Geographic information systems*. ACM, 2005, pp. 71–78.

[4] A. Boukerche and X. Li, "An agent-based trust and reputation management scheme for wireless sensor networks," in *GLOBECOM'05. IEEE Global Telecommunications Conference, 2005.*, vol. 3. IEEE, 2005, pp. 5–pp.

[5] Y. Ren and A. Boukerche, "Modeling and managing the trust for wireless and mobile ad hoc networks," in *2008 IEEE International Conference on Communications*. IEEE, 2008, pp. 2129–2133.

[6] A. Boukerche, "Performance comparison and analysis of ad hoc routing algorithms," in *Conference Proceedings of the 2001 IEEE International Performance, Computing, and Communications Conference (Cat. No. 01CH37210)*. IEEE, 2001, pp. 171–178.

[7] A. Boukerche, H. A. Oliveira, E. F. Nakamura, and A. A. F. Loureiro, "A novel location-free greedy forward algorithm for wireless sensor networks," in *2008 IEEE International Conference on Communications*. IEEE, 2008, pp. 2096–2101.

[8] A. Boukerche, H. A. Oliveira, E. F. Nakamura, and A. A. Loureiro, "Dv-loc: a scalable localization protocol using voronoi diagrams for wireless sensor networks," *IEEE Wireless Communications*, vol. 16, no. 2, pp. 50–55, 2009.

[9] A. Boukerche and X. Fei, "A coverage-preserving scheme for wireless sensor network with irregular sensing range," *Ad hoc networks*, vol. 5, no. 8, pp. 1303–1316, 2007.

[10] T. Antoniou, I. Chatzigiannakis, G. Mylonas, S. Nikoletseas, and A. Boukerche, "A new energy efficient and fault-tolerant protocol for data propagation in smart dust networks using varying transmission range," in *Proceedings of the 37th annual symposium on Simulation*. IEEE Computer Society, 2004, p. 43.

[11] A. Boukerche, I. Chatzigiannakis, and S. Nikoletseas, "A new energy efficient and fault-tolerant protocol for data propagation in smart dust networks using varying transmission range," *Computer communications*, vol. 29, no. 4, pp. 477–489, 2006.

[12] A. Boukerche and D. Turgut, "Secure time synchronization protocols for wireless sensor networks," *IEEE Wireless Communications*, vol. 14, no. 5, pp. 64–69, 2007.

[13] D.-Y. Yeung and C. Chow, "Parzen-window network intrusion detectors," in *Object recognition supported by user interaction for service robots*, vol. 4. IEEE, 2002, pp. 385–388.

[14] R. Gwadera, M. J. Atallah, and W. Szpankowski, "Reliable detection of episodes in event sequences," *Knowledge and Information Systems*, vol. 7, no. 4, pp. 415–437, 2005.

[15] M. Atallah, W. Szpankowski, and R. Gwadera, "Detection of significant sets of episodes in event sequences," in *Fourth IEEE International Conference on Data Mining (ICDM'04)*. IEEE, 2004, pp. 3–10.

[16] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *computers & security*, vol. 28, no. 1-2, pp. 18–28, 2009.

[17] R. J. Bolton, D. J. Hand *et al.*, "Unsupervised profiling methods for fraud detection," *Credit Scoring and Credit Control VII*, pp. 235–255, 2001.

[18] S. Thiprungsri, M. A. Vasarhelyi *et al.*, "Cluster analysis for anomaly detection in accounting data: An audit approach," *The International Journal of Digital Accounting Research*, vol. 11, pp. 69–84, 2011.

[19] C. Phua, D. Alahakoon, and V. Lee, "Minority report in fraud detection: classification of skewed data," *Acm sigkdd explorations newsletter*, vol. 6, no. 1, pp. 50–59, 2004.

[20] W.-K. Wong, A. W. Moore, G. F. Cooper, and M. M. Wagner, "Bayesian network anomaly pattern detection for disease outbreaks," in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 2003, pp. 808–815.

[21] J. Lin, E. Keogh, A. Fu, and H. Van Herle, "Approximations to magic: Finding unusual medical time series," in *18th IEEE Symposium on Computer-Based Medical Systems (CBMS'05)*. Citeseer, 2005, pp. 329–334.

[22] R. Fujimaki, T. Yairi, and K. Machida, "An approach to spacecraft anomaly detection problem using kernel feature space," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 2005, pp. 401–410.

[23] V. Vercruyssen, W. Meert, G. Verbruggen, K. Maes, R. Bäumer, and J. Davis, "Semi-supervised anomaly detection with an application to water analytics," in *Proceedings/IEEE International Conference on Data Mining*. IEEE, 2018.

[24] Y.-L. Tsou, H.-M. Chu, C. Li, and S.-W. Yang, "Robust distributed anomaly detection using optimal weighted one-class random forests," in *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2018, pp. 1272–1277.

