Doctor Thesis

Research of Applying Machine Learning Methods to Outlier Detection in Wireless Sensor Networks

by

Tianyu Zhang

March 2018

Graduate School of Applied Informatics
University of Hyogo

Abstract

Wireless sensor networks (WSNs) can be flexibly deployed and used to collect data from various environments. By analyzing the collected data, WSNs can be used for such tasks as environment monitoring, disaster prevention, and event detection. However, collected datasets sometimes contain outliers, which obviously reduce the accuracy of data analysis and the performance of the WSN (e.g., the outliers may trigger a false alarm that generates unnecessary fears). Therefore, removing such outliers before analyzing the collected data is necessary to improve the performance of the WSNs. Outlier detection is the process of data analysis. In WSNs, outlier detection involves two major approaches, which are defined as centralized and distributed. Our proposed algorithms use the distributed approach, which enables every sensor node to detect outliers on its own and locally. Therefore, in this doctoral thesis, we propose three algorithms for distributed detection of outliers, all based on machine learning. The first and second algorithms are based on supervised and unsupervised learning, respectively. The third is designed to improve the performance of clustering algorithms categorized as unsupervised learning.

The first algorithm is based on supervised learning. It first uses training data to train a classifier on a powerful base node and then distributes this classifier into every remote sensor node. Moreover, this method is founded on a widely used assumption in WSNs in which the entire deploying environment has the same condition. Using this assumption, we can simply gather the training data by defining a normal situation in such an environment. In this simple case, using a user-determined threshold is sufficient. For example, if a WSN is deployed to monitor the temperature of a store, we can determine a threshold based on the previously collected normal data. The threshold can be used to detect those data that represent an outlier. However, when WSN-collected data points contain multiple features, the method based on a threshold is not appropriate. Because a situation involving a data point, such as a normal situation or outlier, is commonly determined by multiple features, when data points have multiple features, a decision bound is used to detect the outliers. In our study, with the help of training data, we used a logistic regression function to calculate the decision bound for multiple-feature outlier detection. In simulations in which the collected dataset contains a different ratio of outliers, this algorithm can provide a believable decision bound. Moreover, the training of the algorithm is executed on the sink node, whereas outlier detection is executed on the sensor nodes.

Although the support vector machine (SVM)-based method can provide an inspired performance under the aforementioned assumption, this assumption is not reasonable when the deploying environment is very large, as this type of situation is no longer normal. For example, based on their different functions, all rooms in a building have their own sub-environments. Therefore, the normal situation standard of the rooms is different. In this case, preparing training data must involve labeling the situation of considerable data

in many sub-environments. Moreover, the sub-environment situation commonly changes over time. For example, people regularly enter or leave a room, which makes the work of preparing training data more difficult. All of these reasons make preparing training data particularly difficult. As a consequence, unsupervised-learning-based methods, which are free of training data, are sensible for solving such problems.

The first unsupervised machine learning algorithm we propose is based on the mean-shift algorithm, which is a clustering algorithm, and we introduce two new distance and anchor data points in our algorithm for outlier detection. In general, clustering algorithms are usually used when data lack additional information or prior experience (e.g., data point labels in the training data). Clustering algorithms are then used to divide a dataset into clusters, where a cluster is defined as a set of data points having similar properties, such as density, in many data analysis tasks. Moreover, we can create a criterion for outlier or event detection by utilizing the results of clustering. In this study, we tested our algorithm on a real dataset from Intel Lab, and it generated an ideal result. Specifically, it found outliers with a low false positive rate and high recall. For generality, we also tested our method on different synthetic datasets.

A clustering algorithm has a drawback in that the number of calculations is high and clustering accuracy sometimes is poor. To enable the clustering algorithm to be faster and more accurate, we propose a new algorithm called the peak searching algorithm (PSA). Traditional clustering algorithms such as EM and k-means algorithms require extensive iterations to form clusters, which result in slow processing speeds. In addition, clustering results are less accurate because of the manner in which clusters are formed. To address these problems, we first propose PSA, which uses Bayesian optimization to find the peaks of the probability of the dataset to enable clustering algorithms to be faster and more accurate, and we then adapt PSA to include the EM and k-means algorithms (PSEM and PSk-means, respectively). Simulation results show that our proposed PSEM and PSk-means algorithms considerably decreased the number of iterations of clustering to 6.3 times (a reduction of 1.99) and improved clustering accuracy to 1.71 times (an increase of 1.69) as compared to the traditional EM and enhanced version of k-means (k-means++) on both synthetic datasets. Moreover, in a simulation of WSN application for outlier detection, PSEM correctly found the outliers in the real dataset. In addition, it decreased iterations by 1.88 times and had a maximum accuracy gain of 1.29 times.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Wireless sensor networks (WSNs) are widely used in various areas from traditional industrial process monitor systems to individual wearable devices. The use of WSNs involves various challenges, such as scalability, data transfer efficiency, and data processing capability. To clarify the nature of these challenges, I introduce the development of WSNs, beginning with types of data communication, the key weakness of using sensor nodes, and other challenges in dealing with big data. Among those challenges, outliers are the specific focus, motivating proposals for methods of improved outlier detection.

1.1 Background

WSNs merge ideas from various fields, including computer science, ubiquitous computing, communication, and sensing techniques. Therefore, WSNs originally derive from many motivations. Easy deployment is a significant feature, and traditional WSNs are widely used in many areas, e.g., industrial process monitoring, intrusion monitoring, and environmental monitoring. Such WSNs consist of many sensor nodes, which collect information from a target environment and transfer the data to a base node. This is convenient because information is transferred between the sensor nodes and base node by wireless communication, and sensor nodes can thus be deployed anywhere the wireless signal can reach. Compared to wired communication, this reduces the maintenance cost of a WSN, by eliminating the cost of cable maintenance, for example. Recently, with the rapid progress in development techniques for Internet-ready devices and WSNs, more and more everyday devices, such as refrigerators, air-conditioning units, and even lights, are equipped with sensor nodes to form huge WSNs. This has considerably extended the range of applications for WSNs and made them more attractive.

Sensor nodes are important components of WSNs. A sensor node is a kind of ubiquitous device that integrates sensing, control, computation, and wireless communication capabilities. All these capabilities share the same CPU and memory, however, so a sensor

node does not always have sufficient resources to accomplish a complex task by itself. Therefore, the question of how to use resource-limited sensor nodes to cooperatively accomplish a specific, complex task is a major problem in WSNs, as in the example of many sensor nodes distributed in a factory to monitor and control various process parameters in industrial production. Configuring an efficient network to assist every sensor node in accomplishing such tasks is thus a key challenge in improving the capability of WSNs. This involves scalability and efficient data transfer.

On the other hand, the development of micro electro mechanical systems (MEMS) has helped sensor nodes become more powerful and smaller, which also makes them a better technology choice for improving the capability of WSNs. Because of the increasing capability of sensor nodes, WSNs have also been widely used in various applications to improve people's lives, including securing property and ensuring safety. For example, WSNs are used in smart houses and other buildings to monitor and regulate the living environment, providing greater comfort and saving energy.

Together with the wide range of novel WSN applications, however, a significant challenge has risen. Powerful sensor nodes can capture various kinds of large data, e.g., HD video streams and voice data have become very common in current WSNs. Hence, traditional data handling and management cannot keep pace, and centralized data analysis is no longer adequate. As a result, more time and cost is wasted on data transit, since a powerful sink node and higher bandwidth are needed to transfer large amounts of complex data. Recently, many researches ([78], [25], [26] and [126]) try to solve such problems by cloud computation and edge computation.

1.2 Motivation

Although WSNs are widely used today and data collection is easier than before, an important challenge remains, as described above: How can we process WSN data more efficiently and improve the performance of WSN applications? Interestingly, the rich quantity of data collected by WSNs facilitates application of new techniques to address this challenge. For example, through machine learning, we can extract experience from this abundant data and use it to improve performance.

Automatic improvement in the performance of a program from previous experience is a remarkable capability of machine learning. Moreover, learning from previous experience can provide a specific solution for a particular user's behavior, which is difficult without using machine learning. For example, sensor nodes deployed in vehicle systems are designed to assist the driver. In this case, machine learning from historical information can provide a model for stopping a car when its sensors detect danger.

There is much research on machine learning methods for automatic data processing

in many different applications or types of WSNs. Examples include environmental monitoring, industrial safety and control, health monitoring, and disaster prediction ([4], [52], [87], [117]). There are also some concrete machine learning methods used in WSNs, such as Shareef, et al. [103] using neural networks to localize objects in WSNs. Bahrepour, et al. [16] proposed a fire detection method based on a neural network, and Moustapha, et al. in[84] used a neural network for faulty data detection. In addition to neural networks, other types of machine learning methods have also been applied in WSNs; for example, Bahrepour, et al. in [17] used a decision tree for event detection.

Although modern WSNs enable easy data collection, they also increase the probability of encountering outliers, or abnormal data points that obviously differ greatly from other data. Outliers mixed together with normal sensor data obviously reduce the performance of WSNs. For example, an outlier might be considered as an event that triggers an alarm.

Outliers are an important concept in other areas such as the field of statistics, which offers two famous definitions. The first one, by Hawkins [54], states that "an outlier is an observation, which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism." According to the second definition, by Barnett and Lewis [1], "an outlier is an observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data." In addition, there are several other definitions [3], which depend on the specific technique used for outlier detection.

We must pay attentions to outliers because they are very common in WSNs. Outliers commonly appear in WSN data for two reasons: (i) Sensor nodes are easily fallible, as WSNs are often deployed in harsh environments ([39], [19], [90], [9]); for example, sensors might operate at extreme temperature or humidity and thus be susceptible to malfunction. [91] and [50] show that harsh environments affect a WSN's capability in that the collected dataset contains outliers. (ii) Wireless signal noise and malicious attacks also lead to outliers [63], [53] in WSNs.

We also have to pay attention to outliers because they obviously reduce the capability of WSNs. For example, some WSN applications involve comparing the collected dataset with the normal conditions of some particular scenario in order to predict or prevent an event. If the dataset contains outliers, however, the event may not be correctly predicted or prevented. Here, we briefly summarize several real-world applications of WSNs that highlight the importance of outlier detection.

- Environmental monitoring: WSNs are deployed in harsh environments to monitor the natural environment, e.g., they are used in monitoring volcanoes to prevent disasters or deployed in forests to monitor forest fires. In such circumstances, however, high temperature and humidity can obviously affect the performance of sensor nodes, which may produce outliers. These outliers can trigger false alarms.

- Habitat monitoring: Taking advantage of their mobility, sensor nodes are deployed

on endangered animals to collect information about their environments and behaviors. Outliers can prevent correct analysis of such data.

- Health and medical monitoring: Patients are equipped with sensor nodes to monitor their health status, e.g., sensor nodes are used to collect heart data, which can be used to cure heart disease. Outlier detection can help distinguish whether an abnormal record is a sign of potential disease or an outlier.

- Industrial production: WSNs are used in industrial production for monitoring and controlling various process parameters. Outlier detection can establish whether an abnormal data point indicates possible malfunction or a missed operation, or is just an outlier.

- Parcel tracking: To quickly track the location of shipped goods in real time, they are equipped with sensor nodes by online retailers such as Taobao, Jingdong, and Amazon. Outlier detection can enhance the accuracy of location tracking.

- Surveillance monitoring: Sensor nodes are deployed in sensitive areas in terms of security, e.g., airports, train stations, and public squares. For example, these sensor nodes can be used to detect gases from explosive devices to improve public safety. Outlier detection in such cases can filter out erroneous information that could lead to a missed alert or false alarm.

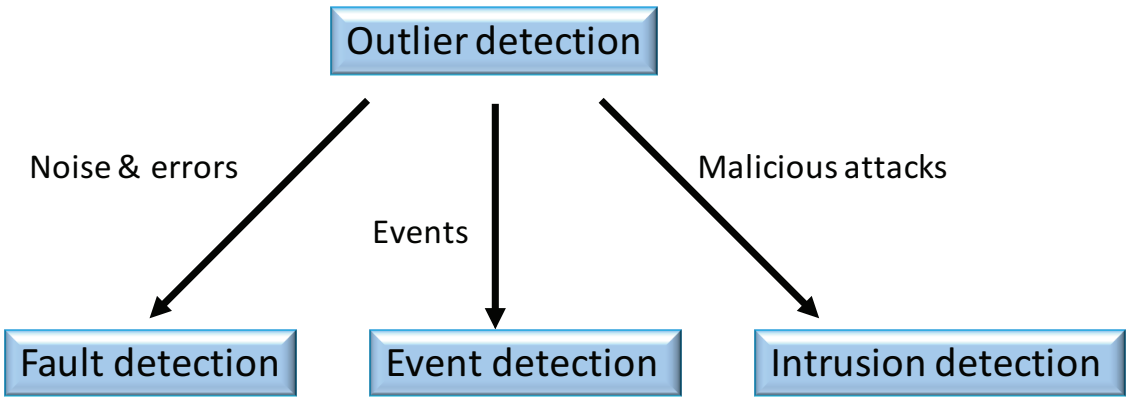

Figure 1.1: Relationships between outlier detection and various applications

In addition, outlier detection can be used for other purposes, such as event detection ([8], [5] [6]), fault detection ([41], [72]), and detection of malware and network intrusions ([111], [35], [59]). Figure 1.1 illustrates the relationships of different applications with outlier detection [127].

In summary, then, outlier detection is very important for guaranteeing the effectiveness of WSN applications and can be used in many other applications, as well.

1.3 Objective

The goal here is a method of automatically detecting outliers in a collected dataset. Specifically, I apply machine learning (ML) methods for outlier detection in WSNs. ML contains three basic methods that are supervised leaning, unsupervised learning and reinforcement learning. In WSNs for outlier detection, the supervised learning and unsupervised learning are commonly used. ML algorithms are based on statistics and can be used to extract features from a dataset, such as face features from an image. ML algorithms are not restricted by fixed thresholds or parameters. Moreover, they can analyze data properties to find differences between a given data value and its historical norm and then exploit the differences for some concrete task, such as outlier detection.

