Hindawi
Scientific Programming
Volume 2020, Article ID 4365356, 17 pages
https://doi.org/10.1155/2020/4365356

Research Article

A Log-Based Anomaly Detection Method with Efficient Neighbor Searching and Automatic *K* Neighbor Selection

Bingming Wang, Shi Ying, and Zhe Yang

School of Computer Science, Wuhan University, Wuhan, China

Correspondence should be addressed to Shi Ying; yingshi@whu.edu.cn

Received 2 January 2020; Accepted 27 April 2020; Published 2 June 2020

Academic Editor: Antonio J. Peña

Using the *k*-nearest neighbor (kNN) algorithm in the supervised learning method to detect anomalies can get more accurate results. However, when using kNN algorithm to detect anomaly, it is inefficient at finding *k* neighbors from large-scale log data; at the same time, log data are imbalanced in quantity, so it is a challenge to select proper *k* neighbors for different data distributions. In this paper, we propose a log-based anomaly detection method with efficient selection of neighbors and automatic selection of *k* neighbors. First, we propose a neighbor search method based on minhash and MVP-tree. The minhash algorithm is used to group similar logs into the same bucket, and MVP-tree model is built for samples in each bucket. In this way, we can reduce the effort of distance calculation and the number of neighbor samples that need to be compared, so as to improve the efficiency of finding neighbors. In the process of selecting *k* neighbors, we propose an automatic method based on the Silhouette Coefficient, which can select proper *k* neighbors to improve the accuracy of anomaly detection. Our method is verified on six different types of log data to prove its universality and feasibility.

1. Introduction

Modern systems are developing to large scale, either by scaling out to complex systems built on thousands of commodity machines (e.g., Spark) or by scaling up to supercomputers with thousands of processors (e.g., Blue Gene/L). These systems are becoming the core part of IT industry; the occurrence of failure and its influence on system performance and operation cost have become a very important concern in the research field. Complex software and systems often include more bugs and are difficult to understand and analyze. Besides, as time goes on, quality of these systems is aging. These problems will cause the collapse of the software or system downtime. For large-scale computer systems, such as supercomputers, unexpected downtime will cause much cost, so operators should find and fix the causes of downtime.

Logs are typically used by developers or operators to ensure the reliability of the software or system, and the software or system can provide logs with the status of its components and the various events that occur at runtime. These logs contain valuable information to support anomaly detection activities, and they are collected at different levels of detail. Analyzing and interpreting a large amount of log data that does not always conform to a standardized structure constitute a daunting task. As the scale increases, distributed systems can generate logs as a collection of huge volume of messages from several components [1]. For example, the supercomputer Spirit with thousands of processors can produce 1TB log data per day. The size and diversity of such logs can be much more in other application domains such as the Internet of Things [2]. If a problem occurs, it is very time-consuming for operators to find system problems through manually examining a great amount of log messages. Therefore, it is not feasible to effectively detect anomalies by applying manual or traditional analysis techniques on such large-scale log data.

A large number of anomaly detection methods based on machine learning technologies have been studied in [3]. Experiments show that supervised learning methods are generally superior to unsupervised learning methods in terms of the three evaluation indicators of accuracy, recall,

and F measure [4]. Therefore, supervised learning method is a good choice. All too often, log lines record the behavior of the system, including normal system behavior and abnormal system behavior. There are some differences between logs that record normal behavior (normal logs) and the ones that record abnormal behavior (abnormal logs). Also, the number of abnormal logs is usually much less than the number of normal logs, which we will discuss in Section 2. Based on the above characteristics, we can treat the abnormal logs as outliers. Thus, using outlier detection methods in machine learning algorithm is an effective way to detect anomalies from large-scale log data. In most of the outlier detection methods, k-nearest neighbor (kNN) algorithm is a supervised learning method which can achieve higher accuracy. Therefore, we detect anomalies with a kNN-based method, which computes distances between logs and gets the small portion of logs that are far away from the majorities, that is, outliers.

However, when kNN algorithm is applied to anomaly detection on large-scale heterogeneous log data, some characteristics of logs will affect the kNN efficiency, of which the following two points are prominent:

(1) Log data has the characteristics of large scale. kNN algorithm needs to calculate distances between the sample to be detected and samples in the training set to obtain neighbor samples, and then the effort of calculating distances will be large, resulting in low anomaly detection efficiency.

(2) Log data has the characteristics of quantity imbalance. The proper k neighbors corresponding to different data distributions\ are different, so it is not appropriates to use a fixed k value for neighbor selection on all log data. We will discuss the details in Section 2.2.

According to the above problems, we propose a log-based anomaly detection method with efficient neighbor selection and automatic k neighbor selection. The main contributions of this paper are as follows:

(1) Aimed at the large scale of log data, a neighbor search method based on minhash and MVP-tree is proposed, which reduces the effort of calculating distances, reduces the number of neighbor samples that need to be compared, and improves the neighbor search efficiency of anomaly detection based on kNN algorithm

(2) Aimed at the quantity imbalance of log data, an automatic k neighbor selection method based on Silhouette Coefficient is proposed, which selects appropriate k neighbors for data with different distributions, thereby improving the accuracy of anomaly detection

(3) Aiming to verify the universality and feasibility of our method, we set up experiments on six log datasets generated by different types of systems

In the rest of this paper, Section 2 describes the background and motivation of our method. The detail of the proposed method is elaborated in Section 3. We evaluate our method and report the results in Section 4. The advantages and disadvantages of this paper are discussed in Section 5. Section 6 reviews the related work. Finally, conclusions and future work are provided in Section 7.

2. Background and Motivation

2.1. Why Can Minhash-Based Method Improve the Efficiency of kNN-Based Anomaly Detection?

2.1.1. Reduction of Effort for Distance Calculation. In kNN-based anomaly detection, we need to calculate distances between the sample to be detected and the training set samples; then these distances are sorted, and the nearest k samples are selected as k neighbors. Due to the large size of the log data, the step of calculating and sorting distances will take much time. Minhash algorithm can group similar logs into the same bucket through hash functions. When searching for neighbors, we only need to calculate distances between the sample to be detected and the samples in the same bucket, thereby greatly reducing the number of samples that need to be calculated. Thus, minhash-based method can reduce the effort of calculating distances.

2.1.2. Reduction of Dimension of Log Vectors. A commonly used method for converting log data into vector is Bag-of-Words (BoW) model. Words in the data are stored into word bag without repetition. The dimension of the vector is equal to the size of word bag, and the number at position i in the vector indicates the frequency of word bag's ith word in this log line. However, because of the large scale of word types in log data, vector dimension will be too high, which will occupy a very large space, making the storage of logs a problem, let alone calculating the similarity between logs. Table 1 shows the number of word types corresponding to different sizes of data in the six datasets. Data shown in the table is the remaining word types after we filtered out unwanted words (such as time stamp, log number, and other parameters). Obviously, if a log line is converted into a vector according to these word types, dimension of the vector will be very high, which will seriously affect the efficiency of anomaly detection. For such a data representation, it is necessary to reduce the dimension, and minhash algorithm can exactly do this.

Minhash algorithm uses the Jaccard similarity to calculate the similarity between logs. This method converts each log line into a vector, in which the ith element represents the feature we extracted from ith frequency matrix. We use different hash functions to get different word orders; each word order corresponds to a word frequency matrix. Thus, the vector dimension is equal to the number of frequency matrixes. We can change the word order to get many frequency matrixes, but it must be smaller than the vector dimension when using the BoW model (in our experiment, 30 frequency matrixes were generated). The vector dimension when using the BoW model is the number of word types in logs.

Table 1: Word types corresponding to data of different sizes in different datasets.

Log sets	Word types corresponding to different sizes of data		
	1M	100M	1G
Liberty	9112	436777	2368053
BGL	10904	845629	4592728
Thunderbird	13444	264629	1165458
Spirit	8631	454437	3055695
HDFS	4538	211080	2459515
Zookeeper	5590	53094	53094

2.2. Why Is It Necessary to Select Proper k Neighbors Automatically? In kNN algorithm, selecting a proper k is very important. As shown in Figure 1, there are three categories of samples: A, B, and x. The x in the middle is the sample to be detected, and the other samples are neighbors of x. When we set $k = 11$, we select 11 nearest neighbors for sample x, in which 4 samples are labeled as A and 7 samples labeled as B. At this time, sample x is classified as B. When we set $k = 19$, then 19 nearest neighbor samples of x contain 10 samples A and 9 samples B. Therefore, x is classified as A.

In the log data, there are differences between the number of normal logs and the number of abnormal logs, but this difference is also different in different datasets. It is not appropriate to use a uniform fixed k value in all log datasets. Therefore, it is necessary to automatically select appropriate k neighbors for data samples with different distributions.

2.3. Why Can Silhouette Coefficient Be Used to Select k Neighbors? Silhouette Coefficient, which was first proposed by Rousseeuw [5], is an evaluation method for the effectiveness of clustering. It combines cohesion and separation, where cohesion indicates the average distance between sample i and samples in the same cluster, and separation indicates the average distance from sample i to all samples in the nearest cluster. Then the Silhouette Coefficient defines a calculation method, and its value is in the range [−1, 1] (more details are provided in Section 3.2). If the Silhouette Coefficient of i is close to 1, this means that the cohesion and separation of sample i are better; that is, sample i is similar to the samples in the same cluster, and the classification is correct.

The idea behind the nearest neighbor selection in kNN algorithm is similar to Silhouette Coefficient. If more samples have the same type as the sample to be detected in its k neighbors, it is more likely to be correctly classified. Therefore, we propose a k neighbor selection method based on the Silhouette Coefficient, so that the selected k neighbors belong to the same category as much as possible, thereby improving the accuracy of anomaly detection. More details are described in Section 3.2.