[25] Y. Djenouri, A. Belhadi, J. C.-W. Lin, D. Djenouri, and A. Cano, "A survey on urban traffic anomalies detection algorithms," *IEEE Access*, vol. 7, pp. 12 192–12 205, 2019.

[26] M. B. Younes and A. Boukerche, "A performance evaluation of an efficient traffic congestion detection protocol (ecode) for intelligent transportation systems," *Ad Hoc Networks*, vol. 24, pp. 317–336, 2015.

[27] G. Pang, K. M. Ting, and D. Albrecht, "Lesinn: Detecting anomalies by identifying least similar nearest neighbours," in *2015 IEEE international conference on data mining workshop (ICDMW)*. IEEE, 2015, pp. 623–630.

[28] H. Du, S. Zhao, D. Zhang, and J. Wu, "Novel clustering-based approach for local outlier detection," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2016, pp. 802–811.

[29] C. Zhou and R. C. Paffenroth, "Anomaly detection with robust deep autoencoders," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 665–674.

[30] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *ACM sigmod record*, vol. 29. ACM, 2000, pp. 93–104.

[31] A. Lazarevic and V. Kumar, "Feature bagging for outlier detection," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 2005, pp. 157–166.

[32] H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek, "Loop: local outlier probabilities," in *Proceedings of the 18th ACM conference on Information and knowledge management*. ACM, 2009, pp. 1649–1652.

[33] H.-P. Kriegel, P. Kroger, E. Schubert, and A. Zimek, "Interpreting and unifying outlier scores," in *Proceedings of the 2011 SIAM International Conference on Data Mining*. SIAM, 2011, pp. 13–24.

[34] E. Schubert, R. Wojdanowski, A. Zimek, and H.-P. Kriegel, "On evaluation of outlier rankings and outlier scores," in *Proceedings of the 2012 SIAM International Conference on Data Mining*. SIAM, 2012, pp. 1047–1058.

[35] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers, "Big data: The next frontier for innovation, competition, and productivity," 2011.

[36] Y. Yan, L. Cao, C. Kulhman, and E. Rundensteiner, "Distributed local outlier detection in big data," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 1225–1234.

[37] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[38] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in *Proceedings of the 5th European conference on Computer systems*. ACM, 2010, pp. 265–278.

[39] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[40] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, *Learning spark: lightning-fast big data analysis.* " O'Reilly Media, Inc.", 2015.

[41] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing.* ACM, 1998, pp. 604–613.

[42] A. Z. Broder, "On the resemblance and containment of documents," in *Compression and complexity of sequences 1997. proceedings.* IEEE, 1997, pp. 21–29.

[43] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig, "Syntactic clustering of the web," *Computer Networks and ISDN Systems*, vol. 29, no. 8, pp. 1157–1166, 1997.

[44] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing.* ACM, 2002, pp. 380–388.

[45] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the twentieth annual symposium on Computational geometry.* ACM, 2004, pp. 253–262.

[46] A. Dasgupta, R. Kumar, and T. Sarlós, "Fast locality-sensitive hashing," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2011, pp. 1073–1081.

[47] J. Gan, J. Feng, Q. Fang, and W. Ng, "Locality-sensitive hashing scheme based on dynamic collision counting," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data.* ACM, 2012, pp. 541–552.

[48] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe lsh: efficient indexing for high-dimensional similarity search," in *Proceedings of the 33rd international conference on Very large data bases.* VLDB Endowment, 2007, pp. 950–961.

[49] Y. Wang, S. Parthasarathy, and S. Tatikonda, "Locality sensitive outlier detection: A ranking driven approach," in *Data Engineering (ICDE), 2011 IEEE 27th International Conference on.* IEEE, 2011, pp. 410–421.

[50] Y. Zhang, S. Chen, and G. Yu, "Efficient distributed density peaks for clustering large data sets in mapreduce," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 12, pp. 3218–3230, 2016.

[51] G. S. Manku, A. Jain, and A. Das Sarma, "Detecting near-duplicates for web crawling," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 141–150.

[52] V. M. Zolotarev, *One-dimensional stable distributions*. American Mathematical Soc., 1986, vol. 65.

[53] P. Haghani, S. Michel, and K. Aberer, "Distributed similarity search in high dimensions using locality sensitive hashing," in *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*. ACM, 2009, pp. 744–755.

[54] P. Haghani, S. Michel, P. Cudré-Mauroux, and K. Aberer, "Lsh at large-distributed knn search in high dimensions." in *WebDB*. Citeseer, 2008.

[55] B. Bahmani, A. Goel, and R. Shinde, "Efficient distributed locality sensitive hashing," in *Proceedings of the 21st ACM international conference on Information and knowledge management*. ACM, 2012, pp. 2174–2178.