On the other hand, we should consider using a distributed way or a centralized way when we use ML-based outliers detection methods in WSNs. The method based on supervised learning is much easy for distributed way. For example, a powerful sink node can learn a model from the training dataset, and then it delivers the model to every sensor node who use the model for outliers detection.

Much research on automatic outlier detection in WSNs requires a previous sensor dataset for comparison. For example, methods based on supervised learning use a historical collected dataset to estimate a model providing an approximation for the underlying distribution that generated the dataset. These methods then detect outliers by using the estimated model. The estimated model may become invalid, however, when the environment changes, because the underlying distribution also changes with the environment. Supervised learning have a similar weakness. We designed a preliminary experiment (Chapter 3) that we distributed an algorithm based on supervised learning throughout the WSN environment. In this experiment, we found several weak points when distributed the algorithm based on supervised. For example, they require training data in which every data point has previously been labeled as normal or outlier in order to estimate a model. Such labels in training data may also become invalid when the environment changes. Moreover, preparing training data is very time-consuming and expensive.

Therefore, automatic outlier detection in WSNs requires a method that can endure environmental changes. Methods based on unsupervised learning use only raw data and can estimate models without requiring a previously collected sensing dataset or prepared training dataset. Hence, unsupervised learning is more adaptable to environmental changes.

Another challenge of automatic processing for outlier detection is that the collected dataset can be considered to be generated by a Gaussian mixture model (GMM), consisting of an unknown number of different Gaussian distributions. In addition, the collected data points do not have any labels representing the nature of their information. For example, consider a sensor node transmitting temperature data from the monitored environment to a sink node. In this case, the sink node cannot judge the state of the environment, because the

data points provide no additional information, such as their status as normal or abnormal. Therefore, we need a method of detecting the status of collected data points.

The first method (Chapter 4) applies clustering to the collected WSN dataset and uses the clustering result to detect outliers. This method uses the mean shift algorithm to cluster the dataset because it can not only cluster the dataset but also find the mode of each cluster, as well. Then, the mode of each cluster and the median value of the sensing dataset can be used to detect which clusters are outliers. To the best of my knowledge, this work is the first to use the mean shift algorithm to detect outliers in WSN data.

Since each Gaussian distribution in a GMM corresponds to a cluster, if we could know the number of distributions, a clustering algorithm could appropriately divide a dataset into different clusters. Moreover, with a correct clustering result, we could gather additional information on data points in the same cluster according to their similar behaviors. For example, normal data points might belong to a cluster, while outliers do not belong.

Therefore, the second method covered in this thesis (Chapter 5) involves an algorithm to improve the capability of clustering algorithms. Given a collected dataset, the proposed peak searching algorithm (PSA) is a Bayesian optimization strategy to search for the data point with the maximum probability value in the dataset, called the peak of the dataset. For example, the peak of a Gaussian distribution is the point corresponding to the mean, and a GMM has several peaks. Thus, we can obtain the number of Gaussian distributions in a GMM.

Chapter 2 introduces various related works on outlier detection and assumptions used in outlier detection. It also examines the development of WSNs using ML methods. The first novel approach in this thesis, described in Chapter 3, is a preliminary experiment of outlier detection method based on supervised learning and distributed in a WSN. Simulation results with that method indicated the need for an outlier detection method based on unsupervised learning (i.e., a clustering method), given in Chapter 4. The benefit of this method is that it does not require training data. In Chapter 5, to accelerate the processing of clustering, Bayesian optimization is applied to obtain the peaks of a GMM. These peaks enable a clustering method to quickly cluster a WSN dataset. Finally, Chapter 6 presents the conclusion of this thesis. All these algorithm we proposed in this thesis are related to outlier detection, although some algorithm may used for other purposes.

Chapter 2

An Overview of Outlier Detection Methods and Machine Learning Methods Used in Wireless Sensor Networks (WSNs)

Machine learning (ML) methods can be categorized into supervised learning, unsupervised learning, and reinforcement learning, depending on whether they require training data. When a training dataset is used, each data instance has a label to facilitate the purpose of the learning algorithm. For example, if a dataset containing images of dogs and cats and the purpose of the learning algorithm is to recognize cats, the images will have labels indicating whether they show cats or not.

Researchers have also applied ML algorithms for wireless sensor networks (WSNs). The review by Abu, et al. [10] classifies ML algorithms for WSNs into two categories. One category addresses functional issues such as routing, localization, clustering, data aggregation query processing, and medium access control. The other category addresses non-functional issues such as quality of service, security, and data integrity. According to their research, ML algorithms can solve many problems and have great potential application in WSNs.

The last part of this chapter reviews related research on using ML methods in WSNs, and on outlier detection methods in WSNs, most of which are based on machine learning. The challenges and assumptions involved in these methods are also considered. Finally, the chapter reviews related work on clustering for outlier detection.

2.1 A Brief Introduction to Machine Learning (ML)

An ML algorithm is a computer program that can improve its performance from previous experience. Such algorithms are widely used in computer science for solving automatic problems. A classic definition of machine learning was given by Tom Mitchell [81]:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."

2.1.1 Supervised Learning

Supervised learning is a family of ML methods, using the general idea shown in Fig. 2.1. Appropriate labels are applied in a working dataset, and the labeled portion is divided into three parts: training data, test data, and validation data. The learning model is applied with the training dataset. The goal is to select a hypothesis from a hypothesis space containing every possible model that can represent the relationship between the data and the labels. Because this space is infinitely large, the training data helps find a probability correct hypothesis. Finally, the hypothesis is applied with an unlabeled dataset to provide labeled data instances. For example, given pictures of different fruits, the label of each picture would indicate the kind of fruit in it.

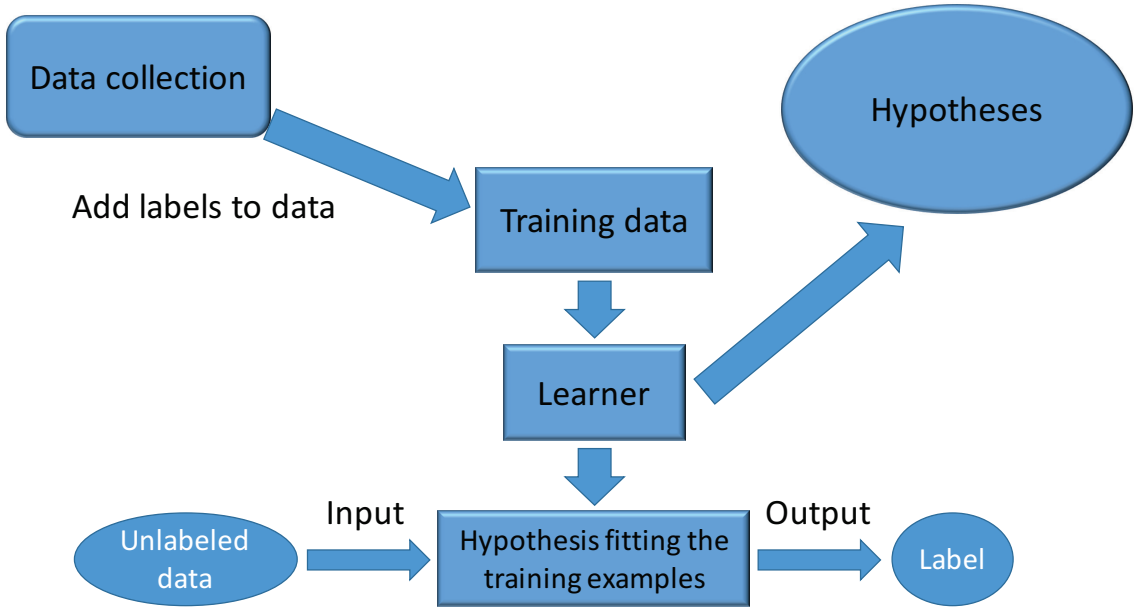

Figure 2.1: Overview of supervised learning

Supervised Learning Algorithms in WSNs

Supervised learning provides many algorithms for solving several issues in WSNs, such as medium access control [18], [66], [106], localization and object targeting [69], [82], [76], intrusion detection and security [68], [61], [28], [64], query processing and event detection [104], [115], [62], [124], and data integrity, quality of service (QoS), and fault detection [96], [108], [85]. Next, some well-known supervised learning algorithms are introduced.

Bayesian Algorithms

Bayesian theory is a very simple, powerful tool in ML because hypotheses can be assigned weights based on prior probability. First, Bayesian methods calculate explicit probabilities for hypotheses. For example, Michie, et al. [73] compared decision tree and neural network methods with a naive Bayesian classifier found they have some similar features. Second, Bayesian methods can provide an explicit perspective for understanding other ML algorithms with implicit probabilities. Bishop, et al. in [24] provided a very explicit explanation from a Bayesian perspective for many ML algorithms. Third, compared to other ML algorithms, if a Bayesian algorithm fully uses the prior experience, then it require less data [27]. The features of Bayesian methods as described in [81] may be summarized as follows.

- Each training instance can incrementally improve or weaken the estimated probability indicating the correctness of a hypothesis. This is an advantage over other methods that have to process all data at the same time. Therefore, Bayesian methods can provide real-time data processing.
- The prior knowledge of a collected dataset is fully utilized, because each hypothesis has a prior weight. Via calculation, we can update the weight of each hypothesis by using posterior probability.
- The label of an unlabeled instance can be predicted simply by maximum likelihood estimation (MLE).
- Even with a high number of dimensions, Bayesian approaches can provided simple, quick prediction.
- Bayesian methods can provide probabilistic prediction.

K-Nearest Neighbors (k-NN) Algorithm

This supervised learning algorithm classifies a data instance (called a query point) according to the labels of neighboring data instances. A positive integer k indicates the number

of neighbors. For example, if a feature of a data instance is missing, we can predict the missing value from the similar features of the k nearest data instances. The usual measure to determine the nearest data neighbors is the Euclidean distance. In a low-dimensional space, e.g., 2-D or 3-D, the calculation is not very complex. Because of the curse of dimensionality, however, the calculation is very complex and the results are not very reliable in a high-dimensional space. Therefore, the k-NN algorithm can only be used in a WSN in which sensor nodes capture few features for feature selection [23]. In [23], results were shown for a high-dimensional space (i.e., with more than $10 - 15$ features). Because the distance is invariant, the results had low accuracy. Other studies [115], [62] have shown that the k-NN algorithm is acceptable in a query processing subsystem.

Decision Tree

In decision tree learning, data features are compared with decision conditions in order to select a specific category. An example of using decision trees for classifying data is discussed in [15]. A decision tree can provide both quantitative (prediction) and qualitative (classification) results. Many decision-tree-based applications in WSNs are used to solve different design challenges. For example, a decision tree can be used to verify link reliability in a WSN. The drawback of decision tree learning is that it is only suitable for processing datasets in which every data instance is linearly separable, as decision tree optimization is NP-complete [100].

Support Vector Machine (SVM)

Support vector machines (SVMs) provide very powerful machine learning algorithms. Unlike regression, an SVM determines a separation hyperplane with a margin so as to maximize the gap between different classes, as illustrated in Fig. 2.2. The SVM divides the dataset into different parts according to data instances called support vectors. The class of a new data instance is determined by the area in which it falls. An SVM in conjunction with a kernel function, which projects data to a higher-dimensional space, can efficiently handle high-dimensional data. An example of an SVM-based application used in WSNs is detection of malicious behavior by nodes. SVMs also provide an alternative to neural networks [15] for solving some non-linear problems. WSN security applications based on SVMs are discussed in [64], [96], [122], [33], and [128]. Sensor node localization is discussed in[67], [113], and [120]. A more detailed discussion of SVMs is given in [109].

Neural Networks

A neural network consists of a large number of layers linked together. Each layer contains a number of neurons, and an "active function" is equipped inside every neuron. Data are

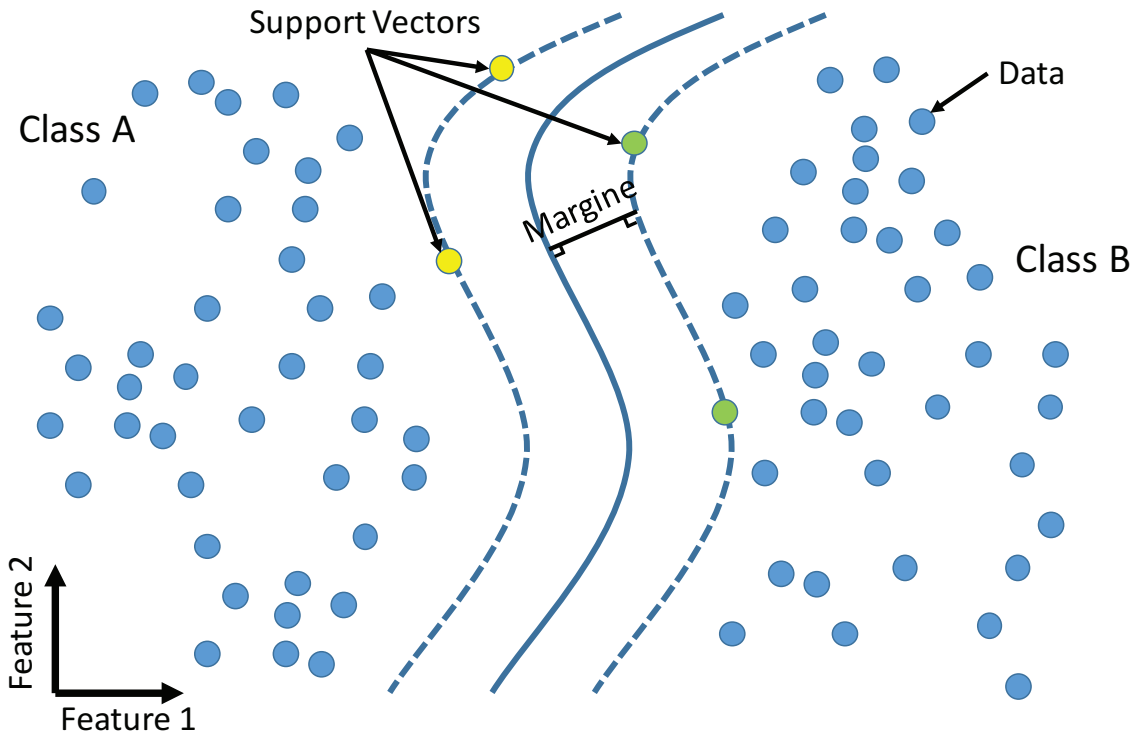

Figure 2.2: Illustration of a support vector machine (SVM)

input to an "input layer," the first layer, and results are output from an "output layer," the last layer. Neural networks can also be used to solve nonlinear problems [21]. In WSNs, however, it is difficult to deploy a neural network in a distributed fashion, because updating each decision unit requires a backpropagation neural network (BPNN) [55], an algorithm based on the chain rule. Moreover, the BPNN is so complex that a single sensor node might not able to execute it because of data and resource limitations. Therefore, neural networks are widely used in WSNs in a centralized way, so that they can learn from multiple inputs at once [74]. Figure 2.3 shows an example of a neural network. A localization application typically uses information such as the received signal strength indicator (RSSI), time of arrival (TOA), and time difference of arrival (TDOA). After training the neural network, it can predict the locations of sensor nodes. Other applications using neural networks include self-organizing maps and learning vector quantization [65]. In high-dimensional space, neural networks have an important application for tuning and dimensionality reduction in big data analysis [56].