3. Our Method

This paper proposes a log-based anomaly detection method with efficient neighbor selection and automatic k neighbor selection, which mainly includes three parts. The overall framework is shown in Figure 2. (1) The first part is neighbor searching based on minhash and MVP-tree. We use the minhash algorithm to group similar logs into a bucket and then build an MVP-tree for samples in each bucket. (2) The second part is automatic selection for k neighbors. We select neighbors from the MVP-tree and store them into the spare neighbor sample set. Based on the Silhouette Coefficient, we define a neighbor evaluation method, which is used to judge whether the sample in the spare neighbor sample set is helpful to improve the accuracy of classification. If the sample can meet the condition, we store it into the actual neighbor sample set. The final actual neighbor sample set is the proper k neighbors for anomaly detection. (3) The third part is anomaly detection. Finally, to detect anomalies, we calculate the average distance between samples of each category in the actual neighbor sample set and the sample to be detected.

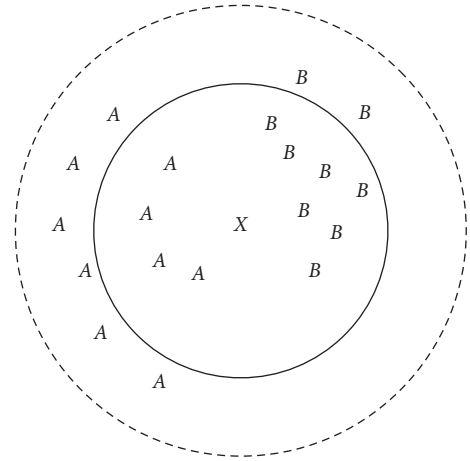

Figure 1: The influence of different k values on classification results.

3.1. Neighbor Searching Based on Minhash and MVP-Tree. Minhash [5, 6] is a scheme devised by Broder for efficiently estimating the similarity of two sets of items. Specifically, it estimates the Jaccard similarity of two sets, which is one of the most common and effective similarity metrics. Therefore, the nearest neighbor search method based on minhash and MVP-tree mainly includes three steps: (1) using minhash algorithm to convert log data into vectors; (2) using Jaccard similarity measure to calculate the similarity between logs and grouping samples with high similarity into a bucket; (3) constructing MVP-tree model for each bucket of log data.

3.1.1. Vectorization of Log Data. We convert logs into a form suitable for Jaccard similarity calculation, that is, the word frequency matrix. Logs are parsed into words according to the space. As shown in Figure 3, the four log lines are parsed as: "QuorumPeer[myid=1]/0:0:0:0:0:0:0:0:2181:Follower@118, -, Got, zxid, 0x100000001, expected, 0x1, 0x100001547, 0x, NIOServerCxn.Factory:0.0.0. 0/0.0.0.0:2181:

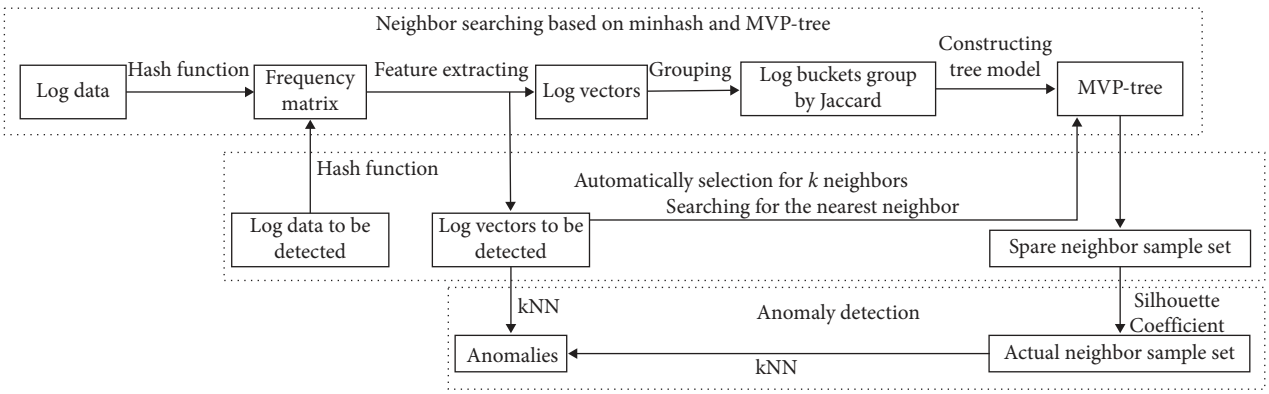

Figure 2: The influence of different k values on classification results.

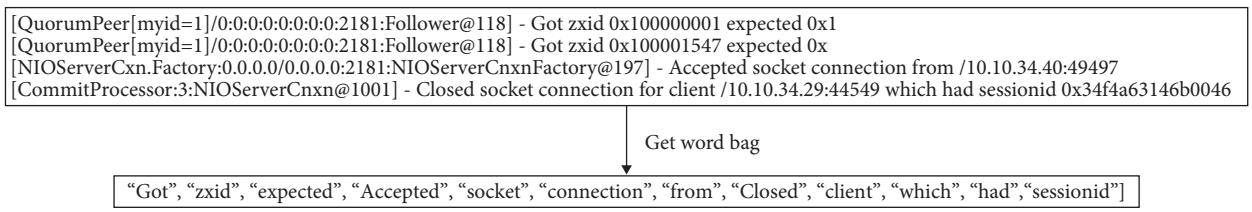

Figure 3: Example of log parsing.

NIOServerCnxnFactory@197, Accepted, socket, connection, from, /10.10.34. 40:49497, CommitProcessor:3:NIO-ServerCnxn@1001, Closed, for, client, /10.10.34. 29:44549, which, had, sessionid, 0x34f4a63146b0046." Among these words, the ones contain numbers have a high probability of indicating variables, such as "0x100000001 and 0x34f4a63146 b0046." Some variables are different in each log line. We delete these words that our method does not need, and the filtered words are stored in the word bag; they are ["Got," "zxid," "expected," "Accepted," "socket," "connection," "from," "Closed," "client," "which," "had," "sessioned"].

Next, we count the frequency of each word in the word bag appearing in each log line. As shown in Figure 4, we convert the log line into a vector [0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1], which indicates that, for the log line in Figure 4, the frequency of each word in the word bag is 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1.

According to the above method, we convert each log line into word frequency vectors. All log vectors form a sparse $M * N$ word frequency matrix. M represents the number of all words in the word bag. N represents the number of log lines. Element in the matrix represents the frequency of mth ($m \in [0, M]$) word in the nth log line ($n \in [0, N]$).

In the obtained word frequency matrix, we extract the position corresponding to the first nonzero frequency word in each column, and this position is the feature for the log line corresponding to this column. That being said, for each log line, we can extract a feature from a frequency matrix. When we change the word order in the matrix, we will get a new matrix, from which each log data can get a new feature. We choose Hn hash functions to adjust the word orders, such as $h((3x + 1)\text{mod}5)$, where x represents the original position of the word. Each hash function is used 10 times; then we will get $10 * Hn$ kinds of word orders, corresponding to $10 * Hn$ kinds of word frequency matrixes, and each log line can get $10 * Hn$ features. These features form a log vector, so the vector dimension of the log line is also $10 * Hn$.

As shown in Figure 5, we use hash function to disrupt word orders in the word bag four times and get four word orders corresponding to (a)–(d), respectively. According to the way we obtain features, Log 1 can be converted into a vector [0, 3, 2, 2], because, in these four frequency matrixes, the first nonzero word corresponding to Log 1 corresponds to positions 0, 3, 2, and 2, respectively. Similarly, vectors of Log 2, Log 3, and Log 4 are [0, 0, 0, 0], [0, 2, 0, 1], and [4, 0, 1, 1].

3.1.2. Similarity Calculation and Grouping. We use the Jaccard measure to calculate the similarity between logs, as shown in (1). In the original formula of Jaccard similarity, $|s \cap p|$ and $|s \cup p|$ represent the intersection and union of sets s and p. When we use it for log vector similarity, we define $|s \cap p|$ as the number of same elements at the same position in vectors of log$s$ and log$p$, and $|s \cup p|$ represents the vector dimensions of log s and log p. $J(s, p)$ represents the Jaccard similarity of log s and log p. If there are more same elements at the same position of s and p, there are more same features extracted from matrixes with multiple transformations; thus, the two log lines are more similar. We set a similarity threshold and group logs whose Jaccard values greater than this threshold into the same bucket:

$$J(s, p) = \frac{|s \cap p|}{|s \cap p|}. \quad (1)$$

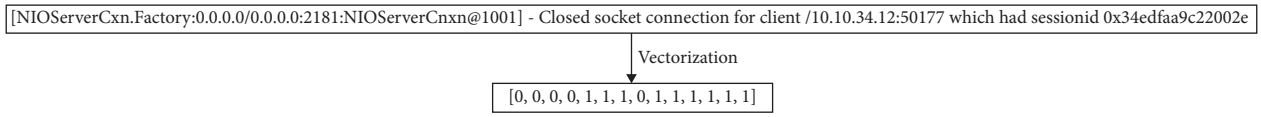

Figure 4: Example of log vectorization.