[56] F. E. Grubbs, "Procedures for detecting outlying observations in samples," *Technometrics*, vol. 11, no. 1, pp. 1–21, 1969.

[57] V. Barnett and T. Lewis, "Outliers in statistical data (probability & mathematical statistics)," 1994.

[58] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[59] M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data," *PloS one*, vol. 11, no. 4, p. e0152173, 2016.

[60] J. Zhang, "Advancements of outlier detection: A survey," *ICST Transactions on Scalable Information Systems*, vol. 13, no. 1, pp. 1–26, 2013.

[61] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," *Data mining and knowledge discovery*, vol. 29, no. 3, pp. 626–688, 2015.

[62] A. Anand, G. Pugalenthi, G. B. Fogel, and P. Suganthan, "An approach for classification of highly imbalanced data using weighting and undersampling," *Amino acids*, vol. 39, no. 5, pp. 1385–1391, 2010.

[63] M. V. Joshi, R. C. Agarwal, and V. Kumar, "Predicting rare classes: Can boosting make any weak learner strong?" in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2002, pp. 297–306.

[64] Y. Sun, M. S. Kamel, A. K. Wong, and Y. Wang, "Cost-sensitive boosting for classification of imbalanced data," *Pattern Recognition*, vol. 40, no. 12, pp. 3358–3378, 2007.

[65] S. Hido, H. Kashima, and Y. Takahashi, "Roughly balanced bagging for imbalanced data," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 2, no. 5-6, pp. 412–426, 2009.

[66] Q. Wang, Z. Luo, J. Huang, Y. Feng, and Z. Liu, "A novel ensemble method for imbalanced data learning: bagging of extrapolation-smote svm," *Computational intelligence and neuroscience*, vol. 2017, 2017.

[67] N. Abe, B. Zadrozny, and J. Langford, "Outlier detection by active learning," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 504–509.

[68] I. Steinwart, D. Hush, and C. Scovel, "A classification framework for anomaly detection," *Journal of Machine Learning Research*, vol. 6, no. Feb, pp. 211–232, 2005.

[69] O. Chapelle, B. Scholkopf, and A. Zien, "Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]," *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 542–542, 2009.

[70] M. M. Moya and D. R. Hush, "Network constraints and multi-objective optimization for one-class classification," *Neural Networks*, vol. 9, no. 3, pp. 463–474, 1996.

[71] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[72] B. Settles, "Active learning literature survey," University of Wisconsin-Madison Department of Computer Sciences, Tech. Rep., 2009.

[73] N. Görnitz, M. Kloft, K. Rieck, and U. Brefeld, "Toward supervised anomaly detection," *Journal of Artificial Intelligence Research*, vol. 46, pp. 235–262, 2013.

[74] D. M. Tax and R. P. Duin, "Support vector data description," *Machine learning*, vol. 54, no. 1, pp. 45–66, 2004.

[75] S. Das, W.-K. Wong, T. Dietterich, A. Fern, and A. Emmott, "Incorporating expert feedback into active anomaly discovery," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 853–858.

[76] S. Boyd, C. Cortes, M. Mohri, and A. Radovanovic, "Accuracy at the top," in *Advances in neural information processing systems*, 2012, pp. 953–961.

[77] T. Pevnỳ, "Loda: Lightweight on-line detector of anomalies," *Machine Learning*, vol. 102, no. 2, pp. 275–304, 2016.

[78] M. A. Siddiqui, A. Fern, T. G. Dietterich, R. Wright, A. Theriault, and D. W. Archer, "Feedback-guided anomaly discovery via online optimization," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 2200–2209.

[79] S. Shalev-Shwartz *et al.*, "Online learning and online convex optimization," *Foundations and Trends® in Machine Learning*, vol. 4, no. 2, pp. 107–194, 2012.

[80] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 2008, pp. 413–422.

[81] K. Wagstaff, C. Cardie, S. Rogers, S. Schrödl *et al.*, "Constrained k-means clustering with background knowledge," in *Icml*, vol. 1, 2001, pp. 577–584.

[82] J. Tang, Z. Chen, A. W.-C. Fu, and D. W. Cheung, "Enhancing effectiveness of outlier detections for low density patterns," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2002, pp. 535–548.

[83] S. Papadimitriou, H. Kitagawa, P. B. Gibbons, and C. Faloutsos, "Loci: Fast outlier detection using the local correlation integral," in *Data Engineering, 2003. Proceedings. 19th International Conference on.* IEEE, 2003, pp. 315–326.

[84] W. Jin, A. K. Tung, J. Han, and W. Wang, "Ranking outliers using symmetric neighborhood relationship," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining.* Springer, 2006, pp. 577–593.