Supervised-Learning-Based Outlier Detection in WSNs

ML algorithms can be applied in two classes of problems: quantitative problems such as using historical temperature to predict future temperature, and qualitative problems such as

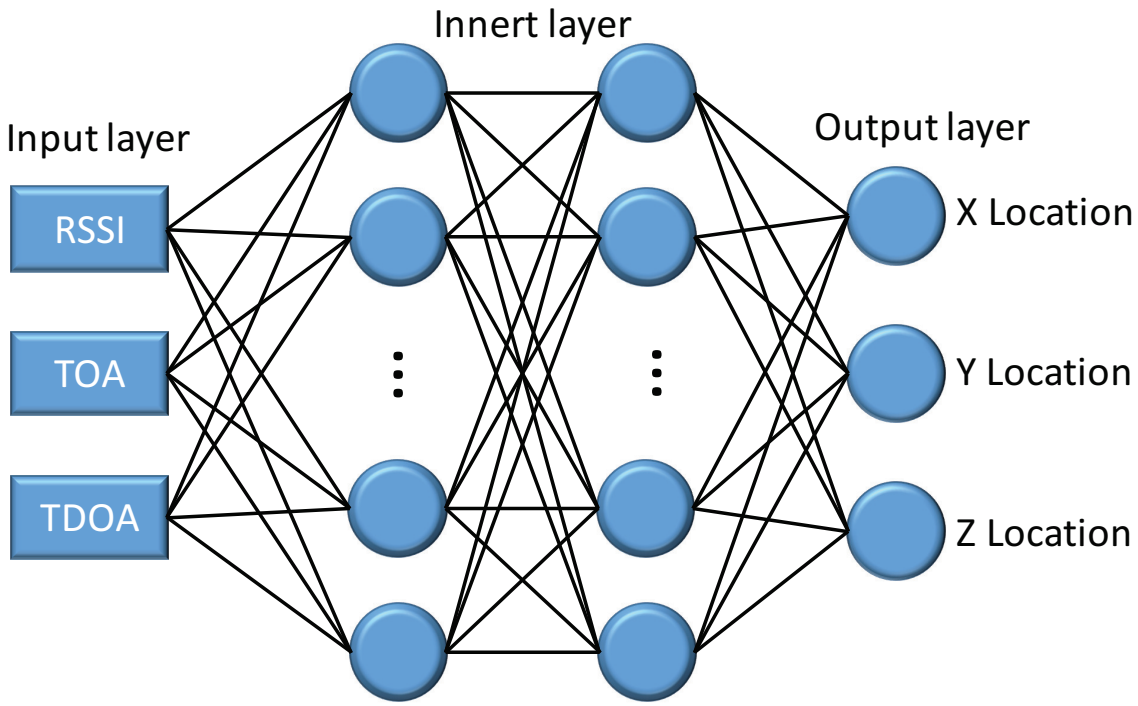

Figure 2.3: Example of a neural network

using credit card activity information to detect whether a credit card has been stolen. Next, some methods based on supervised learning are considered for outlier detection in WSNs.

Many such methods have been applied. Miao, et al. [80] proposed a method using sensor node status data (e.g., radio-on time and number of transmitted packets) to detect potential function faults. Chen, et al. [32] detected faulty data by comparing measured values with values from neighboring sensor nodes. Other faulty data detection methods are proposed in [125], [71], [51], and [79]. Most such methods assume that sensor nodes are deployed in a stable environment.

Another method based on supervised learning was presented by Rajasegarar, et al. [97], and it provides anomaly detection in WSNs by using a one-class quarter-sphere SVM. They use training data to fit a hyper-surface, which is then used to detect outliers.

Some previous works have applied very complex ML methods in WSNs, such as neural networks and decision trees. The learning step is very complex in these ML algorithms, and it is difficult to update single sensor nodes online. Because the learning step and updating consume much battery power, these algorithms cannot be distributed in WSNs. They are conceptually similar to the LR algorithm [94], however, which is not very complex and relatively easy to distribute. The LR algorithm has not previously been applied for faulty data detection in WSNs, so this is the focus of my research.

2.1.2 Unsupervised Learning

As described previously, unsupervised learning does not require training data. Instead, methods based on unsupervised learning extract underlaying distributions from a given dataset, which are then assumed to generate the given dataset. This is similar to the supervised learning depicted in Fig. 2.1, but without the training data. Put another way, the main focus of unsupervised learning is to directly infer the properties of a dataset, including the probability density, without the help of a supervisor (for labeling) providing correct answers or a degree of error for each data instance [47]. Often, the dimension of a data instance is higher in unsupervised learning than in supervised learning, making computation more complex. This section introduces unsupervised leaning for clustering problems. A clustering algorithm is an appropriate method for automatically processing a dataset. The techniques used in clustering algorithm have been previously surveyed from the viewpoint of data mining [31] and [102].

In particular, statistical and partitioning-based techniques are unsupervised learning algorithms widely used in WSNs. Zhang, et al. [121] presented an online local outlier detection method based on an unsupervised, centered quarter-sphere SVM for WSN environmental monitoring applications. Other techniques based on unsupervised learning include approaches based on k-means clustering [95] and principal component analysis (PCA) [75].

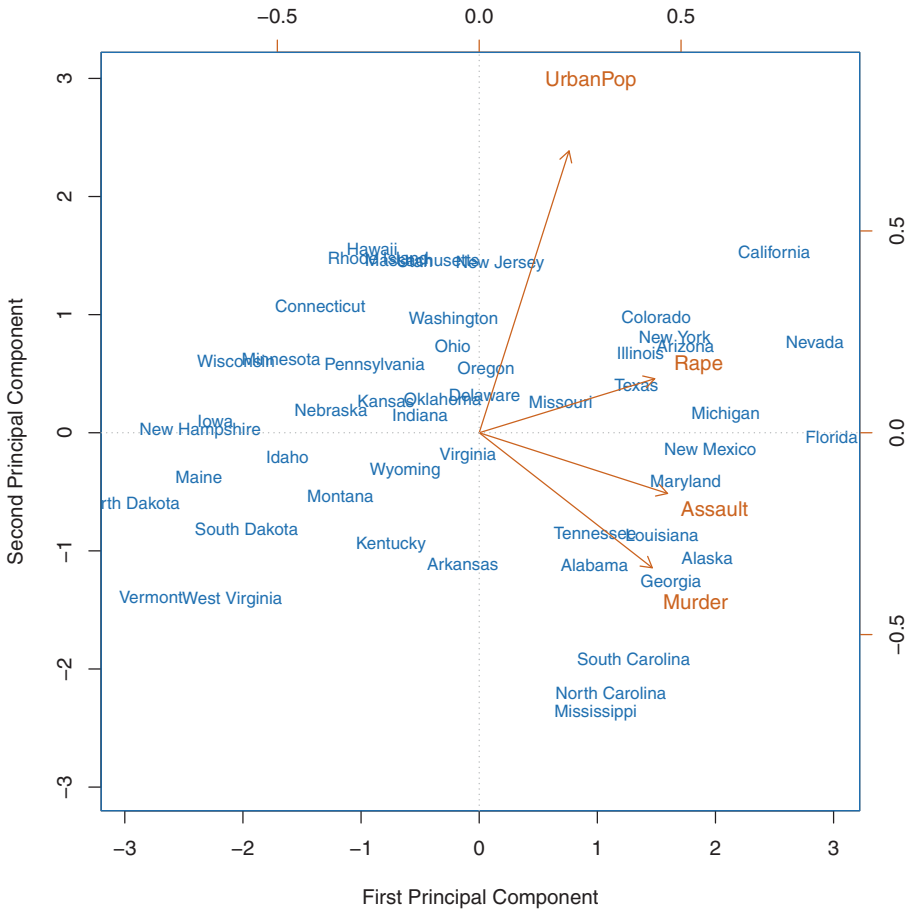

Figure 2.4: Example of PCA[60]

Regarding the latter, PCA is a popular approach for deriving a low-dimensional set of features from a large set of variables [60]. It is sometimes used as a dimension reduction method for regression problems. It can also find latent relationships among different features, however, and it is thus used as a feature extraction method. Figure 2.4 shows an example using a dataset called $USArrests$ to represent the relationships between three types of crimes and population. The original dataset has 50 features, but only four are of interest for this analysis: $Assault$, $Murder$, $Rape$, and $UrbanPop$. The results are apparent in the figure. The locations of the state names in blue indicate scores for the first two principal components, while the orange arrows indicate loading vectors in terms of the first two principal components [60].

Outlier Detection Based on Unsupervised Learning

There have been many surveys of outliers and abnormal data detection, by Beckman, et al. [20], Hodge, et al. [57], Chandola, et al. [31], and Xie, et al. [119]. In these surveys, unsupervised outlier detection methods can be divided into statistical and non-statistical

methods. The statistical methods for outlier detection are similar to supervised learning in ML, because both approaches estimate a model from a given dataset. In contrast, non-statistical methods do not have to estimate a model. Therefore, related works can indeed be classified as statistical or non-statistical methods, as discussed below.

Statistical Methods The goal of statistical methods is to build a probability model, which is assumed to generate the given dataset. According to the concrete methods for generating such a model, the statistical methods can be divided into parametric and non-parametric model techniques.

- **Parametric Model Techniques**: All model-based methods assume that a given dataset is generated by one or more statistical models, such as a Gaussian model. Parametric model techniques thus focus on estimating a model and assuming it generates measured data instances. If a data instance has a low probability by the estimated model, then it is considered an outlier.

  The following three parametric model techniques use statistics to estimate the model. Usually, a Gaussian model is chosen as the default because it has a well-defined property from the central limit theorem. Wu, et al. [118] presented a localized algorithm to identify outlying sensors and events in sensor networks. They use the spatial relationships of neighboring sensor nodes' readings to detect the outliers. Bettencourt, et al. [22] also proposed a local method for detecting outliers in WSNs, which uses the spatio-temporal correlation of measurements between a sensor and its neighbors to build a model. Palpanas, et al. [88] proposed using kernel density estimators to estimate a sensing dataset model on the basis of distance for online deviation detection in streaming data. Markus, et al. [29] estimated $priors$ of the assumed statistical model, such as the mean, median and variance, from the collected dataset in order to fit statistical models.

- **Non-Parametric Model Techniques**: Non-parametric model techniques require fewer assumptions because they do not estimate a prior, although they do make certain assumptions such as smoothness of the probability density. Typical non-parametric methods include those based on histograms, as in [44] and [45], and kernel density estimators, as in [101], [89], and [70]. These techniques also use relationships between data instances, such as the distances between them, and the density of the dataset.

  Subramaniam, et al. [110] enhanced the work of Palpanas, et al. [88] to detect outliers online by approximating sensing data in a sliding window and using a local metric-based algorithm to detect outliers in datasets that are hard to distinguish by distance. Sheng, et al. [107] proposed a non-parametric method using histogram

information to detect outliers in WSNs. The most significant contribution of their method is that the use of histogram information reduces the communication cost.

In summary, the statistical methods provide two main benefits: (i) They provide a probability criterion to determine whether every data instance is an outlier. (ii) They do not require any extra information, such as labels of data instances to indicate their status. On the other hand, statistical techniques cannot be deployed in a distributed manner, because they require many data instances to estimate the mean and variance. Thus, statistical methods are deployed in a centralized manner in WSNs. Algorithms based on these techniques provide a probability model. A data point that does not belong to the model is implied to have low probability and thus considered an outlier.

1.1.1.2.

Non-Statistical Methods Partition-based techniques belong to the category of non-statistical methods. Assuming a dataset contains several partitions, partition-based techniques divide the dataset into a number of initial partitions, where each partition represents a cluster and contains at least one data instance. These techniques then use the probability density or Euclidean distance of every initial partition to transform them into stable final partitions.

The $k\text{-}means$ approach is a well-known, widely used algorithm deriving from partition-based techniques. Another algorithm using partition-based techniques is $k\text{-}medoids$, in which data instances near the center are incorporated into the same partition. Other algorithms using partition-based techniques are $k\text{-}modes$, $Partitioning\ Around\ Medoids$ (PAM), $Clustering\ Large\ Applications\ (CLARA)$, and $Clustering\ Large\ Applications$ $Based\ Upon\ Randomized\ Search\ (CLARANS)$. These partition-based techniques are effective when a dataset is of small or medium size. Thus, they can be used for WSNs in a distributed way. On the other hand, since they cannot provide a probability model, they cannot give a probability criterion for determining whether a data instance is an outlier. For example, in the case of event detection, these techniques cannot filter the outliers in a cluster corresponding to an event.