NUM	WORDS	Log1	Log2	Log3	Log4
0	WARN	2	2	2	0
1	Got	1	0	1	0
2	zxid	1	0	1	0
3	expected	1	0	2	0
4	INFO	1	0	1	1
5	Closed	0	1	1	1
6	socket	0	1	1	2
7	connection	0	1	0	1
8	for	0	1	0	1
9	client	0	1	0	0
10	which	0	1	0	0
11	had	0	1	1	0
12	sessionid	0	1	1	0

(a)

NUM	WORDS	Log1	Log2	Log3	Log4
0	connection	0	1	0	1
1	which	0	1	0	0
2	sessionid	0	1	1	0
3	INFO	1	0	1	1
4	Got	1	0	1	0
5	WARN	2	2	2	0
6	zxid	1	0	1	0
7	had	0	1	1	0
8	client	0	1	0	0
9	expected	1	0	2	0
10	Closed	0	1	1	1
11	for	0	1	0	1
12	socket	0	1	1	2

(b)

NUM	WORDS	Log1	Log2	Log3	Log4
0	had	0	1	1	0
1	Closed	0	1	1	1
2	INFO	1	0	1	1
3	which	0	1	0	0
4	Got	1	0	1	0
5	connection	0	1	0	1
6	expected	1	0	2	0
7	client	0	1	0	0
8	socket	0	1	1	2
9	sessionid	0	1	1	0
10	for	0	1	0	1
11	WARN	2	2	2	0
12	zxid	1	0	1	0

(c)

NUM	WORDS	Log1	Log2	Log3	Log4
0	client	0	1	0	0
1	socket	0	1	1	2
2	WARN	2	2	2	0
3	zxid	1	0	1	0
4	connecion	0	1	0	1
5	expected	1	0	2	0
6	had	0	1	1	0
7	Closed	0	1	1	1
8	sessionid	0	1	1	0
9	for	0	1	0	1
10	INFO	1	0	1	1
11	which	0	1	0	0
12	Got	1	0	1	0

(d)

Figure 5: Example of four different word orders.

3.1.3. Construction of MVP-Tree Model. Due to the large scale of the log data, the number of samples mapped into the same bucket is still large. Aiming at this characteristic of log data, we use the multi-Vantage Point tree (MVP-tree), which is an improvement of Vantage Point tree (VP-tree), to build a tree structure model for each bucket.

VP-tree is an index structure in metric space based on distance. It is a static binary distance tree based on a continuous distance function. Its construction and search algorithms are very intuitive. The basic idea is to use binary search for distance-only information in multidimensional metric space, and feature space is divided by using the distance information between the points of the target point set of the feature space and Vantage Point. The construction complexity of VP-tree is $O(n\log_n)$, and the search complexity can ideally reach $O(\log_n)$ [7]. MVP-tree reduces the effort of distance calculations by increasing the number of Vantage Points and increasing the node output capacity. The construction complexity of MVP-tree is $O(n\log_m n)$, which is $O(n\log_2 n)$ higher than VP-tree, and the search complexity is less than $O(n)$ even in the worst case.

Suppose that a bucket contains n log samples, denoted by $S = \{s_1, s_2, \ldots, s_n\}$, and the Jaccard similarity function between them is $J(s_i, s_j)(i \in [1, n], j \in [1, n])$; p is used to store the precalculated similarity value, k is the maximum output capacity of the leaf node, and the variable level is used to record the number of Vantage Points from the root node to the current child node. The initial value of k is 1. The construction algorithm of MVP-tree is shown in Algorithm 1. The neighbor searching algorithm of MVP-tree is shown in Algorithm 2.

In these two algorithms, the metric space distance function $d(s_i, s_j) = 1/J(s_i, s_j)$. In Algorithm 2, the sample with the smallest distance from the sample to be detected

Require: Log set with n samples $S = \{s_1, s_2, \ldots, s_n\}$, metric space distance function $d(s_i, s_j)$, the precalculated similarity value p, the maximum output capacity of the leaf node k, the number of Vantage Points from the root node to the current child node level = 1
Ensure: MVP-Tree
(1) **function** BUILDTREE $(S, d, p, k, \text{level})$
(2) **if** $|\mathbf{S}| = 0$ **then return**
(3) **else if** $|\mathbf{S}| \leq \mathbf{k} + 2$ **then**
(4) Select an object s_{v1} randomly from S as the first point
(5) Compute $D_1 = \{d(s_i, s_{v1}) \mid s_i \in S, i \neq v1\}$
(6) $s_{v2} = \max(D_1)$, as another Vantage Point
(7) Compute $D_2 = \{d(s_j, s_{v2}) \mid s_j \in S, j \neq v2\}$
(8) **return**
(9) **else**
(10) Select an object s_{v1} randomly from S as the first point
(11) Compute $D_1 = \{d(s_i, s_{v1}) \mid s_i \in S, i \neq v1\}$
(12) **if** level $\leq p$ **then**
(13) Store s_i, PATH[level] $= d(s_i, s_{v1})$
(14) **end if**
(15) Sort D_1, $M =$ Median of D_1
(16) Define $SS_1 = \{s_i \mid d(s_i, s_{v1}) < M, s_i \in S, s_i \neq s_{v1}\}$, $SS_2 = \{s_j \mid d(s_j, s_{v1}) \geq M, s_j \in S\}$
(17) Select an object s_{v2} randomly from SS_2 as the first point
(18) Compute $D_2 = \{d(s_j, s_{v2}) \mid s_j \in S, j \neq v2\}$
(19) **if** level $\leq p$ **then**
(20) Store s_j, PATH[level] $= d(s_j, s_{v2})$
(21) **end if**
(22) Compute $D_{s1} = \{d(s_j, s_{v2}) \mid s_j \in SS_1\}$, $D_{s2} = \{d(s_j, s_{v2}) \mid s_j \in SS_2\}$
(23) Sort D_{s1} and D_{s2}, $M_1 =$ Median of SS_1, $M_2 =$ Median of SS_2, level+ = 2
(24) goto **(16)** for recursion
(25) **end if**
(26) **return** *result*
(27) **end function**

Algorithm 1: The construction algorithm of MVP-tree.

S_{detected} is searched; that is, the similarity between this sample and S_{detected} is the largest. We search the nearest neighbors for S_{detected} according to the above method, store these neighbors into a spare neighbor set, and then search for the next nearest neighbor.

3.2. Automatic Selection for k Neighbors. For kNN-based anomaly detection, k proper neighbors need to be selected for the sample to be detected. If most samples in k-nearest neighbors have the same category as the sample to be detected, the sample to be detected will be correctly classified. We say that such neighbor samples can have a positive impact on detection results. Therefore, k does not need to be very large, but k neighbor samples need to be as similar as possible. Because there is no prior knowledge, directly specifying the value of k is a challenge. At the same time, the exhaustive method is not desirable due to the excessive time overhead. Therefore, this paper proposes a method for automatically selecting k neighbors based on Silhouette Coefficient.

Assuming that the bucket of the sample to be detected contains m samples, the range of k is [1, m]. We define a spare neighbor set Spare_Neighbor to store the samples in the bucket and an actual neighbor set Actual_Neighbor, which contains a neighbor samples that can be finally used for anomaly detection. We search the nearest neighbor for the sample to be detected S and store it into Actual_Neighbor; then it is deleted from Spare_Neighbor. Thus, in the initial state, Spare_Neighbor contains $m - 1$ samples, and Actual_Neighbor contains one sample. Assume that the spare neighbor set is Spare_Neighbor $= \{\text{SN}_1, \text{SN}_2, \ldots, \text{SN}_b\}$ and the actual neighbor set is Actual_Neighbor $= \{\text{AN}_1, \text{AN}_2, \ldots, \text{AN}_a\}$, $a + b < m$. We search the next nearest neighbor for the sample to be detected S from Spare_Neighbor, and it is recorded as S_{near}. We calculate the Silhouette Coefficient of S_{near} as follows:

$$\text{in}J(S_{\text{near}}) = \frac{\sum_{i=1}^{a} J(\text{AN}_i, S_{\text{near}})}{a},$$

$$\text{out}J(S_{\text{near}}) = \frac{\sum_{j=1}^{b} J(\text{SN}_j, S_{\text{near}})}{b}, \tag{2}$$

$$\text{SC}(S_{\text{near}}) = \frac{\text{out}J(S_{\text{near}}) - \text{in}J(S_{\text{near}})}{\max(\text{out}J(S_{\text{near}}), \text{in}J(S_{near}))},$$

where $J(\text{AN}_i, S_{\text{near}})$ is the Jaccard similarity between sample AN_i and S_{near}; $J(\text{SN}_j, S_{\text{near}})$ is the Jaccard similarity between sample SN_j and S_{near}; $\text{in}J(S_{\text{near}})$ is the average similarity

Require: The sample to be detected S_{detected}, the distance threshold r
Ensure: The nearest sample S_{near}
(1) **function** QueryTree (S_{detected}, r)
(2) Compute $d(S_{\text{detected}}, s_{v1})$, $d(S_{\text{detected}}, s_{v2})$
(3) **if** $d(S_{\text{detected}}, s_{v1}) \geq r$ **then return** s_{v1}
(4) **end if**
(5) **if** $d(S_{\text{detected}}, s_{v2}) \geq r$ **then return** s_{v2}
(6) **end if**
(7) For the leaf tree nodes, goto **(8)**, for the intermediate tree nodes, goto **(16)**
(8) **for** $i = 1 \longrightarrow n$ **do**
(9) **if** $d(S_{\text{detected}}, s_{v1}) - r \leq d(s_i, s_{v1}) \leq d(S_{\text{detected}}, s_{v1}) + r$ and $d(S_{\text{detected}}, s_{v2}) - r \leq d(s_i, s_{v2}) \leq d(S_{\text{detected}}, s_{v2}) + r$ **then**
(10) **if** $\text{PATH}[i] - r \leq s_i$, $\text{PATH}[i] \leq \text{PATH}[i] + r$ **then**
(11) **if** $d(S_{\text{detected}}, s_i) \leq r$ **then return** s_i
(12) **end if**
(13) **end if**
(14) **end if**
(15) **end for**
(16) **if** level $\leq p$ **then**
(17) $\text{PATH}[\text{level}]] = d(S_{\text{detected}}, s_{v1})$
(18) **end if**
(19) **if** level $< p$ **then**
(20) $\text{PATH}[\text{level} + 1] = d(S_{\text{detected}}, s_{v2})$
(21) **end if**
(22) **if** $d(S_{\text{detected}}, s_{v1}) - r \leq M$ **then**
(23) **if** $d(S_{\text{detected}}, s_{v2}) - r \leq M_1$ **then**
(24) level+ = 2, goto **(23)** for recursion
(25) **end if**
(26) **if** $d(S_{\text{detected}}, s_{v2}) + r \geq M_1$ **then**
(27) level+ = 2, goto **(26)** for recursion
(28) **end if**
(29) **end if**
(30) **if** $d(S_{\text{detected}}, s_{v1}) + r \geq M$ **then**
(31) **if** $d(S_{\text{detected}}, s_{v2}) - r \leq M_2$ **then**
(32) *level*+ = 2, goto **(31)** for recursion
(33) **end if**
(34) **if** $d(S_{\text{detected}}, s_{v2}) + r \geq M_2$ **then**
(35) level+ = 2, goto **(34)** for recursion
(36) **end if**
(37) **end if**
(38) **end function**

Algorithm 2: The neighbor searching algorithm of MVP-tree.

between sample S_{near} and a samples in Actual_Neighbor; out$J(S_{\text{near}})$ is the average similarity between sample S_{near} and b samples in Spare_Neighbor; $\max(\text{out}J(S_{\text{near}}), \text{in}J(S_{\text{near}}))$ is the larger value between out$J(S_{\text{near}})$ and in$J(S_{\text{near}})$; and $\text{SC}(S_{\text{near}})$ is the Silhouette Coefficient of sample S_{near}.