[85] T. R. Bandaragoda, K. M. Ting, D. Albrecht, F. T. Liu, and J. R. Wells, "Efficient anomaly detection by isolation using nearest neighbour ensemble," in *2014 IEEE International Conference on Data Mining Workshop.* IEEE, 2014, pp. 698–705.

[86] S. Ramaswamy, R. Rastogi, and K. Shim, "Efficient algorithms for mining outliers from large data sets," in *ACM Sigmod Record*, vol. 29. ACM, 2000, pp. 427–438.

[87] F. Angiulli and C. Pizzuti, "Fast outlier detection in high dimensional spaces," in *European Conference on Principles of Data Mining and Knowledge Discovery.* Springer, 2002, pp. 15–27.

[88] E. M. Knorr and R. T. Ng, "Algorithms for mining distance-based outliers in large datasets," in *VLDB*, vol. 98. Citeseer, 1998, pp. 392–403.

[89] C. C. Aggarwal, "Outlier analysis," in *Data mining.* Springer, 2015, pp. 237–263.

[90] K. M. Ting, T. Washio, J. R. Wells, and S. Aryal, "Defying the gravity of learning curve: a characteristic of nearest neighbour anomaly detectors," *Machine learning*, vol. 106, no. 1, pp. 55–91, 2017.

[91] M.-F. Jiang, S.-S. Tseng, and C.-M. Su, "Two-phase clustering process for outliers detection," *Pattern recognition letters*, vol. 22, no. 6-7, pp. 691–700, 2001.

[92] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[93] Z. He, X. Xu, and S. Deng, "Discovering cluster-based local outliers," *Pattern Recognition Letters*, vol. 24, no. 9-10, pp. 1641–1650, 2003.

[94] M. Amer and M. Goldstein, "Nearest-neighbor and clustering based anomaly detection algorithms for rapidminer," in *Proc. of the 3rd RapidMiner Community Meeting and Conference (RCOMM 2012)*, 2012, pp. 1–12.

[95] A. Rodriguez and A. Laio, "Clustering by fast search and find of density peaks," *Science*, vol. 344, no. 6191, pp. 1492–1496, 2014.

[96] B. G. Amidan, T. A. Ferryman, and S. K. Cooley, "Data outlier detection using the chebyshev theorem," in *2005 IEEE Aerospace Conference*. IEEE, 2005, pp. 3814–3819.

[97] D. Achlioptas, "Database-friendly random projections," in *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2001, pp. 274–281.

[98] T. De Vries, S. Chawla, and M. E. Houle, "Finding local anomalies in very high dimensional space," in *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 2010, pp. 128–137.

[99] E. Schubert, A. Zimek, and H.-P. Kriegel, "Fast and scalable outlier detection with approximate nearest neighbor ensembles," in *International Conference on Database Systems for Advanced Applications*. Springer, 2015, pp. 19–36.

[100] G. M. Morton, "A computer oriented geodetic data base and a new technique in file sequencing," *IBM Germany Scientific Symposium Series*, 1966.

[101] S. Hariri, M. Carrasco Kind, and R. J. Brunner, "Extended Isolation Forest," *ArXiv e-prints*, Nov. 2018.

[102] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU Press, 2012, vol. 3.

[103] A. Guttman, *R-trees: A dynamic index structure for spatial searching*. ACM, 1984, vol. 14.

[104] K.-I. Lin, H. V. Jagadish, and C. Faloutsos, "The tv-tree: An index structure for high-dimensional data," *The VLDB Journal*, vol. 3, no. 4, pp. 517–542, 1994.

[105] N. H. Vu and V. Gopalkrishnan, "Efficient pruning schemes for distance-based outlier detection," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2009, pp. 160–175.

[106] J. A. Orenstein and T. H. Merrett, "A class of data structures for associative searching," in *Proceedings of the 3rd ACM SIGACT-SIGMOD symposium on Principles of database systems*. ACM, 1984, pp. 181–190.

[107] T. Li, Y. Lin, and H. Shen, "A locality-aware similar information searching scheme," *International Journal on Digital Libraries*, vol. 17, no. 2, pp. 79–93, 2016.

[108] S. Deegalla and H. Bostrom, "Reducing high-dimensional data by principal component analysis vs. random projection for nearest neighbor classification," in *null*. IEEE, 2006, pp. 245–250.

[109] W. B. Johnson and J. Lindenstrauss, "Extensions of lipschitz mappings into a hilbert space," *Contemporary mathematics*, vol. 26, no. 189-206, p. 1, 1984.

[110] A. Zimek, E. Schubert, and H.-P. Kriegel, "A survey on unsupervised outlier detection in high-dimensional numerical data," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 5, no. 5, pp. 363–387, 2012.

[111] G. Kollios, D. Gunopulos, N. Koudas, and S. Berchtold, "Efficient biased sampling for approximate clustering and outlier detection in large data sets," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 5, pp. 1170–1187, 2003.