In conclusion, both statistical and non-statistical methods of outlier detection have some weaknesses regarding their use in WSNs. Statistical techniques require sufficient data to estimate a model and are thus difficult to use in a distributed way. On the other hand, although partition-based techniques can be executed in a distributed way in WSNs, they cannot provide a probability criterion for outlier detection. Moreover, both techniques require much computing power, which would reduce the life of low-powered sensor nodes. The $PSEM$ algorithm proposed in Chapter 5 for outlier detection does provide a probability criterion, and its capability was tested through simulation on a real dataset.

2.1.3 Reinforcement Learning in WSNs

Reinforcement learning provides a novel solution for automatically improving the capability of an agent when it interacts with its environment. The agent is similar to the learning model in supervised learning. During the process of training an agent, if its action approaches a designed goal, it receives a reward; otherwise, there is no reward. This training process makes the agent approach the designed goal incrementally. A well-known algorithm of this type is Q-learning [114], as illustrated in Fig. 2.5. The agent repeatedly updates its rewards according to its chosen action. The reward is calculated by the following equation:

$$Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) + \gamma(r(s_t, a_t) - Q(s_t, a_t)), \tag{2.1}$$

where $Q(s_{t+1}, a_{t+1})$ is the reward based on action a_{t+1} at environment status s_{t+1}, $r(s_t, a_t)$ is the current reward, and γ is the training rate, which controls how fast the agent learns from its environment.

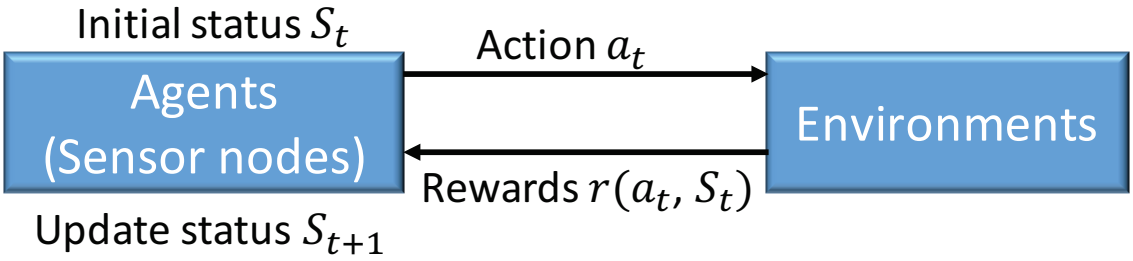

Figure 2.5: Illustration of reinforcement learning

As mentioned above, reinforcement learning requires agents to interact with their environment. Therefore, in designing a WSN, sensor nodes can naturally be considered agents because they can collect measurement data from the environment or WSN. Moreover, by using the collected information, sensor nodes can apply different strategies, as in WSN routing problems. Research on routing problems in WSNs is covered in [112], [42], [46], and [13].

2.2 ML in WSNs

WSNs have two major issues. The first issue is the "dynamic problem," which refers to both an internal dynamic property of WSNs and an extremal dynamic property. The second issue is data processing.

- **Inner dynamic property**: As mentioned before, WSNs usually adopt an ad-hoc network infrastructure. In this type of network configuration, sensor nodes must automatically and rapidly construct clusters, where a cluster is a small network configuration composed of several sensor nodes. Sensor nodes in the same cluster are not selected randomly but rather by some particular process. The process of clustering is performed according to certain measurements related to wireless signal characteristics, such as time of arrival, time difference of arrival, signal strength, angle of arrival, and received signal strength. These measurements change by time, in particular, as when a sensor node has low battery power. In this case, it will adopt a power strategy of reducing its wireless signal strength to keep its kernel function working. This phenomenon may change the cluster configuration of a WSN.

- **External dynamic property**: This property results from changes in the WSN deployment environment. In addition, it is necessary to consider temporal and spatial factors to explain such environmental changes. As an analogy, we can compare a fact there might be heavy traffic during the day but fewer cars on the road at night. The spatial factors are related to mobile sensor nodes, like those on an airplane, whose measurements depend on the airplane's location.

Regarding the second issue of data processing, collecting data from the environment is the major function of a WSN. The increasing scale of WSNs results in so much data, however, that it cannot all be processed promptly. Hence, efficient, correct processing of collected data becomes an important issue.

In conclusion, by considering the definition of ML, we can see that appropriate ML methods can solve such dynamic challenges. In this thesis, in particular, I focus on the external dynamic property.

2.2.1 Summary of Advantages of Using ML in WSNs

- Greater suitability for monitoring dynamic environments: For example, cargo ships on the Pacific Ocean encounter frequent changes in wind direction wind and tides. WSNs on cargo ships assist with navigation and enable navigation strategies to change with the dynamic environment. Machine-learning-based WSNs can provide a solution.

- Rapid computation: Over the years, the field of ML has gradually accumulated a series of mathematical optimization models for complicated environments. For example, by the central limit theorem we can appropriately assume that an environment follows a Gaussian mixture model (GMM), which can be solved by the EM algorithm [38], a well-known ML algorithm.

- Extraction of unexpected relationships among features of a dataset: When data has many features, it is difficult to recognize relationships among them by observation. For example, most data collected by WSNs includes spatial and temporal information. Clearly, establishing the relationships among on spatial and temporal features is very important for improving the performance of WSNs. In ML methods for dimensional reduction, such as PCA [116] and t-SNE [77], can solve this problem.

- Increased choice for automation and novel application: ML can facilitate greater automation in a large-scale WSN. For example, the Internet of things and machine-to-machine technologies can provide more intelligent applications, and such applications require less human intervention [12], [99].

- Exploration of the unknown: WSNs can be deployed in very difficult locations, such as volcanoes, rainforest, marshes, and undersea. WSNs can enable development of exploratory applications for early detection of volcanic eruptions, forest fires, and tsunamis [43], as well as detection of anomalies and unexpected behavior patterns.

2.2.2 Challenges of Using ML in WSNs

The previous section introduced several advantages of applying ML in WSNs, which have excited many researchers. Several challenges, however, remain.

- Resource constraints: The major weakness of sensor nodes is their limited battery, memory, computational capacity, and communication bandwidth. In contrast, most ML algorithms are very complex and require much data to support model building, putting high demand on sensor resources. Therefore, an ML algorithm cannot be simply and directly used in a WSN.

- High communication cost: Most ML algorithms are not designed to work in a distributed fashion but on a given dataset. Distributed implementation increases the volume of radio communication required for collecting data from every sensor node, and this is the source of most battery consumption in sensor nodes. Thus, the communication cost is usually several times higher than the computation cost [11].

- Real-time data processing: Data in WSNs can be considered as streams between sensor nodes and a base station. The environment of each sensor node changes with time, and thus the streams may also change. On the other hand, ML algorithms are centralized and process all data at once. Moreover, research [49] shows that direct computation of probabilities is difficult.

- Dynamic network topology, mobility, and heterogeneity of sensors: Because a mobile sensor node can change its location at any time, the network has to be able to

change its configuration. Moreover, if a sensor fails in a node equipped with many different types of sensors, an ML algorithm cannot give a correct result.

- Increasing scale of WSNs: Developments in related techniques have made large-scale WSNs possible, with hundreds or even thousands of sensor nodes. A more appropriate protocol for managing these sensor nodes is necessary. This protocol should not only control how data is transmitted at the physical level but also support data processing strategies, such as the use of ML algorithms.

In conclusion, the question of how to efficiently use resources to prolong the lifetime of a WSN is an issue in adopting ML methods, because they commonly require more memory and a more powerful CPU to execute complex algorithms. Sensor nodes often have low memory and a low-capability CPU, making it difficult to process as much data as possible by an ML algorithm and while maintaining high accuracy result.

2.3 Outlier Detection Methods Based on ML Methods in WSNs

Many outlier detection methods based on ML were introduced in the above sections. I emphasize again that outlier detection methods based on ML must pay attention to the external dynamic property. Moreover, because the concrete circumstances of different environments vary greatly, outlier detection methods must be based on different assumptions. Using an appropriate assumption can sometimes simplify the outlier detection problem in a concrete environment, so this section introduces some important assumptions.

2.3.1 Assumptions for Outlier Detection Methods in WSNs

As in Chapter 1, I again introduce the definition of an outlier, because of its importance. Although there are many definitions, two are very well-known. The first is by Hawkins [54]:

Definition 1. *An outlier is an observation, which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism.*

The second is by Barnett and Lewis [1]:

Definition 2. *An outlier is an observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data.*

These definitions are very similar, and this thesis uses the following definition derived from them:

Definition 3. *An outlier is a kind of data instance exhibiting different behaviors from most other data instances.*

The proposed methods for outlier detection are based on this definition.

Next, we show the development of assumptions for outlier detection in WSNs. I describe the differences between them and use those differences to explain how each assumption should be used under given conditions.

Assumption 1. *Normal data instances in a dataset belong to one cluster, while data instances from another cluster are considered outliers.*

This assumption is quite strict, and this can only be used in very less cases. For example, when a WSN is used to monitor a static environment. Such an environment has similar conditions everywhere, e.g., the distribution of temperature is similar everywhere. However, if the environment contains multi status, e.g., a large room where the distributions of temperature are different, this assumption cannot be used.

Assumption 2. *Using Euclidean distance normal data instances are close to the center of a cluster, while outliers are far from any such center.*

This is an upgraded version of **Assumption 1**. It applies in the case of a monitored environment containing a variety of different conditions, e.g., the distributions of temperature in the kitchen and living room of a home are not the same. This assumption has one degree of freedom, namely, the distance from a data instance to the center of a cluster. This degree of freedom is important because it is used to determine which data instance is an outlier. Therefore, when we do not have enough prior experience of the environment to specify this degree of freedom, more errors will occur during outlier detection.

Assumption 3. *If a cluster contains numerous data instances and has high density, then it is a normal cluster; otherwise, it is a outlier cluster.*

As mentioned above, **Assumption 2** is difficult to apply without sufficient prior experience of the environment. In contrast, **Assumption 3** does not require such prior experience, because it is based on the central limit theorem and the law of large numbers. The central limit theorem says that the distribution of independent random variables is a normal distribution, even if the original variables themselves are not normally distributed. The law of large numbers states that when the same experiment is repeated many times, the results of these experiments converge to the same value. Therefore, we can say that when data instances share many similar features, those data instances represent a normal state and follow a normal distribution.

These assumptions obviously become more and more complex in complex situations of outlier detection. Such complex assumptions also make implementation more complex and

thus very difficult to execute with scarce sensor node resources. In conclusion, deciding which assumption to adopt depends not only on the concrete environment status but also on the type of sensor node, e.g., a powerful sensor node can execute a more complex method.

2.3.2 Challenges of Outlier Detection in WSNs

Although the above assumptions can simplify outlier detection methods, several challenges must still be addressed. These challenges are summarized below.

- It is difficult to define an overall normal status encompassing every possible normal status for data instances, because the environment of a WNS is dynamic, with many possible statuses. This means that the boundary between normal and outlier data is usually unclear.
- Outliers generated by a malicious attack are likely to be very similar to normal data instances. Moreover, sensor nodes can easily be attacked because they are not usually monitored. Therefore, detecting outliers due to malicious attacks is a challenge.
- The normal status of a monitored environment changes with time. Outlier detection methods in WSNs should be capable of detecting outliers in a dynamic environment.
- Data instances gathered by sensor nodes contain noise. For example, a temperature sensor might be shaded by leaves, making the collected data neither an outlier nor the correct temperature.

These challenges cannot be solved synchronously because they do not often occur. In designing outlier detection methods for WSNs, however, we should consider solutions for these challenges.

2.4 Related Works on Outlier Detection

This section describes techniques used in clustering algorithms for automatically dividing a given dataset into different clusters. There are two main types of clustering approaches. The first is based on parametric techniques, in which the parameters of a statistical model must be calculated. The EM algorithm is one such parametric technique. The second type of clustering approach uses non-parametric techniques, which can cluster a dataset without calculating the parameters of a statistical model. Typical non-parametric techniques include the k-$means$ and k-$means$++ algorithms.

The following subsections describe these two types of clustering approaches. Because outlier detection is critical, a later subsection describes relevant outlier detection approaches.

2.4.1 Parametric Techniques

Parametric techniques assume that a dataset is generated from several parametric models, such as GMMs. The clustering process is conducted by calculating the parameters of each Gaussian model and assuming that data instances in the same cluster can be represented by the same Gaussian model. Usually, a Gaussian model is chosen as the default model, because it conforms to the central limit theorem. Researchers [118], [29] have used parametric techniques in which detailed $a\ priori$ estimates of statistical parameters are calculated for the assumed statistical model (for example, the mean, median, and variance). This allows them to fit statistical models.

EM [38] is a well-known, widely used algorithm using parametric techniques to cluster datasets. The EM algorithm first calculates responses with respect to given parameters, such as means and variances. This is referred to as the E-step. Then, in the M-step, the algorithm uses the responses to update the given parameters. These two steps are iteratively executed until the parameters approach the true parameters of the dataset. Once those parameters are determined, the Gaussian models in the GMM are fixed, making clustering possible with the Gaussian models.

Many benefits are associated with parametric techniques. Among the benefits, first, these techniques provide a probability criterion to determine whether every data instance belongs to a cluster. Second, they do not require additional information, such as labels on data instances to indicate their states. On the other hand, parametric techniques cannot be deployed in a distributed way, because a significant number of data instances is required to estimate the mean and variance. Thus, outlier detection methods using parametric techniques are deployed in a centralized way.