We convert (2) into a more intuitive form, as shown in (3). The similarity between data in the same cluster is larger, and the similarity between different clusters is smaller. If sample S_{near} is more similar to cluster Actual_Neighbor, then in$J(S_{\text{near}})$ is larger than out$J(S_{\text{near}})$, and $\text{SC}(S_{\text{near}})$ is less than 0. It is highly possible that the category of S_{near} is consistent with that of most samples in Actual_Neighbor; otherwise, when in$J(S_{\text{near}})$ is less than out$J(S_{\text{near}})$, $\text{SC}(S_{\text{near}})$ is greater than 0, which means that the category of S_{near} and that of most samples in Actual_Neighbor are likely to be different:

$$\text{SC}(S_{\text{near}}) = \left\{ \begin{array}{ll} 1 - \dfrac{\text{in}J(S_{\text{near}})}{\text{out}J(S_{\text{near}})}, & \text{in}J(S_{\text{near}}) < \text{out}J(S_{\text{near}}) \\ 0, & \text{in}J(S_{\text{near}}) = \text{out}J(S_{\text{near}}) \\ \dfrac{\text{out}J(S_{\text{near}})}{inJ(S_{near})} - 1, & \text{in}J(S_{\text{near}}) > \text{out}J(S_{\text{near}}) \end{array} \right\}. \tag{3}$$

The value of k cannot be too small; we need to find as many neighbor samples with positive impact as possible. As the number of samples in Actual_Neighbor increases, the average distance between sample S_{near} searched from Spare_Neighbor and samples in Actual_Neighbor will become smaller. Therefore, if the Silhouette Coefficient of

Require: The sample to be detected S, m samples in the bucket of sample S, an actual neighbor set containing $a\,(a \in [1, m])$ neighbor samples that can be finally used for anomaly detection Actual_Neighbor $= \{AN_1, AN_2, \ldots, AN_a\}$ and the initial value of a is 1, a spare sample set containing the remaining $b\,(b \in [1, m])$ samples in the bucket Spare_Neighbor $= \{SN_1, SN_2, \ldots, SN_b\}$ and the initial value of b is $m-1$, $a+b=m$, the distance threshold r
Ensure: The final selected neighbors Actual_Neighbor
(1) **function** Auto $-$ $k(S, m,$ Spare_Neighbor, Actual_Neighbor)
(2) $S_{near} = \text{QueryTree}(S, r)$
(3) **for** $i = 1 \longrightarrow a$ **do**
(4) in $J(S_{near}) += J(AN_i, S_{near})$
(5) **end for**
(6) **for** $j = 1 \longrightarrow b$ **do**
(7) out $J(S_{near}) += J(SN_j, S_{near})$
(8) **end for**
(9) $SC(S_{near}) = (\text{out } J(S_{near}) - \text{in } J(S_{near}))/(\max(\text{out } J(S_{near}), \text{in } J(S_{near})))$
(10) **if** $SC(S_{near}) < 0$ **then**
(11) **if** $SN_p = S_{near}\,(p \in [1, b])$ **then**
(12) $a \mathrel{+}= 1$
(13) $AN_a = SN_p$
(14) delete SN_p from Spare_Neighbor
(15) $b \mathrel{-}= 1$
(16) goto **(2)**
(17) **end if**
(18) **else**
(19) **return** Actual_Neighbor
(20) **end if**
(21) **end function**

Algorithm 3: The strategy of automatic selection for k neighbors.

S_{near} is less than 0, we think that S_{near} can have a positive impact on anomaly detection. We remove S_{near} from Spare_Neighbor and store it into Actual_Neighbor, which is denoted as AN_{a+1}. Then the next nearest neighbor sample is searched from Spare_Neighbor, which is denoted as S_{near}; otherwise, when the Silhouette Coefficient of S_{near} is greater than 0, it means that S_{near} is not similar to samples in Actual_Neighbor. At this point, we stop searching for neighbors and set the optimal k value to the number of samples in Actual_Neighbor. The strategy of automatic selection for k neighbors is shown in Algorithm 3.

Instead of using a fixed k value, we choose the appropriate k for log data with different distributions through the method described above. In this way, we can get as many neighbor samples as possible, which are consistent with the same type of samples to be detected, thereby improving the accuracy of the anomaly detection.

3.3. Anomaly Detection. We use the final neighbors in actual neighbor set Actual_Neighbor to classify the sample to be detected. In our method, logs can be divided into two categories, that is, normal logs and abnormal logs. Abnormal logs are negative samples, while the rest are positive ones. In k neighbor samples, the positive sample set is $P = \{S_{p1}, S_{p2}, \ldots, S_{pm}\}$, and the negative sample set is $F = \{S_{f1}, S_{f2}, \ldots, S_{fn}\}$, where $0 \le m \le k, 0 \le n \le k, m + n = k$. The average similarity for the two types of samples is calculated, as shown in (4) and (5), where $\text{Aver} - \text{sim}(S, P)$ is the average similarity for the m positive samples and $\text{Aver} - \text{sim}(S, F)$ is the average similarity for the n negative samples:

$$\text{Aver} - \text{sim}(S, P) = \frac{\sum_{i=1}^{m} J(S, S_{pi})}{m}, \tag{4}$$

$$\text{Aver} - \text{sim}(S, F) = \frac{\sum_{j=1}^{n} J(S, S_{fj})}{n}. \tag{5}$$

We compare the values of Aver−sim (S, P) and Aver−sim (S, F). If $\text{Aver} - \text{sim}(S, P) \ge \text{Aver} - \text{sim}(S, F)$, we mark sample S as normal; otherwise, we mark it as abnormal. The strategy of anomaly detection is shown in Algorithm 4.

4. Experiment and Analysis

4.1. Log Data. We used six different types of log data to evaluate our method: Liberty, Blue Gene/L (BGL), Thunderbird, Spirit, HDFS, and Zookeeper. These logs have a total of 90 GB and contain 314,647,599 anomalies, as shown in Table 2. Liberty is a server application system, BGL and Thunderbird are supercomputers, Spirit is an operating system, and HDFS and Zookeeper are distributed systems. The Liberty, Thunderbird, and Spirit systems are installed in the Sandia National Laboratory (SNL) in New Mexico, USA, and the BGL is installed at the Lawrence Livermore National Laboratory (LLNL) in California.

Require: The sample to be detected S. An actual neighbor set containing $k(a \in [1, m])$ neighbor samples for sample S Actual_Neighbor = $\{AN_1, AN_2, \ldots, AN_k\}$. A positive sample set containing m normal samples in Actual_Neighbor $P = \{S_{p1}, S_{p2}, \ldots, S_{pm}\}$. A negative sample set containing n abnormal samples in Actual_Neighbor $F = \{S_{f1}, S_{f2}, \ldots, S_{fn}\}.m + n = k, m \in [1, k], n \in [[1, tk]$
Ensure: The label of S

```
(1) function ANOMALYDETECTION (P, F, m, n)
(2)    for i = 1 ⟶ m do
(3)       Aver − sim(S, P)+ = J(S, S_pi)
(4)    end for
(5)    for j = 1 ⟶ n do
(6)       Aver − sim(S, F)+ = J(S, S_fi)
(7)    end for
(8)    if Aver − sim(S, P)+ = J(S, S_pi) ≥ Aver − sim(S, F)+ = J(S, S_fi) then
(9)       return normal
(10)   else
(11)      return abnormal
(12)   end if
(21) end function
```

Algorithm 4: The strategy of anomaly detection.

Table 2: Log data.

Systems	Size	Log lines	Number of anomalies	System type
Liberty	29.5 G	266991013	191,839,098	Server application system
BGL	1.207 G	4747963	949024	Supercomputer
Thunderbird	27.367 G	211212192	43,087,287	Supercomputer
Spirit	30.28 G	272298969	78360273	Operating system
HDFS	1.58 G	11175629	362793	Distributed system
Zookeeper	10.4 M	74380	49124	Distributed system

4.2. Research Questions. To evaluate our method, we designed experiments to address the following three research questions:

RQ1: Can minhash and MVP-tree based neighbor search method improve the efficiency of neighbor search in kNN algorithm?

When using kNN-based log anomaly detection, there are many factors that affect the efficiency of neighbor searching, three of which are the vector dimension of the log data, the effort of distance calculation, and the number of samples that need to be compared when searching neighbors. Therefore, we design experiments for the above three aspects and study whether our method can improve the efficiency of neighbor searching in kNN algorithm. Besides, the overall searching time is also compared.