[112] M. Wu and C. Jermaine, "Outlier detection by sampling with accuracy guarantees," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 767–772.

[113] S. D. Bay and M. Schwabacher, "Mining distance-based outliers in near linear time with randomization and a simple pruning rule," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2003, pp. 29–38.

[114] W. Jin, A. K. Tung, and J. Han, "Mining top-n local outliers in large databases," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2001, pp. 293–298.

[115] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is "nearest neighbor" meaningful?" in *International conference on database theory*. Springer, 1999, pp. 217–235.

[116] A. Hinneburg, C. C. Aggarwal, and D. A. Keim, "What is the nearest neighbor in high dimensional spaces?" in *26th Internat. Conference on Very Large Databases*, 2000, pp. 506–515.

[117] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, "On the surprising behavior of distance metrics in high dimensional space," in *International conference on database theory*. Springer, 2001, pp. 420–434.

[118] A. Ghoting, S. Parthasarathy, and M. E. Otey, "Fast mining of distance-based outliers in high-dimensional datasets," *Data Mining and Knowledge Discovery*, vol. 16, no. 3, pp. 349–364, 2008.

[119] H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek, "Outlier detection in axis-parallel subspaces of high dimensional data," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2009, pp. 831–838.

[120] F. Keller, E. Muller, and K. Bohm, "Hics: high contrast subspaces for density-based outlier ranking," in *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*. IEEE, 2012, pp. 1037–1048.

[121] S. Sathe and C. C. Aggarwal, "Subspace outlier detection in linear time with randomized hashing," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 459–468.

[122] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215, 1994, pp. 487–499.

[123] J. Zhang and H. Wang, "Detecting outlying subspaces for high-dimensional data: the new task, algorithms, and performance," *Knowledge and information systems*, vol. 10, no. 3, pp. 333–355, 2006.

[124] S. Sathe and C. Aggarwal, "Lodes: Local density meets spectral outlier detection," in *Proceedings of the 2016 SIAM International Conference on Data Mining*. SIAM, 2016, pp. 171–179.

[125] G. Pang, L. Cao, L. Chen, and H. Liu, "Learning representations of ultrahigh-dimensional data for random distance-based outlier detection," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 2041–2050.

[126] M. Salehi and L. Rashidi, "A survey on anomaly detection in evolving data:[with application to forest fire risk prediction]," *ACM SIGKDD Explorations Newsletter*, vol. 20, no. 1, pp. 13–23, 2018.

[127] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *Proceedings of the 2007 SIAM international conference on data mining*. SIAM, 2007, pp. 443–448.

[128] F. Angiulli and F. Fassetti, "Detecting distance-based outliers in streams of data," in *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*. ACM, 2007, pp. 811–820.

[129] D. Yang, E. A. Rundensteiner, and M. O. Ward, "Neighbor-based pattern detection for windows over streaming data," in *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*. ACM, 2009, pp. 529–540.

[130] M. Kontaki, A. Gounaris, A. N. Papadopoulos, K. Tsichlas, and Y. Manolopoulos, "Continuous monitoring of distance-based outliers over data streams," in *2011 IEEE 27th International Conference on Data Engineering*. IEEE, 2011, pp. 135–146.

[131] L. Cao, D. Yang, Q. Wang, Y. Yu, J. Wang, and E. A. Rundensteiner, "Scalable distance-based outlier detection over high-volume data streams," in *2014 IEEE 30th International Conference on Data Engineering*. IEEE, 2014, pp. 76–87.

[132] D. Pokrajac, A. Lazarevic, and L. J. Latecki, "Incremental local outlier detection for data streams," in *2007 IEEE symposium on computational intelligence and data mining*. IEEE, 2007, pp. 504–515.

[133] M. Salehi, C. Leckie, J. C. Bezdek, T. Vaithianathan, and X. Zhang, "Fast memory efficient local outlier detection in data streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 12, pp. 3246–3260, 2016.

[134] G. S. Na, D. Kim, and H. Yu, "Dilof: Effective and memory efficient local outlier detection in data streams," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 1993–2002.

[135] B. Póczos, L. Xiong, and J. Schneider, "Nonparametric divergence estimation with applications to machine learning on distributions," *arXiv preprint arXiv:1202.3758*, 2012.

[136] Y. Chen and L. Tu, "Density-based clustering for real-time stream data," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2007, pp. 133–142.

[137] M. Elahi, K. Li, W. Nisar, X. Lv, and H. Wang, "Efficient clustering-based outlier detection algorithm for dynamic data stream," in *2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery*, vol. 5. IEEE, 2008, pp. 298–304.

[138] I. Assent, P. Kranen, C. Baldauf, and T. Seidl, "Anyout: Anytime outlier detection on streaming data," in *International Conference on Database Systems for Advanced Applications*. Springer, 2012, pp. 228–242.