2.4.2 Non-Parametric Techniques

Other outlier detection algorithms use non-parametric techniques, which cluster datasets without using statistical models. These techniques make certain assumptions, such as density smoothness. They typically use histograms, as in [105], [44], and [45]. Histogram-based approaches are appropriate for datasets in low-dimensional spaces, because the computations in these techniques are exponential in the dimensions of the dataset. On the other hand, this type of approach has low scalability to problems with larger numbers of data instances and higher-dimensional spaces.

One typical non-parametric cluster algorithm is $k\text{-}means$ [58]. In $k\text{-}means$, when

candidate cluster centers are first provided to the algorithm, the number of centers is equal to the number of clusters. Then, the algorithm calculates the sum of the distances from the center of each cluster to every data instance. These two steps are iteratively executed, and the given cluster centers are updated by minimizing the calculated sum. Once cluster centers are determined, clusters are formed. Unfortunately, the k-$means$ algorithm cannot guarantee that the candidate centers will be close to the true cluster centers. The iterative nature and clustering accuracy of the algorithm are thus not satisfactory.

To overcome the disadvantages of the k-$means$ algorithm, Arthur, et al. [2] proposed an extension called k-$means$++. The difference is that k-$means$++ uses the number of k values to perform a calculation that identifies appropriate data instances to use as initial centers. In contrast, in the k-$means$ algorithm the initial centers are randomly selected, which increases the number of clustering iterations. Therefore, k-$means$++ requires fewer iterations than k-$means$.

In conclusion, there are disadvantages associated with the use of both parametric and non-parametric techniques in WSNs. Parametric techniques can only estimate a model when sufficient data is available, and therefore, they are difficult to use in a distributed way. In contrast, non-parametric techniques can be executed in a distributed way but cannot provide a probability criterion for detection. Moreover, both techniques need a massive number of iterations to form clusters, requiring significant computing power, and both use random data instances to start, causing low accuracy.

2.4.3 Outlier Detection in WSN Applications

Outliers are very common in collected WSN datasets for two reasons. First, sensor nodes are vulnerable to failure, because WSNs are often deployed in harsh environments [39], [19], [90], [9], and outliers are commonly found in datasets collected by such WSNs [91], [50]. Second, wireless signal noise and malicious attacks both create outliers [63], [53], which obviously reduce WSN capabilities.

Clustering methods are also used for outlier detection in WSN applications. For instance, to robustly estimate the positions of sensor nodes, [14] used the EM algorithm to iteratively detect outlier measurements. The algorithm was used to calculate variables that could indicate whether a particular measurement was an outlier. [123] conducted similar work using the EM algorithm to detect outliers. Additionally, [86] proposed a novel flow-based outlier detection scheme based on the k-means clustering algorithm. This approach separates a dataset containing unlabeled flow records into normal and anomalous clusters. Similar research by [40] used k-means to detect heart disease. Unfortunately, approaches using the EM and k-means algorithms to detect outliers suffer from the previously mentioned problems of heavy iteration for clustering and low accuracy. In contrast, the novel approach introduced in Chapter 5 can solve such problems.

Chapter 3

Distributed a Supervised Learning Algorithm in WSNs

A Preliminary Experiment

3.1 Introduction

WSNs that monitor environments are usually used for preventing and predicting events. For example, WSNs are used to monitor dangerous volcanoes or predict weather changes. WSNs cooperating with data processing approaches can be used to minimize effects of disasters. However, faulty sensor data detection is a critical problem in WSNs, because faulty sensor data can reduce the abilities of WSNs and lead to substantial problems. This issue has been investigated by many researchers. Previous work [125] proposed two types of faulty sensor data. One type of fault is due to physical damage of sensor nodes that is easy to detect and is defined as "function fault". For example, sensor nodes may have some physical problems that stop communicating with other sensor nodes. The other fault type is defined as a "faulty data" that is difficult to detect. Because the sensor node can be still communicating with other sensor nodes, however, the measurements of the sensor node are incorrect.

There is a high probability to encounter faulty data when with more and more data is gathered by advanced WSNs from environment, which motivated us to develop efficient methods to detect faulty data. Several methods have been proposed. For example, improved routing algorithms have been used to reduce data communication, centralize data, and handle all of the data in a base node. Although these methods can increase lifetime of WSNs, it cannot directly obtain result from a remote sensor node. Moreover, these improved routing algorithms require fixed thresholds or parameters, and the thresholds or parameters cannot be updated, so these algorithms are not adapted to dynamic environments.

In contrast, machine learning algorithms are based on statistics that dynamically extract

properties from environment data, thus machine learning algorithms do not restricted by fixed threshold or parameters. Moreover, machine learning algorithms can analyze these properties to find changes in data, and then exploit the changes to manipulate thresholds or parameters so that machine learning algorithms are adapted to dynamic environment. One problem of applying machine learning algorithms in WSNs is that these algorithms are very complex, and sensor nodes cannot execute these machine learning algorithms for a long time. One solution is appropriate distributing these computationally intensive machine learning algorithms over the WSNs. However, it is difficult to distribute machine learning algorithms in WSNs, because machine learning algorithms analyze data in a centralized way, which means the machine learning algorithms need to collect all sensor nodes' data.

Furthermore, machine learning algorithms can be applied to two classes of problems; quantitative problems such as using historical temperature to predict future temperature, and qualitative problems such as using information of credit cards to detect whether a credit card has been stolen. Faulty data detection is a qualitative problem, and logistical regression algorithm (one of machine learning algorithms) can appropriately solve the problem of faulty data detection in WSNs. Because the logistical regression algorithm do not need many features of environment, which it can reduce computation complex. Moreover, we divide the logistical regression algorithm into two steps. The first step executed on sink node that is a powerful sensor node and other sensor nodes connect with it. Sink node processes data of all sensor nodes. The second step executed on each sensor nodes, sensor nodes detect whether new measured data is a faulty data.

3.2 Proposed method

In this section, we first briefly discuss Logistical Regression, and then we describe how to distribute our logistical regression algorithm in WSNs.

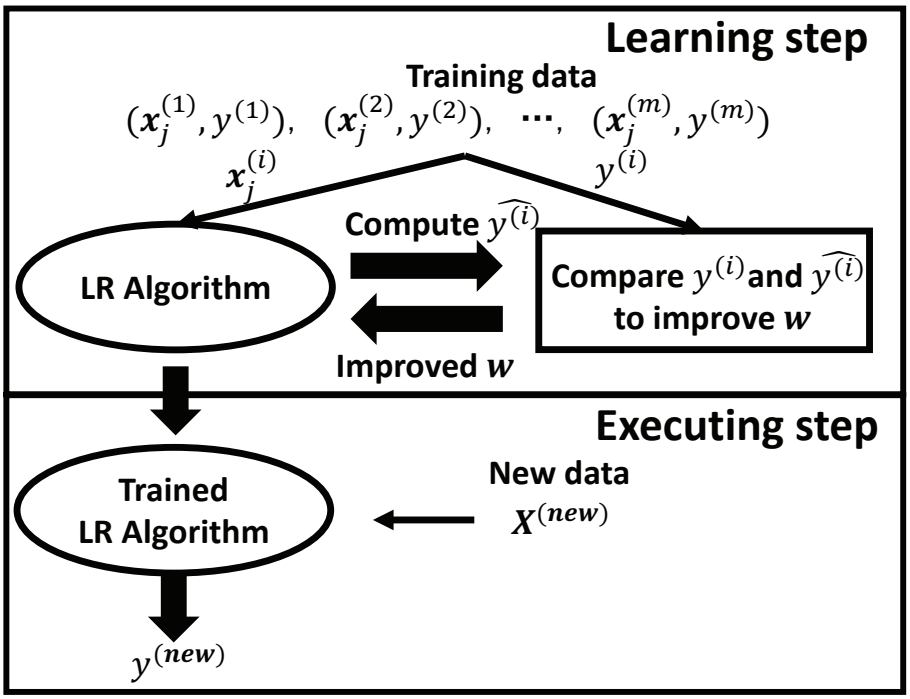

Figure 3.1: Relationship Between Learning Step and Executing Step

3.2.1 Logistic Regression

Logistical Regression (LR) algorithm is typically used to solve qualitative problems such as classification problems. A machine learning algorithms contain two steps, which are Learning and execution steps, so as the LR algorithm. We respectively introduce these two steps and the Fig. 3.1 shows the relationship between them.

Learning step

In the Learning step, parameters **w** of LR algorithm are calculated using the training data (which contains input data $\mathbf{x}^{(i)}$ and its corresponding result $y^{(i)}$) to make difference between $y^{(i)}$ and $\hat{y}^{(i)}$ as small as possible. The $\hat{y}^{(i)}$ is estimation of $y^{(i)}$, which can be represented by sigmoid function. Thus, if we gather a appropriate **w** in the learning step, we can use LR algorithm to calculate corresponding $\hat{y}^{(new)}$ of new input data $\mathbf{x}^{(new)}$.

Logistical regression is used to predict a qualitative result. For example, using temperature data gathered by a WSN and a given qualitative result, we can fit a line that divides the data into two classes indicating the qualitative result. Then, this line can be used to classify new data into their corresponding class, and predict the qualitative result of this new data. However, a data falls into which side needs a quantity of probability to present. For example, a data has 90% probability fall into normal side, and we can predict this data is a normal one. Therefore, the decision line and the probability presentation are the basic

requirements of the LR algorithm. Hence, a function that can present the decision line and probability is needed. Fortunately, the sigmoid function satisfies these requirements, and is defined as

$$P(\mathbf{x}^{(i)}) = \frac{1}{1 + e^{-\mathbf{w}\mathbf{x}^{(i)}}}; \quad i \in (1, \cdots, m). \tag{3.1}$$

Suppose that we have m sets of training data $(\mathbf{x}^{(i)}, y^{(i)})$. Each vector $\mathbf{x}^{(i)}$ has n input values that are used to calculate the probability of qualitative result $P(\mathbf{x}^{(i)})$. For example $\mathbf{x}^{(i)}$ has n $x_j^{(i)}$ $j \in (1, 2, \cdots, n)$. Each $x_j^{(i)}$ is one type of gathered data, e.g., $x_1^{(i)}$ could represent temperature and $x_2^{(i)}$ could represent the data transmission time. Each $x_j^{(i)}$ must be gathered in the same situation i, otherwise LR cannot predict the qualitative result. For example, at time i there is relationship between $x_1^{(i)}$ and $x_2^{(i)}$ that can be used to predict the qualitative result, but there is less correlation between $x_1^{(i)}$ and $x_2^{(i+1)}$. $y^{(i)}$ is a given qualitative result. The purpose of this function is to make the value of $P(\mathbf{x}^{(i)})$ approach $y^{(i)}$ by optimizing the parameter $\mathbf{w}$, which has n elements w_j $j \in (1, 2, \cdots, n)$ that are associated with the $x_j^{(i)}$. To optimize $\mathbf{w}$, we use a mathematical equation called a likelihood function as follows.

$$l(\mathbf{w}) = \prod_{i=1}^{m} P(\mathbf{x}^{(i)})^{y^{(i)}} [1 - P(\mathbf{x}^{(i)})]^{1-y^{(i)}}. \tag{3.2}$$

In this function, we introduce a variable $y^{(i)}$ that reduces $P(x^{(i)})$ when it is incorrect. $y^{(i)}$ is defined as

$$y_{(i)} = \begin{cases} 0 & \text{if } P(\mathbf{x}^{(i)}) < \text{threshold} \\ 1 & \text{otherwise.} \end{cases} \tag{3.3}$$

To calculate the optimized value of the likelihood function, instead of obtaining the maximum, we take log of the above function and construct the cost function

$$\begin{aligned} J(\mathbf{w}) = \; & - \; \frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log(P(\mathbf{x}^{(i)})) \\ & - \; (1 - y^{(i)}) \log(1 - P(\mathbf{x}^{(i)}))]. \end{aligned} \tag{3.4}$$

Then, we calculate the minimum value of the cost function. The figure of cost function is shown in Fig. 3.2. In this figure, it shows the function is convergence, and the function has a minimum value at the intersection of the two curves.

To find the minimum value of $J(\mathbf{w})$, we must solve $\frac{\partial J(\mathbf{w})}{\partial w_{(j)}} = 0$. The partial derivative of $J(\mathbf{w})$ with respect to $\mathbf{w}$ is

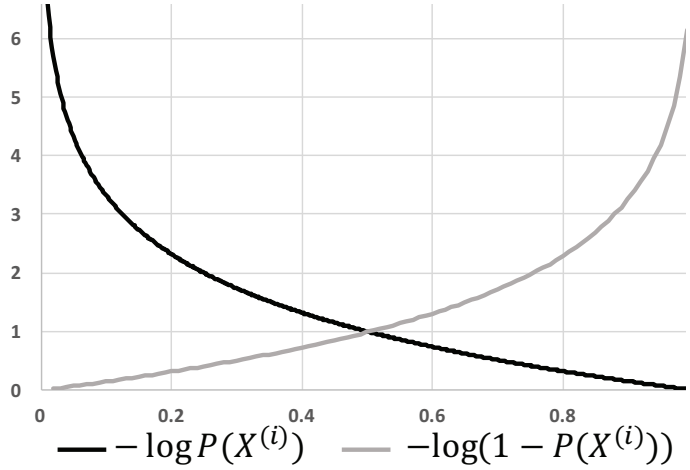

Figure 3.2: Figure of cos function

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m}[P(x^{(i)}) - y^{(i)}]x_j^{(i)} \tag{3.5}$$

$$= \frac{1}{m}(\mathbf{x}^{(i)})^T[P(\mathbf{x}^{(i)}\mathbf{w}) - \mathbf{y}^{(i)}]. \tag{3.6}$$

Furthermore, we can optimize w using

$$w_j^{(new)} = w_j - \frac{\alpha}{m}\sum_{i=1}^{m}[P(x^{(i)}) - y^{(i)}]x_j^{(i)} \tag{3.7}$$

$$\mathbf{w}^{(new)} = \mathbf{w} - \frac{\alpha}{m}(\mathbf{x}^{(i)})^T[P(\mathbf{x}^{(i)}\mathbf{w}) - \mathbf{y}^{(i)}]. \tag{3.8}$$

Execution step

α is a constant that controls the training speed. In the execution step, we apply the optimized **w** and new input data to calculate $P(\mathbf{x}^{(i)})$ using the sigmoid function. Then, we can also determine y, which predicts the qualitative result for the new input data.