(1) Reduction in vector dimension. Generally speaking, it is a common method to convert log data into vectors by extracting features according to word frequency, while minhash algorithm extracts features based on word frequency matrix. We get different sizes of sample sets from the six datasets used in this paper and use minhash algorithm in each dataset, respectively, to obtain the vector dimension. We compare the result with the dimension of word frequency vecto, and study whether the dimension of log vectors obtained with minhash algorithm is reduced.

(2) Reduction in the effort of distance calculation. We calculate the number of samples in the biggest bucket after each dataset is divided by hash functions and study how much effort of distance calculation is reduced by our neighbor search method.

(3) Reduction in the number of samples to be compared. We calculate the average and maximum number of nodes that need to be compared when we search the nearest neighbor from MVP-tree model, and compare the results with the number of samples that need to be compared in the traditional kNN algorithm; then we can study whether the number of samples that need to be compared can be reduced by our MVP-tree based method.

(4) Reduction in the overall searching time. We select sample sets with different sizes from six datasets and use the original neighbor search method and our nearest neighbor search method. The overall time required by the two methods is compared to analyze whether the neighbor search method in this paper can improve the efficiency of neighbor searching.

RQ2: Can the k neighbors selected in this paper improve the accuracy of kNN algorithm?

We first use a fixed k value for anomaly detection; then the k value automatically selected with our method is used

for anomaly detection. Accuracy of the anomaly detection corresponding to the two values is compared to study whether the k neighbors selected in this paper can improve the accuracy of kNN algorithm. The fixed k value was determined through several experiments, and the details are shown in Section 4.3.

RQ3: Is our anomaly detection method superior to other homologous anomaly detection methods?

As described in Section 3, the log-based anomaly detection method proposed in this paper is based on kNN algorithm, which is an outlier detection method in machine learning. We design experiments, respectively compare the method we proposed with other outlier detection methods and methods without outlier detection, and analyze the results. Reference [6] uses a clustering algorithm to sort the log sequences, which is an outlier detection method and the same as our method. The methods in [8, 9] are not outlier detection methods; [8] uses an anomaly detection method based on finite state automaton and [9] uses the information entropy of log messages for identifying exceptions.

We use these four methods for anomaly detection on six datasets, and the detection results are compared from three aspects: accuracy, recall, and F measure. Then we analyze the advantages and disadvantages of these four methods and study whether the method proposed in this paper is superior to the other three methods.

4.3. Experimental Design and Results. In this section, we analyze the results of the above three research questions. For the six datasets used in the paper, we divide each dataset into a training set and a test set, respectively, each of which is half of the overall dataset.

RQ1: Can minhash and MVP-tree based neighbor search method improve the efficiency of neighbor search in kNN algorithm?

4.3.1. Reduction in Vector Dimension. Table 3 shows the dimension of log vectors based on the BoW model and the log vector dimension based on the minhash algorithm. BoW model extracts the word frequency feature, and the vector dimension is equal to the number of word types in log data. Results in Table 3 represent the number of word types after filtering out the unnecessary words (the words with numbers) in the training set.

In the minhash-based neighbor search method, dimension of the log vector is the same as the number of word frequency matrixes obtained by changing the word orders. We select 3 hash functions, such as $h((3x+1)\text{mod}5)$, where x represents the original position of the word. For example, data in the first row of the original matrix is changed to the fourth row in the new matrix after transformation with $h((3x+1)\text{mod}5)$. We use each hash function to transform the matrix 10 times and get 30 matrixes in total. Each log line extracts a feature from each matrix. Finally, the dimension of log vector based on minhash algorithm is 30, which is smaller than most of the dimensions with BoW method. Therefore, the minhash algorithm can tremendously reduce the dimension of log vectors.

4.3.2. Reduction in the Effort of Distance Calculation. We use the minhash algorithm to group similar logs in the training set into the same bucket, and the number of samples in different buckets is different. Table 4 shows the average and maximum number of samples in the bucket for six datasets. In the traditional kNN algorithm, we need to calculate distances between the sample to be detected and all samples in the training set. In our experiments, the training set is set to 50% of the overall data; then the effort for traditional method is equal to 50% of the overall data.

As shown in Table 4, the number of samples in the maximum bucket of BGL data is the largest among the six datasets, which is still much smaller than the effort with traditional kNN algorithm. Therefore, the minhash-based neighbor searching method can greatly reduce the effort of distance calculation.

4.3.3. Reduction in the Number of Samples to Be Compared. Table 5 shows the number of nodes we need to compare when searching for the nearest neighbor set from MVP-tree. For each sample to be detected, the number of samples to be compared is different, so results in the table represent the average numbers of the samples to be compared.

Traditional kNN algorithm needs to compare distances between the sample to be detected and all training set samples, and then the nearest one is selected. Therefore, the number of samples to be compared for the traditional kNN algorithm is the size of training set, which is 50% of the total dataset.

Obviously, compared with the traditional kNN, our nearest neighbor search method reduces the number of samples to be compared. Our method builds a tree model for samples which are similar to the sample to be detected and then compares the nodes from top to bottom. In this process, we do not need to compare all the nodes. Therefore, our method greatly reduces the number of samples to be compared when searching for the nearest neighbor.

4.3.4. Reduction in the Overall Searching Time. Table 6 shows the time cost of the proposed method and the traditional kNN method when searching for the nearest neighbor under different dimensions and different sizes of data sample sets. Intuitively, no matter which method is used, our method or the kNN algorithm, when the data size is fixed, the smaller the dimension of the data, the shorter the search time of the nearest neighbor; when the dimension of the data is fixed, the smaller the number of data samples, the shorter the search time.

When dimension and data size are equal, our method is much more efficient than the traditional kNN algorithm. For example, when the vector dimension is 100 and the data size is 1G, the search time of our method is generally within 30 s, while it is about 2-3 minutes with the traditional kNN algorithm; when the dimension is increased to 300, it will take nearly 8 minutes to search neighbors for 1 G Spirit dataset.

Table 3: Dimension comparison with BoW and minhash.

Systems	Size	Log lines	Number of anomalies	System type
Liberty	29.5 G	266991013	191,839,098	Server application system
BGL	1.207 G	4747963	949024	Supercomputer
Thunderbird	27.367 G	211212192	43,087,287	Supercomputer
Spirit	30.28 G	272298969	78360273	Operating system
HDFS	1.58 G	11175629	362793	Distributed system
Zookeeper	10.4 M	74380	49124	Distributed system

Table 4: Comparison of the effort of distance calculation.

Datasets	Size Average samples in buckets	Log lines Maximum samples in buckets	Effort of distance calculation with traditional kNN
Liberty	29.5 G	266991013	191,839,098
BGL	1.207 G	4747963	949024
Thunderbird	27.367 G	211212192	43,087,287
Spirit	30.28 G	272298969	78360273
HDFS	1.58 G	11175629	362793
Zookeeper	10.4 M	74380	49124

Table 5: Comparison of the number of samples to be compared.

Datasets	Number of samples to be compared with traditional kNN	Number of samples to be compared with MVP-tree
Liberty	133,495,506	205
BGL	2,373,981	2603
Thunderbird	105,606,096	479
Spirit	136,149,484	205
HDFS	5,587,814	141
Zookeeper	37,190	1730

Table 6: Comparison of the overall searching time(s).

	Dimension	30	100	300		100	
	Data size		1 G		10 M	100 M	1 G
Liberty	Our method	12.38	27.34	35.02	0.281	2.243	27.34
	Traditional kNN	78.39	171.98	336.36	2.63	21.05	171.98
BGL	Our method	11.09	21.52	33.42	0.279	2.546	21.52
	Traditional kNN	35.01	103.35	198.46	1.68	13.18	103.35
Thunderbird	Our method	13.71	28.25	29.34	0.273	2.037	28.25
	Traditional kNN	114.68	236.26	440.97	3.2	27.29	236.26
Spirit	Our method	12.63	30.28	34.58	0.254	1.899	30.28
	Traditional kNN	112.21	247.79	463.87	3.14	27.33	247.79
HDFS	Our method	6.16	19.79	23.15	0.181	1.32	19.79
	Traditional kNN	82.08	177.69	320.46	2.51	20.99	177.69
Zookeeper	Our method	8.01	17.49	21.76	0.224	1.68	17.49
	Traditional kNN	86.29	197.25	366.99	2.79	23.5	197.25

Experiments show that our method based on minhash and MVP-tree can improve the searching time.

RQ2: Can the k neighbors selected in this paper improve the accuracy of kNN algorithm?

We use several different k values on some small datasets, and the results show that when k is set to 5, we can achieve better results. Therefore, in our experiments, the fixed k value is set as 5. Our method selects different k for datasets with different distributions, each dataset is divided into different buckets, and the data distribution in these buckets is also different, so the best k corresponding to each bucket is also different. Therefore, we will not show the automatically selected k value here.