[139] P. Kranen, I. Assent, C. Baldauf, and T. Seidl, "Self-adaptive anytime stream clustering," in *2009 Ninth IEEE International Conference on Data Mining*. IEEE, 2009, pp. 249–258.

[140] M. Salehi, C. A. Leckie, M. Moshtaghi, and T. Vaithianathan, "A relevance weighted ensemble model for anomaly detection in switching data streams," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining 2014*, 2014, pp. 461–473.

[141] M. Chenaghlou, M. Moshtaghi, C. Leckie, and M. Salehi, "An efficient method for anomaly detection in non-stationary data streams," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–6.

[142] P. Kranen and T. Seidl, "Harnessing the strengths of anytime algorithms for constant data streams," *Data Mining and Knowledge Discovery*, vol. 19, no. 2, pp. 245–260, 2009.

[143] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[144] M. Moshtaghi, S. Rajasegarar, C. Leckie, and S. Karunasekera, "An efficient hyperellipsoidal clustering algorithm for resource-constrained environments," *Pattern Recognition*, vol. 44, no. 9, pp. 2197–2209, 2011.

[145] M. Moshtaghi, T. C. Havens, J. C. Bezdek, L. A. F. Park, C. Leckie, S. Rajasegarar, J. M. Keller, and M. Palaniswami, "Clustering ellipses for anomaly detection," *Pattern Recognition*, vol. 44, no. 1, pp. 55–69, 2011.

[146] R. A. Johnson, D. W. Wichern *et al.*, *Applied multivariate statistical analysis*. Prentice hall Upper Saddle River, NJ, 2002, vol. 5.

[147] D. W. Henderson and E. Moura, *Experiencing geometry on plane and sphere*. Cornell University, Dept. of Mathematics, 1994.

[148] A. Boukerche, S. Hong, and T. Jacob, "A distributed algorithm for dynamic channel allocation," *Mobile Networks and Applications*, vol. 7, no. 2, pp. 115–126, 2002.

[149] A. Boukerche and C. Tropper, "A distributed graph algorithm for the detection of local cycles and knots," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 8, pp. 748–757, 1998.

[150] A. Boukerche, S. K. Das, and A. Fabbri, "Swimnet: a scalable parallel simulation testbed for wireless and mobile networks," *Wireless Networks*, vol. 7, no. 5, pp. 467–486, 2001.

[151] A. Boukerche and C. Dzermajko, "Performance evaluation of data distribution management strategies," *Concurrency and Computation: Practice and Experience*, vol. 16, no. 15, pp. 1545–1573, 2004.

[152] A. Boukerche, N. J. McGraw, C. Dzermajko, and K. Lu, "Grid-filtered region-based data distribution management in large-scale distributed simulation systems," in *38th Annual Simulation Symposium*. IEEE, 2005, pp. 259–266.

[153] E. E. Ajaltouni, A. Boukerche, and M. Zhang, "An efficient dynamic load balancing scheme for distributed simulations on a grid infrastructure," in *Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*. IEEE Computer Society, 2008, pp. 61–68.

[154] R. E. De Grande and A. Boukerche, "Dynamic balancing of communication and computation load for hla-based simulations on large-scale distributed systems," *Journal of Parallel and Distributed Computing*, vol. 71, no. 1, pp. 40–52, 2011.

[155] A. Boukerche and S. Rogers, "Gps query optimization in mobile and wireless networks," in *Proceedings. Sixth IEEE Symposium on Computers and Communications*. IEEE, 2001, pp. 198–203.

[156] A. Boukerche, K. El-Khatib, L. Xu, and L. Korba, "A novel solution for achieving anonymity in wireless ad hoc networks," in *Proceedings of the 1st ACM international workshop on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*. ACM, 2004, pp. 30–38.

[157] ——, "An efficient secure distributed anonymous routing protocol for mobile and wireless ad hoc networks," *computer communications*, vol. 28, no. 10, pp. 1193–1203, 2005.

[158] M. Elhadef, A. Boukerche, and H. Elkadiki, "Performance analysis of a distributed comparison-based self-diagnosis protocol for wireless ad-hoc networks," in *Proceedings of the 9th ACM international symposium on Modeling analysis and simulation of wireless and mobile systems*. ACM, 2006, pp. 165–172.

[159] ——, "Diagnosing mobile ad-hoc networks: two distributed comparison-based self-diagnosis protocols," in *Proceedings of the 4th ACM international workshop on Mobility management and wireless access*. ACM, 2006, pp. 18–27.

[160] A. Boukerche, X. Fei, and R. B. Araujo, "An optimal coverage-preserving scheme for wireless sensor networks based on local information exchange," *Computer Communications*, vol. 30, no. 14-15, pp. 2708–2720, 2007.