We give an example of LR algorithm by Fig.3.1. Supposing we have m sets of training data $(\mathbf{x}^{(i)}, y^{(i)})$ $i \in (1, 2, \cdots, m)$, and $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \cdots, x_j^{(i)})$ $j \in (1, 2, \cdots, n)$. Each $x_j^{(i)}$ indicates one feature (e.g., temperature or humidity). $y^{(i)}$ is a given qualitative result corresponding to the $\mathbf{x}^{(i)}$. Parameter of $\mathbf{x}^{(i)}$ is **w** who also has n elements, and initialization value of each element is 0. We optimize parameter **w** by training data. Then, we use the optimized **w** to predict a qualitative result $\hat{y}^{(i)}$. Hence, the final goal of the LR algorithm is to find optimal parameter **w** using the training data $(\mathbf{x}^{(i)}, y^{(i)})$. In conclusion, learning step optimizes parameter **w**, and execution step predicts qualitative result $\hat{y}$.

3.2.2 Algorithm

We deploy learning step in the sink node of WSNs. Every sensor node sends measured data, which is input data, to sink node, then sink node executes the learning step, and sink node sends optimized LR algorithm to corresponding sensor nodes. The Learning step of the LR algorithm is as follows.

Algorithm 1: Logistical Regression (**Learning step**)

Set $w_j \leftarrow 0$; $D \leftarrow 0$; $i \in (1, 2, \cdots, m)$; $j \in (1, 2, \cdots, n)$
while $i < m$ **do**
  while $j < n$ **do**
    $D \leftarrow D + [\hat{y}^{(i)} - y^{(i)}]x_j^{(i)}$;
    $w_j = w_j - \frac{\alpha}{m}D$;
    $j \leftarrow j + 1$;
  end while
  $i \leftarrow i + 1$;
end while

In algorithm1, α is a constant that controls learning speed of optimizing parameter **w**. D is a temporary value that used to update the **w**. $\hat{y}^{(i)}$ is an estimation of $y^{(i)}$, which can be represented by sigmoid function and you can find the detail in appendix. After learning step, as shown in Fig.3.3, sink node sends optimized **w** to every cooperating sensor nodes. Then, each sensor node executes the execution step. The algorithm of execution step is described as follows.

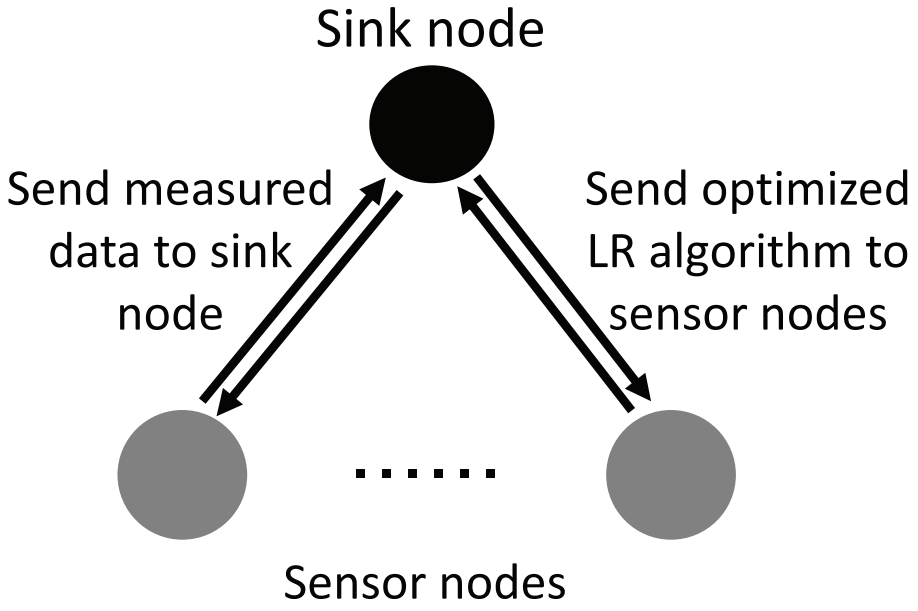

Figure 3.3: WSN Structure

Algorithm 2: Linear Regression **execution step**

Set $Y \leftarrow 0$
if $\left(\hat{y}^{(i)} > threshold\right)$ **then**
    $Y = 1$
else
    $Y = 0$
end if

In algorithm 2, Y is a binary value that indicates status of measured data of sensor nodes. $Y = 1$ indicates faulty measured sensor data and $Y = 0$ indicates normal measured sensor data. The value of Y is decided by a 'threshold'. A higher threshold corresponds to a higher accuracy. However, an excessively higher threshold leads to overfitting, which can result in false positive (normal sensor data is classed as faulty). In this paper, we set the default value of threshold to 0.5, which is commonly used in in machine learning area.

3.3 Simulation

When a WSN is deployed in an environment to realize some certain application, if measured data collected by sensor nodes contain a lot of faulty data, the faulty data have significant effects on efficiency and accuracy. In this paper, we suppose that sensor nodes are gathering data, such as temperature and humidity, from its deployed environment. Then, sensor nodes transmit the measured data to its sink node. In the sink node, we use the measured data to train LR algorithm first. Then, sink node send the optimized LR algorithm to sensor nodes, and sensor nodes use the LR algorithm to detect the status of new measured data.

3.3.1 Simulation introduction

Our simulation use three different synthetic data sets to test robustness of our method. Each of the three synthetic data sets was generated from normal distribution with $\sigma \in (0.2,\ 0.5,\ 1)$ respectively. σ is the variance of a Gaussian distribution, and it controls the amount of faulty data, a higher σ corresponds to higher amount of faulty data. Moreover, each σ has two different sizes of training sets, which respectively contains 200 and 600 data, to investigate the influence of the size of the training set on $\mathbf{w}$. Every synthetic data set contains two variables, temperature and humidity. The mean of temperature is $36.5C^{\circ}$, and the mean of humidity is 20%. We also prepare test data set to assess prediction accuracy of our method, in which the data status (normal or faulty) of $\mathbf{x}^{(i)}$ in test sets is known. We use the optimized LR algorithm and $\mathbf{x}^{(i)}$ in test set to predict the status of of $\mathbf{x}^{(i)}$ of test sets. Because the status of $\mathbf{x}^{(i)}$ in test sets is known, we can assess prediction accuracy of our

method. Additionally, the test data contains 500 data.

3.3.2 Simulation results

Figures 3.4–3.15 show the prediction results by our method under different data set, where σ are $0.2,\ 0.5$ and 1. The triangles in every figures indicate the normal sensor data, and the crosses indicate outliers. The top plot of each figure shows that the LR algorithm exploits training data to estimate a bounder line that surrounds normal sensor data. The bottom plot of each figure shows that the result of sensor nodes use the optimized LR algorithm, the normal data is bounded by a boundary line.