We use a Silhouette Coefficient–based method to select k neighbors automatically. Figure 6 shows the accuracy comparison between the automatically selected k neighbors and the fixed k. We selected three datasets, Liberty, BGL, and HDFS, for anomaly detection, because the types of these three datasets are different. Liberty is a server application

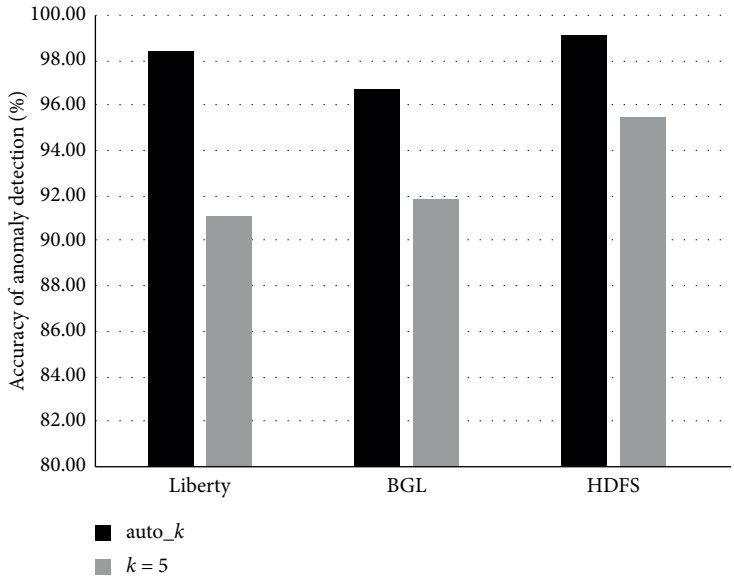

Figure 6: Comparison of the accuracy of automatically selected k neighbors and fixed k value.

system, BGL is a supercomputer, and HDFS is a distributed system.

Obviously, the method of automatically selecting k neighbors can improve the accuracy of anomaly detection. When the value of k is 5, the accuracy of anomaly detection on the three datasets has been relatively high, in which HDFS even reaches 95.4%. However, our method can still improve it to 99%. The larger the log data, such as Liberty (29.5 G), the higher the accuracy our method can improve. This is because when the size of the training set becomes larger, we can choose proper k neighbors according to the distribution of samples in the training set, which cannot be achieved by the fixed value of 5.

We also count the number of normal logs and abnormal logs in Liberty, BGL, and HDFS datasets. In the Liberty dataset, the number of normal logs is 46 times that of abnormal logs. The difference in Blue Gene/L dataset is not so obvious, but the difference is still large; the number of normal logs is 7.95 times the number of abnormal logs. Among the three datasets, the difference between the number of normal logs and abnormal logs is the most. Correspondingly, the difference in accuracy between the automatic k and fix k is the most, too. Thus, the results show that the k neighbors selected with our method can improve the accuracy of kNN algorithm, and the improvement is more effective on more uneven datasets.

In conclusion, our method of automatically selected neighbors can improve the accuracy of anomaly detection.

RQ3: Is our anomaly detection method superior to other homologous anomaly detection methods?

We compare the method proposed in this paper with the three methods that are, respectively, proposed in [6, 8, 9]. The results of the comparison are shown in Figures 7–12.

As shown in Figure 7, we take logs generated by BGL as an example; all of the four methods can be effectively used for log-based anomaly detection. The method used in [6] can achieve high recall rate (87%), but its accuracy and F measure are relatively low. This is because log vectors have the characteristics of high dimension and sparsity, and it is difficult for the log clustering method to separate abnormal logs from normal logs accurately, which results in a large number of false positives and a lower accuracy of anomaly detection. At the same time, the hierarchical clustering algorithm is an unsupervised learning method. When applied to log detection, this method is less accurate than the supervised learning method used in this paper. Our method reduces the dimensions of log vectors, as discussed in RQ1. Thus, our method performs better in terms of accuracy and F measure. The method in [9] can achieve higher accuracy (85.27%), but it is not higher than the method in this paper. The anomaly detection method it used is a finite state automaton, which is not the outlier detection method. kNN algorithm used in this paper is an outlier detection method, which is more prominent on the log data with uneven distribution.

The recall and F measure achieved using the method in [9] are the lowest among the four methods. Information entropy is used in [9] to detect anomalies of log messages, which are estimated by the probability of terms appearing in logs. In our method, we extract features from multiple matrixes transformed with hash functions, which can better reflect the log features than the word frequency features, so higher recall and F measure can be achieved in our method.

All the four methods perform better on two of the six datasets, Thunderbird and BGL. These two log sets are produced by the supercomputer with a simpler structure. Although the accuracy of anomaly detection with the method in [6] is relatively lower than that with the other three methods, it is the highest (62.13%) on the Zookeeper dataset. The reason behind this is that the Zookeeper dataset used in this paper is relatively small (only 10.4 M) and is more suitable for handling clustering algorithms.

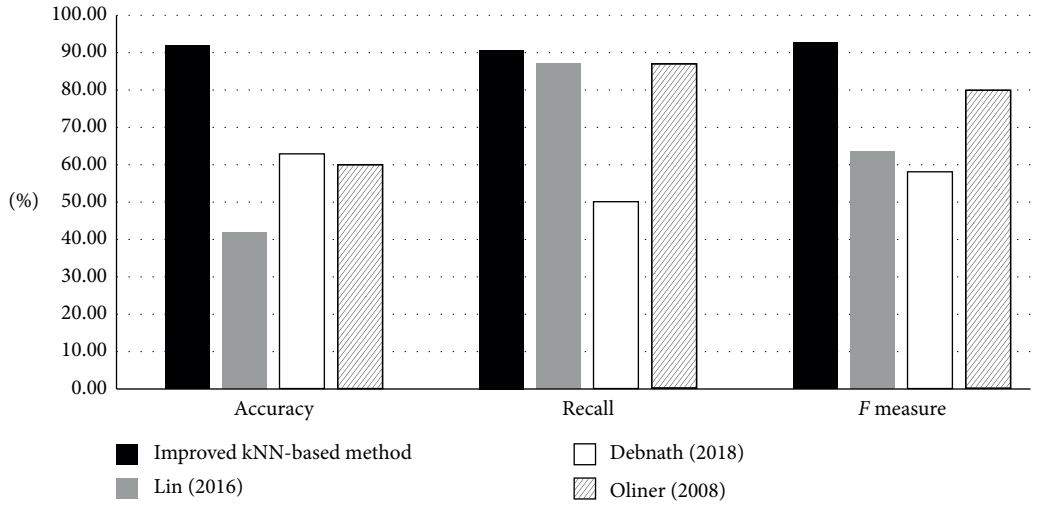

Figure 7: Comprehensive comparison on BGL log set.

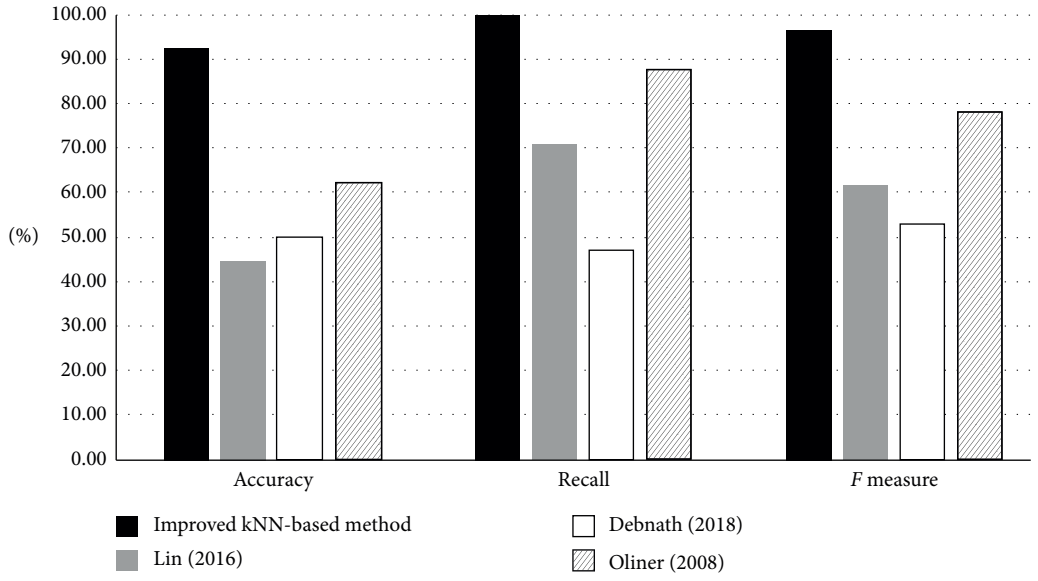

Figure 8: Comprehensive comparison on Spirit log set.

Our results show that our kNN-based anomaly detection method is effective for log-based anomaly detection and can better demonstrate its superiority when dealing with heterogeneous data.

5. Discussion

5.1. Categories of the Abnormal Logs. This paper only considers detecting the log lines with abnormal messages, but the abnormal logs are not classified. Usually, the abnormal logs can be divided into several categories, such as hardware fault, software fault, and network fault, and there are also some differences in quantities and severity levels between each fault category. From the perspective of severity levels, abnormal log data can be classified into warning, error, failure, fatal, and so on. The log data can be analyzed from the perspective of different categories of anomaly logs in the future work.

5.2. Limitation of Training Set Selection. In this paper, a training set is randomly selected. Log data has the characteristics of quantity imbalance, so the randomly selected training set may also be unbalanced. Although our method of automatically selecting k neighbors can alleviate the impact of this imbalance on the accuracy of anomaly detection, if we can improve the balance of training set, the accuracy of log-based anomaly detection will be further improved.