[161] A. Boukerche and K. Abrougui, "An efficient leader election protocol for wireless quasi-static mesh networks: Proof of correctness," in *2007 IEEE International Conference on Communications*. IEEE, 2007, pp. 3491–3496.

[162] A. Boukerche and Y. Ren, "A security management scheme using a novel computational reputation model for wireless and mobile ad hoc networks," in *Proceedings of the 5th ACM symposium on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*. ACM, 2008, pp. 88–95.

[163] A. Bamis, A. Boukerche, I. Chatzigiannakis, and S. Nikoletseas, "A mobility aware protocol synthesis for efficient routing in ad hoc mobile networks," *Computer Networks*, vol. 52, no. 1, pp. 130–154, 2008.

[164] M. Elhadef, A. Boukerche, and H. Elkadiki, "A distributed fault identification protocol for wireless and mobile ad hoc networks," *Journal of parallel and distributed computing*, vol. 68, no. 3, pp. 321–335, 2008.

[165] A. Boukerche, A. Zarrad, and R. Araujo, "A cross-layer approach-based gnutella for collaborative virtual environments over mobile ad hoc networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 7, pp. 911–924, 2009.

[166] Z. Zhang, R. W. Pazzi, and A. Boukerche, "A mobility management scheme for wireless mesh networks based on a hybrid routing protocol," *Computer Networks*, vol. 54, no. 4, pp. 558–572, 2010.

[167] A. Boukerche, C. Rezende, and R. W. Pazzi, "Improving neighbor localization in vehicular ad hoc networks to avoid overhead from periodic messages," in *GLOBECOM 2009-2009 IEEE Global Telecommunications Conference*. IEEE, 2009, pp. 1–6.

[168] K. Abrougui, A. Boukerche, and R. W. N. Pazzi, "Design and evaluation of context-aware and location-based service discovery protocols for vehicular networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 3, pp. 717–735, 2011.

[169] C. Rezende, A. Mammeri, A. Boukerche, and A. A. Loureiro, "A receiver-based video dissemination solution for vehicular networks with content transmissions decoupled from relay node selection," *Ad Hoc Networks*, vol. 17, pp. 1–17, 2014.

[170] C. Rezende, A. Boukerche, H. S. Ramos, and A. A. Loureiro, "A reactive and scalable unicast solution for video streaming over vanets," *IEEE Transactions on Computers*, vol. 64, no. 3, pp. 614–626, 2014.

[171] F. A. Silva, A. Boukerche, T. R. Silva, L. B. Ruiz, E. Cerqueira, and A. A. Loureiro, "Vehicular networks: A new challenge for content-delivery-based applications," *ACM Computing Surveys (CSUR)*, vol. 49, no. 1, p. 11, 2016.

[172] R. Oliveira, C. Montez, A. Boukerche, and M. S. Wangham, "Reliable data dissemination protocol for vanet traffic safety applications," *Ad Hoc Networks*, vol. 63, pp. 30–44, 2017.

[173] A. Boukerche, S. Hong, and T. Jacob, "An efficient synchronization scheme of multimedia streams in wireless and mobile systems," *IEEE transactions on Parallel and Distributed Systems*, vol. 13, no. 9, pp. 911–923, 2002.

[174] A. Boukerche, R. W. Pazzi, and J. Feng, "An end-to-end virtual environment streaming technique for thin mobile devices over heterogeneous networks," *Computer Communications*, vol. 31, no. 11, pp. 2716–2725, 2008.

[175] W. Lu, Y. Shen, S. Chen, and B. C. Ooi, "Efficient processing of k nearest neighbor joins using mapreduce," *Proceedings of the VLDB Endowment*, vol. 5, no. 10, pp. 1016–1027, 2012.

[176] 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*. ACM, 2012, pp. 38–49.

[177] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 36, no. 11, pp. 2227–2240, 2014.

[178] G. Chatzimilioudis, C. Costa, D. Zeinalipour-Yazti, W.-C. Lee, and E. Pitoura, "Distributed in-memory processing of all k nearest neighbor queries," *IEEE transactions on knowledge and data engineering*, vol. 28, no. 4, pp. 925–938, 2016.

[179] C. Kuhlman, Y. Yan, L. Cao, and E. Rundensteiner, "Pivot-based distributed k-nearest neighbor mining," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2017, pp. 843–860.

[180] K. Bhaduri, B. L. Matthews, and C. R. Giannella, "Algorithms for speeding up distance-based outlier detection," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2011, pp. 859–867.

[181] F. Angiulli, S. Basta, S. Lodi, and C. Sartori, "Distributed strategies for mining outliers in large data sets," *IEEE transactions on knowledge and data engineering*, vol. 25, no. 7, pp. 1520–1532, 2013.

[182] F. Angiulli, S. Basta, and C. Pizzuti, "Distance-based detection and prediction of outliers," *IEEE transactions on knowledge and data engineering*, vol. 18, no. 2, pp. 145–160, 2006.