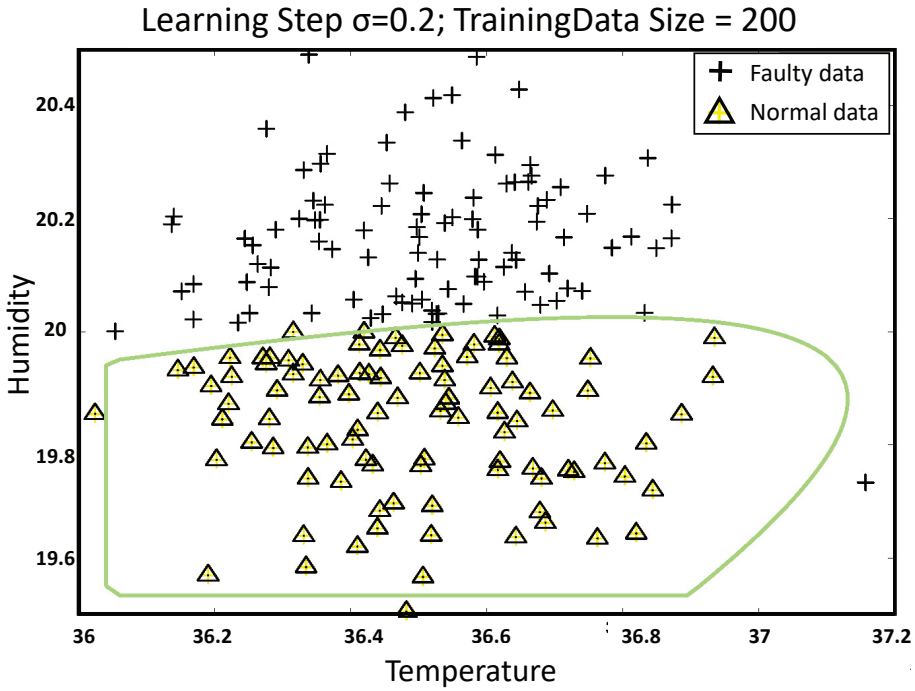

Figure 3.4: Result of Training data is 200, $\sigma = 0.2$

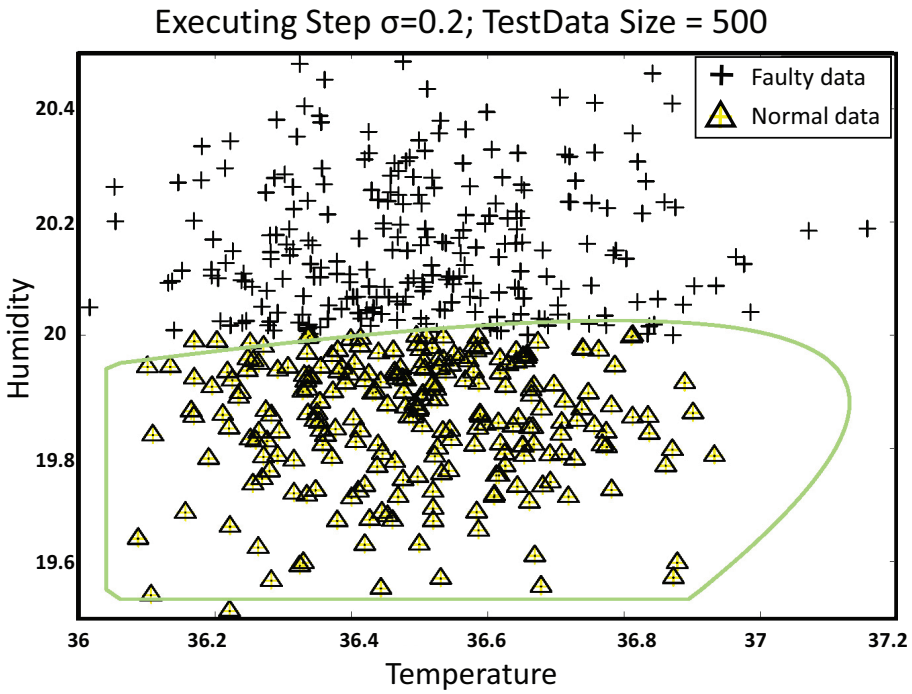

Figure 3.5: Result of Training data is 200, $\sigma = 0.2$

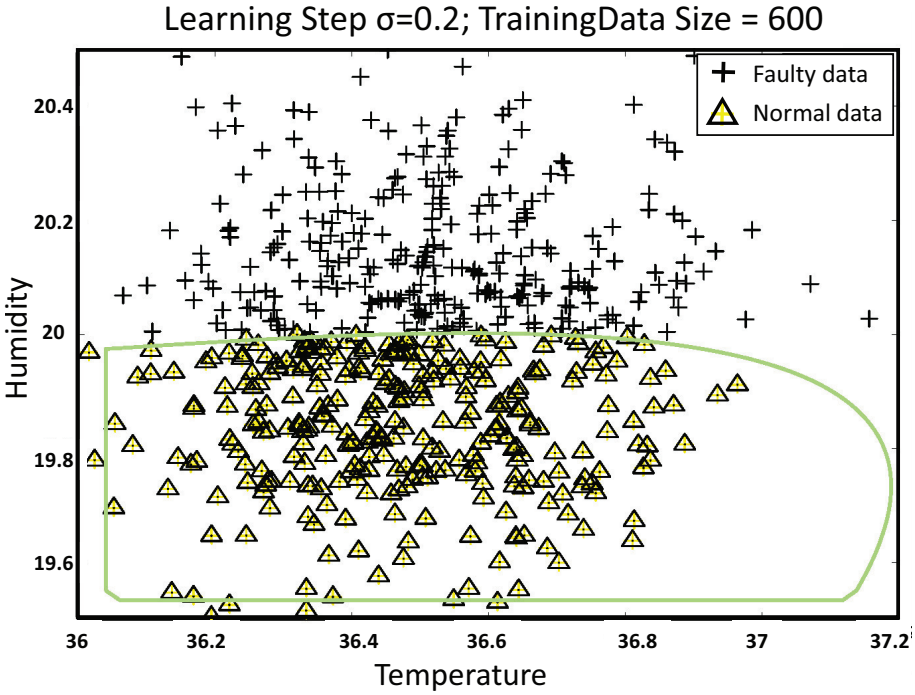

Figure 3.6: Result of Training data is 600, $\sigma = 0.2$

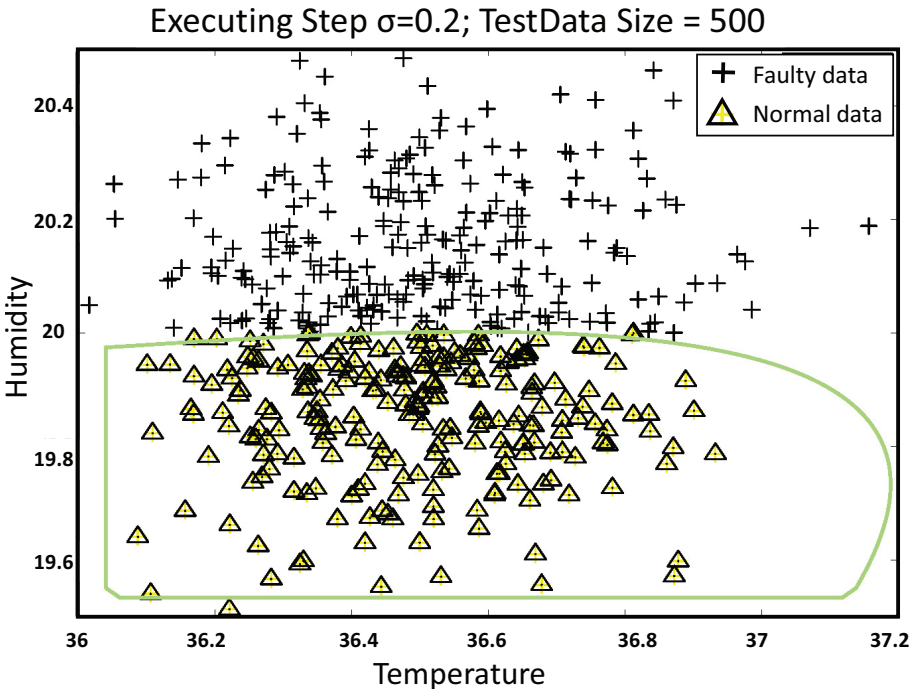

Figure 3.7: Result of Training data is 600, $\sigma = 0.2$

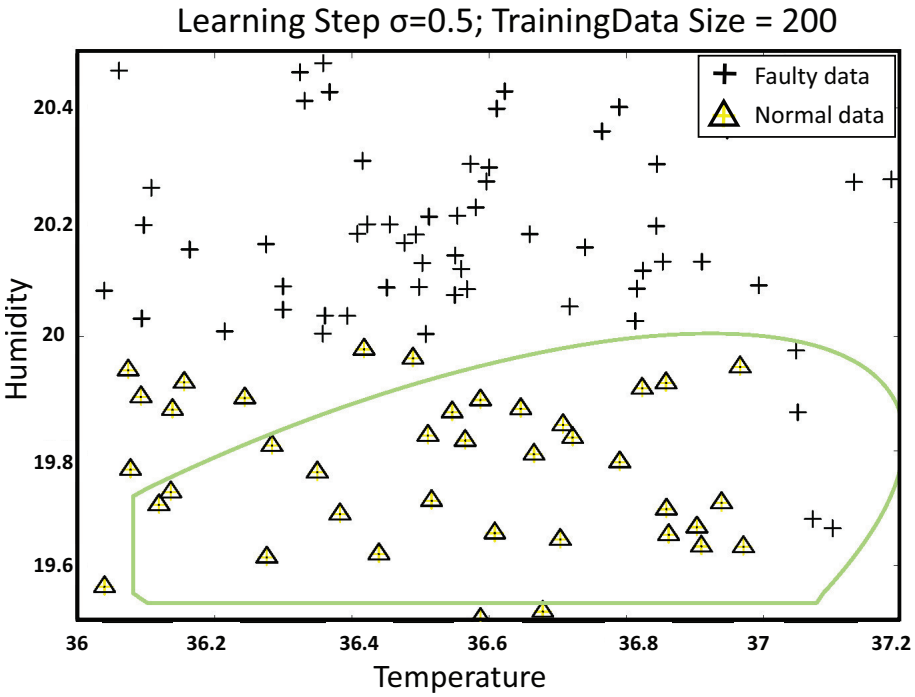

Figure 3.8: Result of Training data is 200, $\sigma = 0.5$

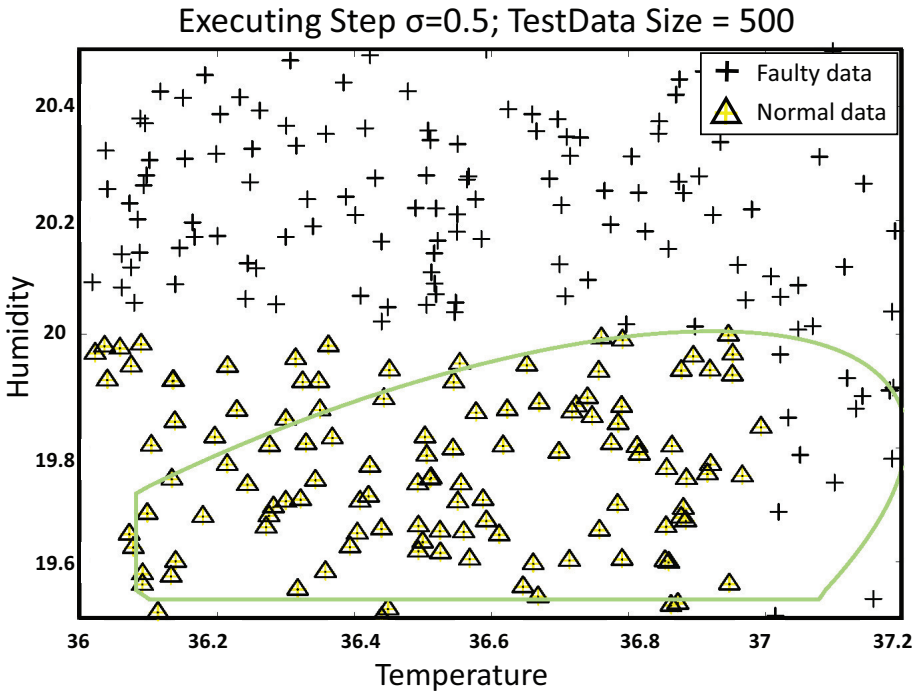

Figure 3.9: Result of Training data is 200, $\sigma = 0.5$

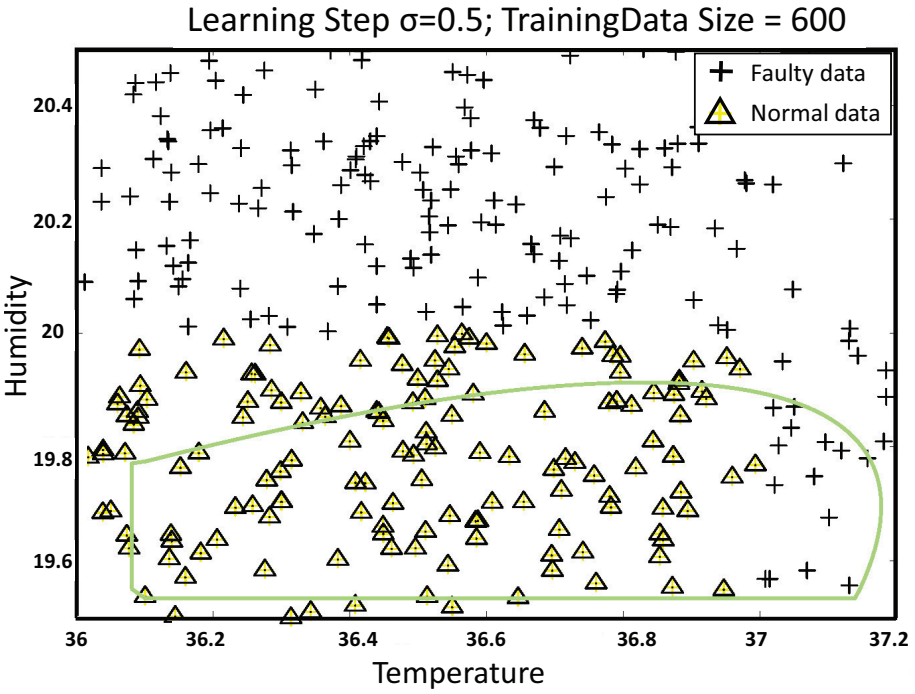

Figure 3.10: Result of Training data is 600, $\sigma = 0.5$

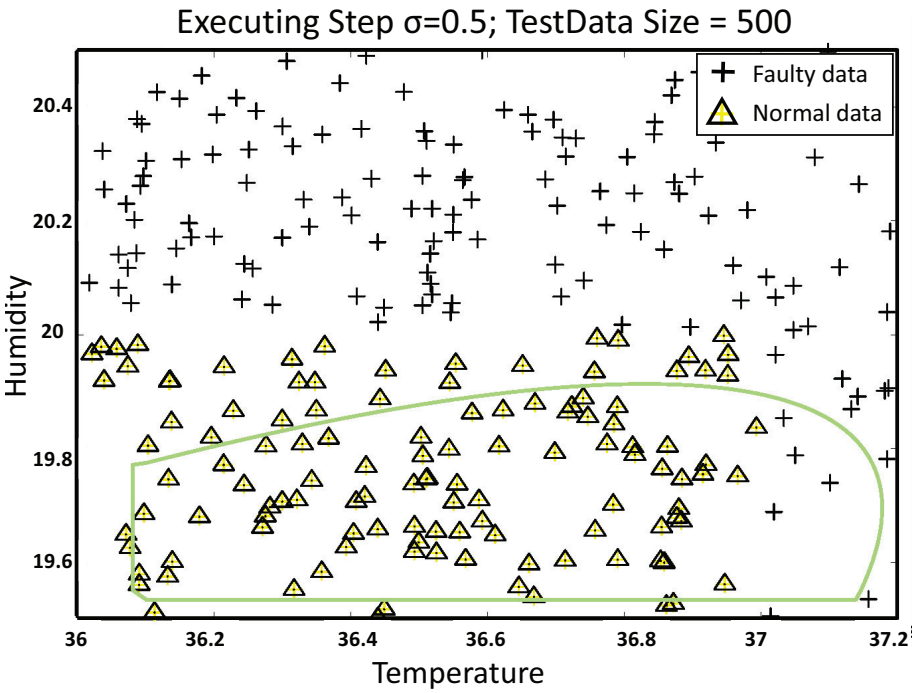

Figure 3.11: Result of Training data is 600, $\sigma = 0.5$

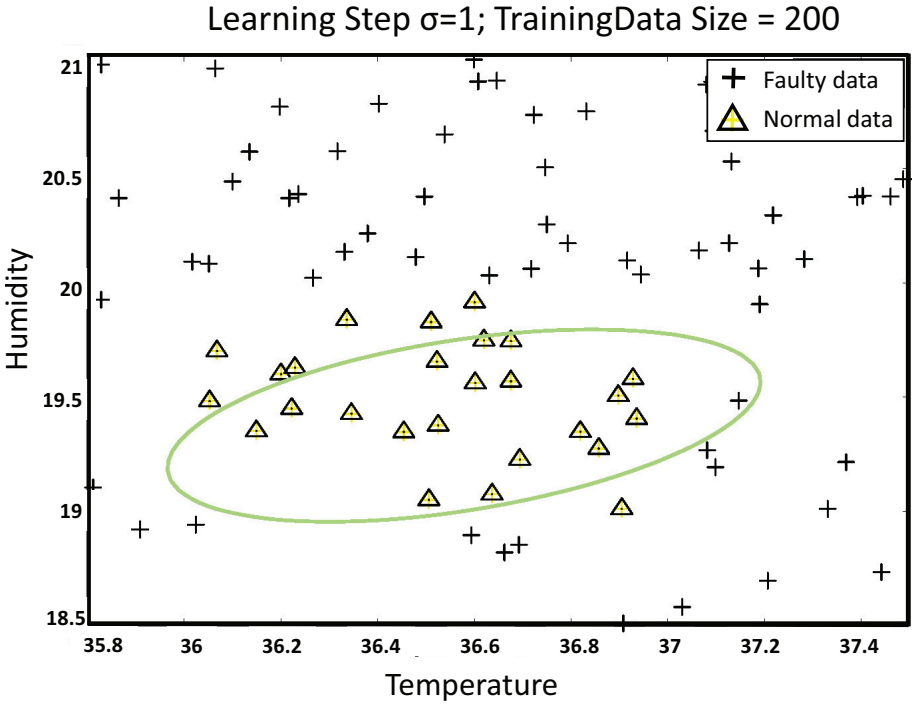

Figure 3.12: Result of Training data is 200, $\sigma = 1$

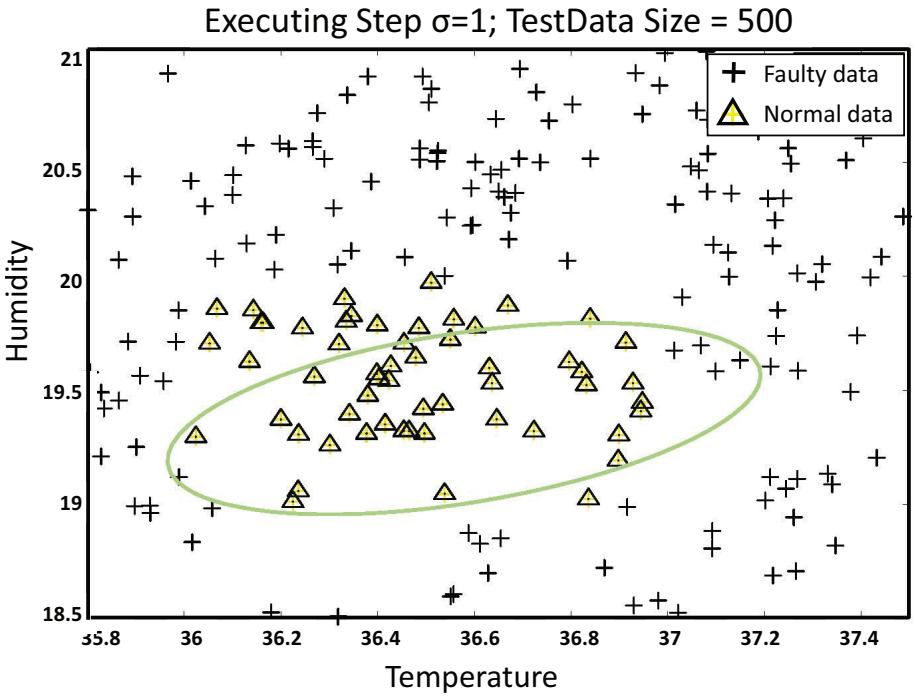

Figure 3.13: Result of Training data is 200, $\sigma = 1$

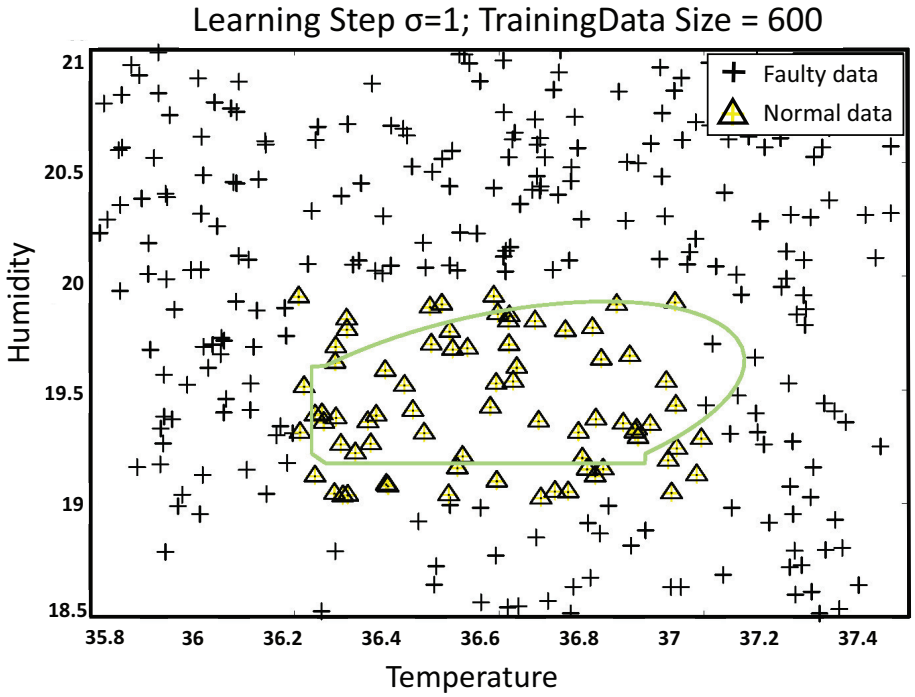

Figure 3.14: Result of Training data is 600, $\sigma = 1$

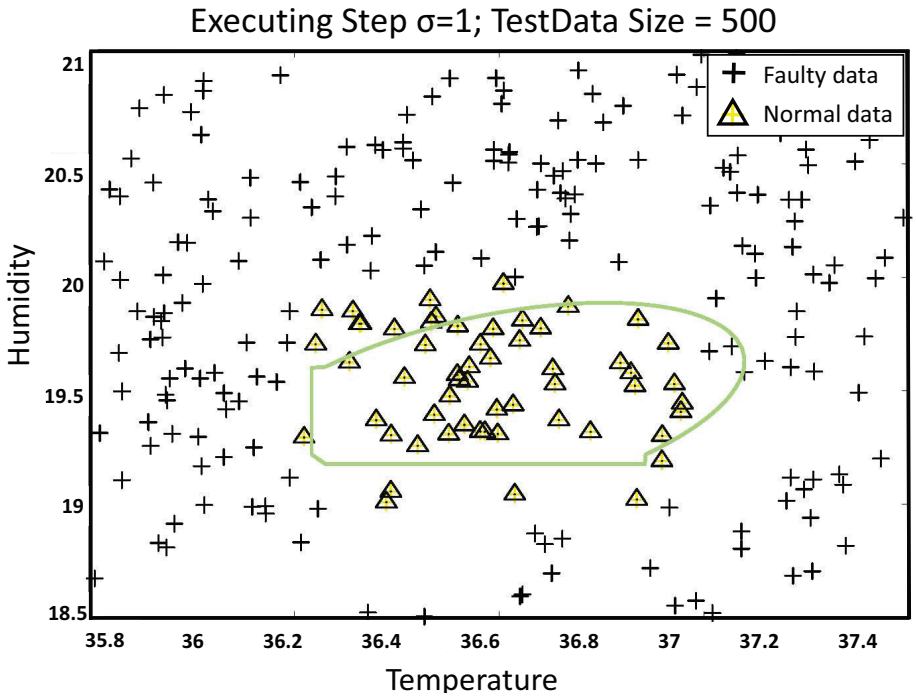

Figure 3.15: Result of Training data is 600, $\sigma = 1$

As shown in the figures, the learning step in the sink node can properly optimize LR algorithm. We can see this fact in every top plot of figures, that the bounder line can precisely bound normal data. Then, this optimized LR algorithm is sent to sensor node who can utilize the optimized LR algorithm to predict the status of sensor data. We can see the prediction result in bottom plot of every figures. Table 3.1 shows the prediction accuracy result of simulation.

Table 3.1: Prediction Accuracy Result

Size of Data	$\sigma = 0.2$	$\sigma = 0.5$	$\sigma = 1$
200	95.8%	89.6%	95.20%
400	99%	83.80%	88.80%
600	98.20%	87.00%	96.60%

When $\sigma = 0.2$, there is more normal data than faulty data. It is easy to train LR algorithm to distinguish normal and faulty data. The bottom plot in Fig.3.5 and Fig.3.6 Fig.3.7 show that sensor nodes precisely classified the sensor data. We also can see a high prediction accuracy of LR algorithm under the condition σ is 0.2. A contrast case is that when σ is equal to 1. In this condition, there is more faulty data and less normal data, which is shown in Fig.3.12, 3.13 and Fig. 3.14, 3.15. We can see the prediction results in both bottoms of Fig.3.13 and Fig.3.15 are also successful. These two simulation results show that the LR algorithm is very efficient and accurate when there is a big difference

between two classes. This can be proved when σ is equal to 0.5, thus the amount of normal data and faulty data is similar. We can see the prediction accuracy of LR algorithm is relatively low compared with $\sigma = 0.2$ and $\sigma = 1$. The comparison result is shown in Table3.1.

In this preliminary experiment, we found that a LR can be used to detect outliers. Preparing a training dataset is very difficult, we assume the all the data is normal in the first beginning time. Therefore, if the environment changes or some sensor nodes are faulty after a period, we cannot simply prepare a set of training data. Moreover, we assume to use a powerful sensor node to train this LR. Therefore, a single sensor node has no ability to change this trained model. For solving theses problems, we consider the unsupervised learning is much better, because it does not need training data.

Chapter 4

An Mean-shift Algorithm Based Outlier Detection in WSNs

4.1 Introduction

In this section, we first introduce types of outliers and then introduce the related concepts and assumption in our proposed method. Finally, we introduce the clustering algorithm that we used in our method: “mean-shift algorithm”.

4.1.1 Types of Outliers

Outliers are usually categorized as “global outliers” and “local outliers” (Fig. 4.1). Global outliers significantly deviate from the rest of the data points[53]. They are the simplest type of outliers and can be easily removed with some filters, such as “anchor data”, that will be used in our method. On the other hand, local outliers are data points whose pattern significantly deviates from the pattern of the local area, so additional information of neighbor data points is needed for detecting local outliers. Therefore, detecting local outliers is more difficult than detecting global outliers.

4.1.2 Related Concepts and Assumption

There are three main indexes to show the center of a dataset: “mean value”, “median value”, and “mode”.