6. Related Work

Locality Sensitive Hashing (LSH) [10] is arguably the most popular unsupervised hashing method and has been applied to many problem domains, including information retrieval and computer vision. Reference [11] has shown that there exists a simple and general framework for solving the (r1,

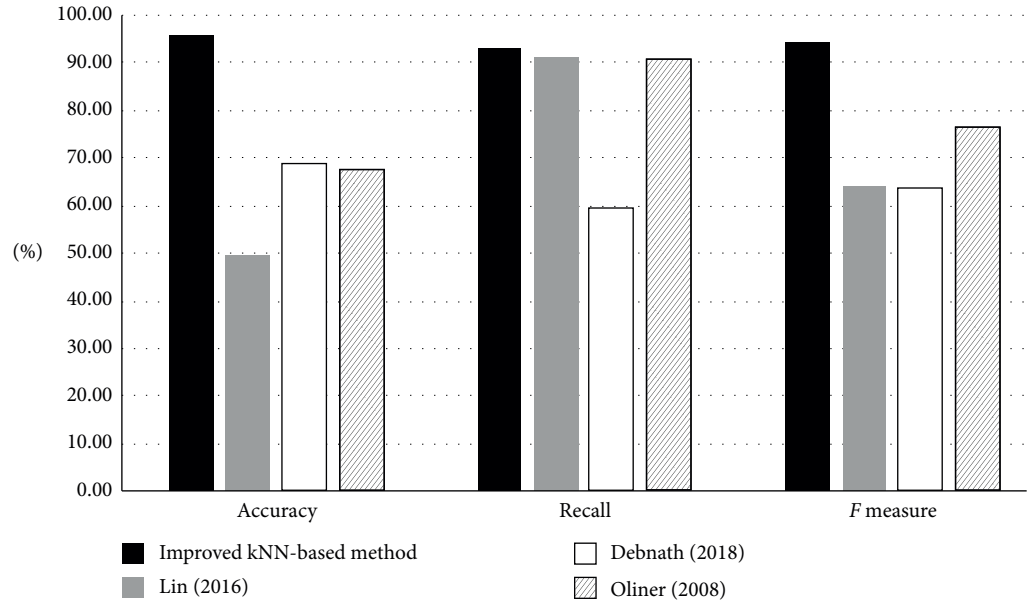

Figure 9: Comprehensive comparison on Thunderbird log set.

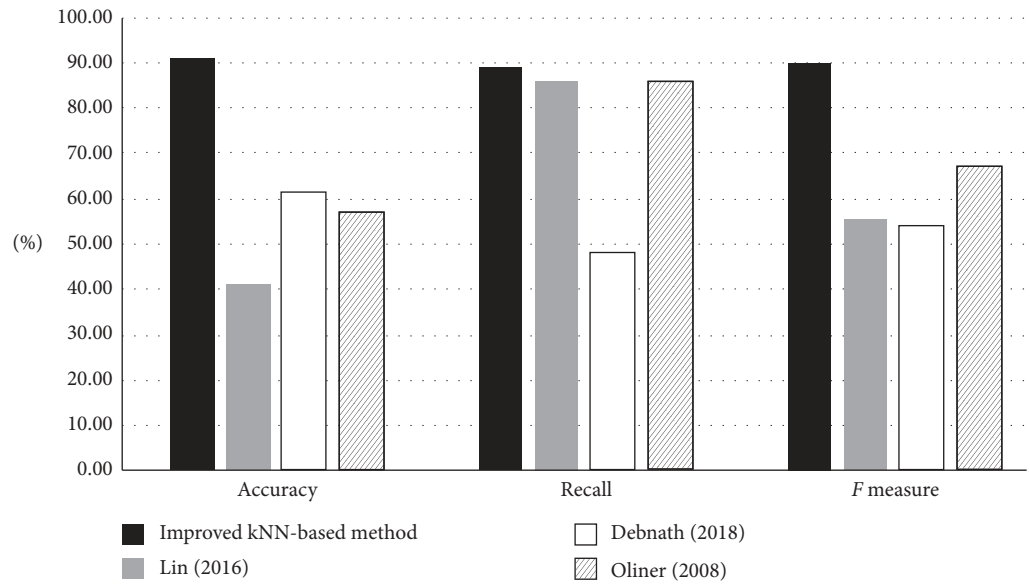

Figure 10: Comprehensive comparison on Liberty log set.

$r2$)-near neighbor problem using only few LSH functions and with a reduced word-RAM complexity matching the number of lookups. Minhash algorithm used in our method is an LSH algorithm with Jaccard similarity method, and we combine it with MVP-tree to further improve the efficiency of neighbor search. Reference [12] uses a minhash-based method to obtain sublinear complexity in number of patterns. The Thunderbird, Windows, and Spark datasets used by them are the largest datasets to be used for log parsing so far. For most of the datasets, Delog fares almost two times better in training time performance as compared to the previous state-of-the-art. Moreover, the quality of patterns generated by Delog is also consistently better than the existing parsing algorithms. In our method, minhash is used to improve the efficiency of anomaly detection rather than log parsing. Because minhash can reduce the effort of distance calculation and the number of logs that need to be compared, it can also improve the efficiency of anomaly detection. Reference [13] proposes a relatively general way of creating efficient Las Vegas versions of state-of-the-art high-dimensional search data structures. It showed an optimal algorithm for the nearest neighbor without false negatives for Hamming space and Braun-Blanquet metric. The model mentioned in this method is for two data structures, one of which is Jaccard similarity approximation search, and it is the method used in this paper. The authors in [14] proposes

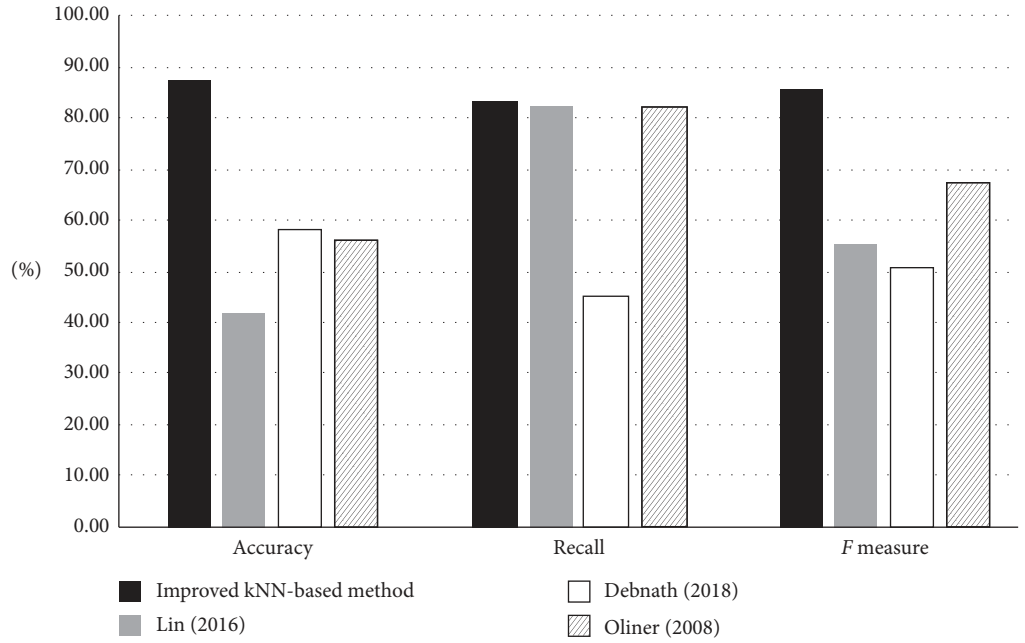

Figure 11: Comprehensive comparison on HDFS log set.

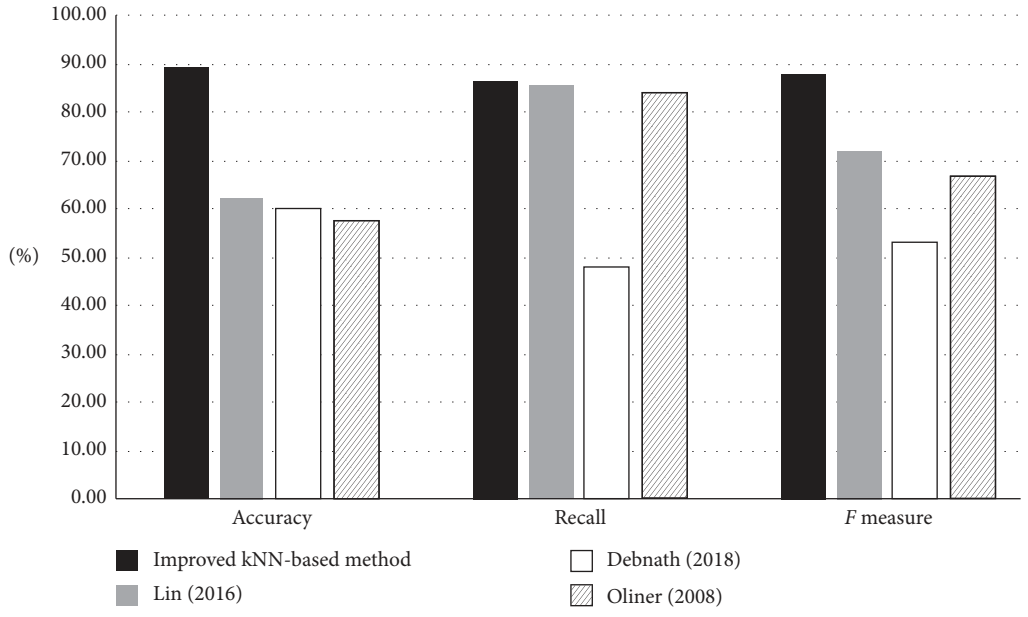

Figure 12: Comprehensive comparison on Zookeeper log set.

an unsupervised anomaly detection method called ACE, which is superior to existing methods designed without taking into account the computational complexity of the estimation process. They leverage advances in probabilistic indexing and redesign a superfast statistical measure which requires significantly lesser resources. At the core of the ACE algorithm, there is a novel statistical estimator which is derived from the sampling view of LSH. In our method, minhash is used to improve the efficiency of kNN, where kNN is a supervised learning method. Studies have shown that supervised learning methods are generally more accurate than unsupervised learning methods [4], so the accuracy of anomaly detection combined with minhash and kNN is higher.