[183] Y. Yan, L. Cao, and E. A. Rundensteiner, "Distributed top-n local outlier detection in big data," in *Big Data (Big Data), 2017 IEEE International Conference on*. IEEE, 2017, pp. 827–836.

[184] C. Désir, S. Bernard, C. Petitjean, and L. Heutte, "One class random forests," *Pattern Recognition*, vol. 46, no. 12, pp. 3490–3506, 2013.

[185] M. Bai, X. Wang, J. Xin, and G. Wang, "An efficient algorithm for distributed density-based outlier detection on big data," *Neurocomputing*, vol. 181, pp. 19–28, 2016.

[186] Y. Yan, L. Cao, and E. A. Rundensteiner, "Scalable top-n local outlier detection," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 1235–1244.

[187] L. Mainetti, L. Patrono, and A. Vilei, "Evolution of wireless sensor networks towards the internet of things: A survey," in *SoftCOM 2011, 19th international conference on software, telecommunications and computer networks*. IEEE, 2011, pp. 1–6.

[188] A. Boukerche, Y. Du, J. Feng, and R. Pazzi, "A reliable synchronous transport protocol for wireless image sensor networks," in *2008 IEEE Symposium on Computers and Communications*. IEEE, 2008, pp. 1083–1089.

[189] S. Samarah, M. Al-Hajri, and A. Boukerche, "A predictive energy-efficient technique to support object-tracking sensor networks," *IEEE Transactions on Vehicular Technology*, vol. 60, no. 2, pp. 656–663, 2010.

[190] R. W. Coutinho, A. Boukerche, L. F. Vieira, and A. A. Loureiro, "Gedar: geographic and opportunistic routing protocol with depth adjustment for mobile underwater sensor networks," in *2014 IEEE International Conference on communications (ICC)*. IEEE, 2014, pp. 251–256.

[191] A. Darehshoorzadeh and A. Boukerche, "Underwater sensor networks: A new challenge for opportunistic routing protocols," *IEEE Communications Magazine*, vol. 53, no. 11, pp. 98–107, 2015.

[192] R. W. Coutinho, A. Boukerche, L. F. Vieira, and A. A. Loureiro, "A novel void node recovery paradigm for long-term underwater sensor networks," *Ad Hoc Networks*, vol. 34, pp. 144–156, 2015.

[193] ——, "Design guidelines for opportunistic routing in underwater networks," *IEEE Communications Magazine*, vol. 54, no. 2, pp. 40–48, 2016.

[194] O. O'Malley, "Hadoop terasort package description," [Online]. Available: https://https://hadoop.apache.org/docs/r2.7.1/api/org/apache/hadoop/examples/terasort/package-summary.html, 2015, accessed on: February, 2019.

[195] J. L. Bentley, "Multidimensional divide-and-conquer," *Communications of the ACM*, vol. 23, no. 4, pp. 214–229, 1980.

[196] Dheeru, Dua and Karra Taniskidou, Efi, "UCI Machine Learning Repository," http://archive.ics.uci.edu/ml, 2017.

[197] J. Y. Lee, U. Kang, D. Koutra, and C. Faloutsos, "Fast anomaly detection despite the duplicates," in *Proceedings of the 22nd International Conference on World Wide Web*. ACM, 2013, pp. 195–196.

[198] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones." in *NSDI*, vol. 13, 2013, pp. 185–198.

[199] A. Andoni, "E2lsh 0.1 user manual," *http://www. mit. edu/andoni/LSH/*, 2005.

[200] G. Seni and J. F. Elder, "Ensemble methods in data mining: improving accuracy through combining predictions," *Synthesis lectures on data mining and knowledge discovery*, vol. 2, no. 1, pp. 1–126, 2010.

[201] L. Rokach, *Pattern classification using ensemble methods*. World Scientific, 2010, vol. 75.

[202] C. C. Aggarwal and S. Sathe, *Outlier ensembles: An introduction*. Springer, 2017.

[203] M. Sugiyama and K. Borgwardt, "Rapid distance-based outlier detection via sampling," in *Advances in Neural Information Processing Systems*, 2013, pp. 467–475.

[204] E. J. Candès, X. Li, Y. Ma, and J. Wright, "Robust principal component analysis?" *Journal of the ACM (JACM)*, vol. 58, no. 3, p. 11, 2011.

[205] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[206] A. Makhzani and B. J. Frey, "Winner-take-all autoencoders," in *Advances in neural information processing systems*, 2015, pp. 2791–2799.

[207] D. L. Donoho and C. Grimes, "Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data," *Proceedings of the National Academy of Sciences*, vol. 100, no. 10, pp. 5591–5596, 2003.

[208] P. Li, T. J. Hastie, and K. W. Church, "Very sparse random projections," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 287–296.