A number of variations of the kNN-based approaches have also been developed. Reference [15] proposes a LMKNCN classifier that assigns to each query pattern a class label with the nearest local centroid mean vector so as to improve the classification performance. The proposed scheme not only takes into account the proximity and spatial distribution of k neighbors, but also utilizes the local mean vector of k neighbors from each class in making classification

decision. Reference [16] utilizes sparse representation and collaborative representation to design the effective nearest neighbor classification. Guo et al. propose two locality constrained representation-based k-nearest neighbor rules with the purpose of further improving the kNN-based classification performance. One is the weighted representation-based k-nearest neighbor rule (WRkNN), and the other is the weighted local mean representation-based k-nearest neighbor rule (WLMRkNN). In the linear combination of the class-specific k-local mean vectors, to represent the test sample, the localities of k-local mean vectors per class are considered as the weights to constrain the representation coefficients of k-local mean vectors. Reference [17] proposes a generalized mean distance-based kNN classifier, which is called GMDkNN. In this classifier, the multi-generalized mean distances and the nested generalized mean of each class are introduced. They calculated the k-local mean vectors per class, which can represent the local sample distributions of each class. The proposed method can employ more nearest neighbors for the favorable classification and has less sensitiveness to the values of k.

There are also many log-based anomaly detection methods. Reference [6] uses a cluster-based method to detect anomalies in logs. They take into account all the features of the online service system logs and create vectors that contain word messages from log message sequences, in which the log message sequences are obtained from the unique task ID. In the process of classification, different events have different effects on problem identification, so they assign weights to log messages and group similar log sequences into the same category. Reference [18] adopts a statistical approach to study how to scale up specification mining and other log analysis algorithms for long and complex logs. The scalability issues with different algorithms are addressed. Only a subset of the log data is analyzed, and the validity of the results is also statistically guaranteed simultaneously.

There are also many cases of log-based anomaly detection without outlier detection methods. In [4], several machine learning-based anomaly detection methods are studied, including supervised learning methods and unsupervised learning methods. Most of the supervised learning methods are superior to the unsupervised ones in terms of accuracy, recall rate, and F measure. In the supervised learning method, SVM has higher efficiency, but the method needs to adjust more parameters. However, kNN algorithm only needs to estimate k value, and no training is needed, which is more convenient to use.

Most research studies [3, 19–22] on log analysis and anomaly detection with data mining and machine learning technologies have focused on extracting useful information (events, invariant, etc.) from logs. Since some log data may not have these similar identifier fields, in this paper, we group logs with high similarity and convert them to vectors with a minhash-based method.

Other prior work on log-based anomaly detection focuses on detecting dependencies [23], anomalies [24], and performance debugging [25, 26]. More sophisticated analysis has included the study of the statistical properties of reported failure events to localize and predict faults [27] and mining patterns from multiple log events [28].

7. Conclusion

In this paper, we propose a log-based anomaly detection method with efficient neighbor searching and automatic neighbor selection. Because supervised learning methods can achieve higher accuracy in anomaly detection, we use kNN algorithm for log-based anomaly detection. Due to the large scale of log data, kNN algorithm has a low efficiency of neighbor searching. This paper proposes an efficient neighbor search method, which combines minhash algorithm and MVP-tree to reduce the effort of distance calculation and the number of samples that need to be compared, improving the neighbor search efficiency of kNN-based anomaly detection. Since the quantity imbalance of the log data will have a negative impact on the accuracy of kNN algorithm, we propose a method for automatically selecting k neighbors based on the Silhouette Coefficient, which selects k neighbor samples that can have a positive impact on the detection results. In order to verify the validity and universality of our method, we performed experiments on six log datasets with different types, and the results show that our method can be effectively used for log-based anomaly detection and improve the efficiency of anomaly detection based on kNN algorithm. At the same time, the accuracy of the anomaly detection is guaranteed.

Data Availability

The datasets used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare no conflicts of interest.

Acknowledgments

This work was supported in part by grants of National Natural Science Foundation of China (61672392 and 61373038) and National Key Research and Development Program of China (2016YFC1202204).

References

[1] S. Zawoad, A. K. Dutta, and R. Hasan, "SecLaaS," in *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security—ASIA CCS '13*, pp. 219–230, Hangzhou, China, May 2013.

[2] P. Patel and D. Cassou, "Enabling high-level application development for the internet of things," *Journal of Systems and Software*, vol. 103, pp. 62–84, 2015.

[3] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the 22nd ACM Symposium on Operating Systems Principles 2009, SOSP 2009*, pp. 117–132, Big Sky, Montana, USA, October 2009.

[4] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: system log analysis for anomaly detection," in *Proceedings of the 27th*

IEEE International Symposium on Software Reliability Engineering, ISSRE 2016, pp. 207–218, Ottawa, Canada, October 2016.

[5] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987.

[6] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016*, pp. 102–111, Austin, TX, USA, May 2016.

[7] P. Sarolahti, M. Kojo, and K. Raatikainen, "F-RTO," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 2, pp. 51–63, 2003.

[8] B. Debnath, M. Solaimani, M. A. Gulzar et al., "Loglens: a real-time log analysis system," in *Proceedings of the 38th IEEE International Conference on Distributed Computing Systems, ICDCS 2018*, pp. 1052–1062, Vienna, Austria, July 2018.

[9] A. J. Oliner, A. Aiken, and J. Stearley, "Alert detection in system logs," in *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008)*, pp. 959–964, Pisa, Italy, December 2008.

[10] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proceedimgs of The VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases*, pp. 518–529, September 1999, Edinburgh, UK.

[11] T. Christiani, "Fast locality-sensitive hashing frameworks for approximate near neighbor search," in *Proceedings of the Similarity Search and Applications—12th International Conference, SISAP 2019*, pp. 3–17, Newark, NJ, USA, October 2019.

[12] A. Agrawal, A. Dixit, D. Kapadia, R. Karlupia, V. Agrawal, and R. Gupta, "Delog: a privacy preserv-ing log filtering framework for online compute platforms," 2019, https://arxiv.org/abs/1902.04843.

[13] T. D. Ahle, "Optimal Las Vegas locality sensitive data structures," 2017, https://arxiv.org/abs/1704.02054.

[14] C. Luo and A. Shrivastava, "Arrays of (locality-sensitive) count estimators (ACE): anomaly detection on the edge," in *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018*, pp. 1439–1448, Lyon, France, April 2018.

[15] J. Gou, W. Qiu, Z. Yi, Y. Xu, Q. Mao, and Y. Zhan, "A local mean representation-based K -nearest neighbor classifier," *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 3, pp. 1–25, 2019.

[16] J. Gou, W. Qiu, Z. Yi, X. Shen, Y. Zhan, and W. Ou, "Locality constrained representation-based k-nearest neighbor classification," *Knowledge-Based Systems*, vol. 167, pp. 38–52, 2019.

[17] J. Gou, H. Ma, W. Ou, S. Zeng, Y. Rao, and H. Yang, "A generalized mean distance-based k-nearest neighbor classifier," *Expert Systems with Applications*, vol. 115, pp. 356–372, 2019.

[18] T. Barik, R. DeLine, S. M. Drucker, and D. Fisher, "The bones of the system: a case study of logging and telemetry at microsoft," in *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016*, pp. 92–101, Austin, TX, USA, May 2016.

[19] D. Lo, H. Cheng, J. Han, S. Khoo, and C. Sun, "Classification of software behaviors for failure detection: a discriminative pattern mining approach," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 557–566, Paris, France, June 2009.

[20] T. Reidemeister, M. Jiang, and P. A. S. Ward, "Mining unstructured log files for recurrent fault diagnosis," in *Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management, IM 2011*, pp. 377–384, Dublin, Ireland, May 2011.

[21] W. Xu, L. Huang, and M. I. Jordan, "Experience mining google's production console logs," in *Proceedings of the Workshop on Managing Systems via Log Analysis and Machine Learning Techniques, SLAML'10*, Vancouver, Canada, October 2010, https://www.usenix.org/conference/slaml10/experience-mining-googles-production-console-logs.

[22] W. Xu, L. Huang, A. Fox, D. A. Patterson, and M. I. Jordan, "Online system problem detection by mining patterns of console logs," in *Proceedings of the ICDM 2009, The Ninth IEEE International Conference on Data Mining*, pp. 588–597, Miami, FL, USA, December 2009.

[23] J. Lou, Q. Fu, Y. Wang, and J. Li, "Mining dependency in distributed systems through unstructured logs analysis," *Operating Systems Review*, vol. 44, no. 1, pp. 91–96, 2010.

[24] G. Jiang, H. Chen, C. Ungureanu, and K. Yoshihira, "Multi-resolution abnormal trace detection using varied-length n-grams and automata," in *Proceedings of the Second International Conference on Autonomic Computing (ICAC 2005)*, pp. 111–122, Seattle, WA, USA, June 2005.

[25] J. Tan, X. Pan, S. Kavulya, R. Gandhi, and P. Narasimhan, "SALSA: analyzing logs as state machines," in *Proceedings of the First USENIX Workshop on the Analysis of System Logs, WASL 2008*, San Diego, CA, USA, December 2008, http://www.usenix.org/events/wasl/tech/full papers/tan/tan.pdf.

[26] J. Tan, X. Pan, S. Kavulya, R. Gandhi, and P. Narasimhan, "Mochi: visual log-analysis based tools for debugging hadoop," in *Proceedings of the Workshop on Hot Topics in Cloud Computing, HotCloud'09*, San Diego, CA, USA, June 2009, https://www.usenix.org/conference/hotcloud-09/mochi-visual-log-analysis-based-tools-debugging-hadoop.

[27] Y. Liang, Y. Zhang, A. Sivasubramaniam, M. Jette, and R. K. Sahoo, "Bluegene/l failure analysis and prediction models," in *Proceedings of the 2006 International Conference on Dependable Systems and Networks (DSN 2006)*, pp. 425–434, Philadelphia, PA, USA, June 2006.

[28] J. L. Hellerstein, S. Ma, and C. Perng, "Discovering actionable patterns in event data," *IBM Systems Journal*, vol. 41, no. 3, pp. 475–493, 2002.