The mean value is the average of the set of numbers, which can be easily calculated. However, it is easily affected by outliers because it becomes larger or smaller due to the effect of outliers.

The median value is the middle value in numerical order of a dataset. It is not observably affected by the outliers because if a dataset contains outliers, the median value is still

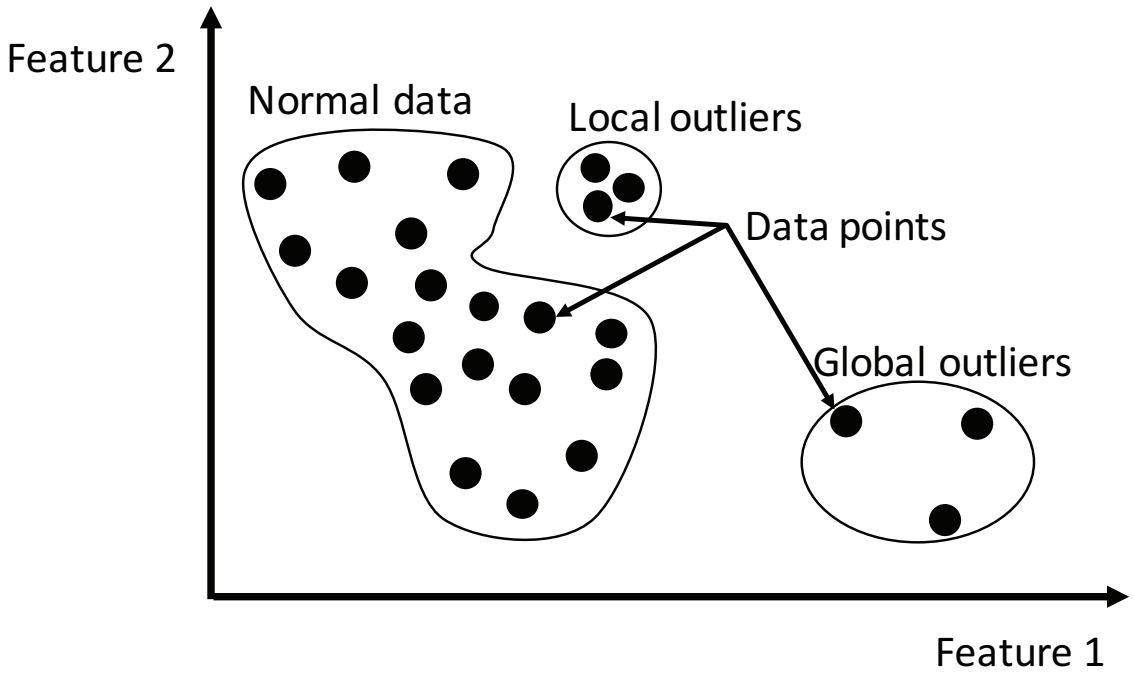

Figure 4.1: Global Outliers and Local Outliers

decided by the majority of the non-outlier data points. Hence, most data points of a dataset are around the median value of the dataset.

The mode is a point that corresponds to the maximum probability density of a dataset. Hence, most data points are around the mode, which is similar to the median. However, calculating the mode of the dataset needs a lot of calculations. We can get an approximate value for the mode by using the median of the dataset.

In this paper, we assume that data points from a similar environment are generated by the same probability density function (PDF). Moreover, outliers are generated by other PDFs. The collected sensing dataset is mixed with normal data points and outliers. As stated above, the majority of data points should be around the center of the PDF. Moreover, the probability of outliers occurring is very low [92]. Hence, most of the data points in the dataset can be considered as normal data points, and they are around the center of the PDF. We choose the median value of the dataset to approximately represent the center of the PDF that generated the normal data points.

4.1.3 Mean-Shift Algorithm

The mean-shift algorithm [36] is an unsupervised learning based cluster algorithm developed by Fukunaga and Hostetler [48] in 1975. It is an intuitive "mode" seeking method. Cheng et al. [34] showed that the mean-shift algorithm procedure is equivalent to the gradient ascent by kernel density estimation. The result of kernel density estimation is the

mode.

First, we introduce the general idea of the mean-shift algorithm. Assuming that a dataset contains N data points in an M-dimension Euclidean space, each data point contains M features, such as $\mathbf{x}_i = (x_{i1}, \ldots, x_{iM}), i = (1, \ldots, N)$. We now explain a window, radius, mean-shift vector, and mode in the mean-shift algorithm.

A window is a subset of the dataset that has center $\mathbf{x}_j$ and radius h (Fig. 4.2). It contains data points within a radius of h. The window notation in this paper is $win(\mathbf{x}_j, h)$. Every data point in a dataset can be considered as a center; hence, every data point can generate a window with radius h when initiating a mean-shift algorithm.

The radius h of a window is the only parameter of the mean-shift algorithm. The appropriate radius h is the deviation of the dataset [37]. Moreover, a stable dataset density is needed to get radius h to adapt to the dynamic environment. Hence, we introduce anchor data points.

The mean-shift vector is calculated within a window. It decides the distance (length of mean-shift vector) and direction for moving the window from the previous center ($\mathbf{x}_j$) to the next center ($\mathbf{x}_{j+1}$). At the next center, the mean-shift repeats to make a new window and calculate the mean-shift vector of the new window. This process will terminate when the length of the mean-shift vector approaches zero. The mean-shift vector is calculated with the density gradient of the kernel density estimator according to Chengs study [34]. We show the derivations in the following subsection.

The mode is the center where a window finally stops moving. Data points swept by the movement of the window are contained in the same cluster because they have the same mode (center). Moreover, if some windows share the same mode (i.e. the modes are very close together), clusters generated by those windows are merged into one cluster. Figure 4.2 shows the moving window procedure. The mode window is indicated by $win\left(\mathbf{c}_l, h\right)$, where $\mathbf{c}_l$ is called the mode of cluster l.

Kernel Density Estimator for Window

By referring to Fig. 4.2, the total kernel density estimation of probability density at window $win(\mathbf{x}_j, h)$ [24] is

$$p\left(\mathbf{x}_j\right) = \frac{1}{n^{(j)} h^M} \sum_{i=1}^{n^{(j)}} K\left(\frac{\mathbf{x}_j - \mathbf{x}_i}{h}\right), \tag{4.1}$$

where $n^{(j)}$ is the total number of data points in $win(\mathbf{x}_j, h)$.

$K\left(\bullet\right)$ is defined as the kernel function. In accordance with the radially symmetric mentioned by Cheng [34], we are only interested in kernel function $K(\mathbf{u})$ that satisfies

$$K\left(\mathbf{u}\right) = ck\left(\|\mathbf{u}\|^2\right), \tag{4.2}$$

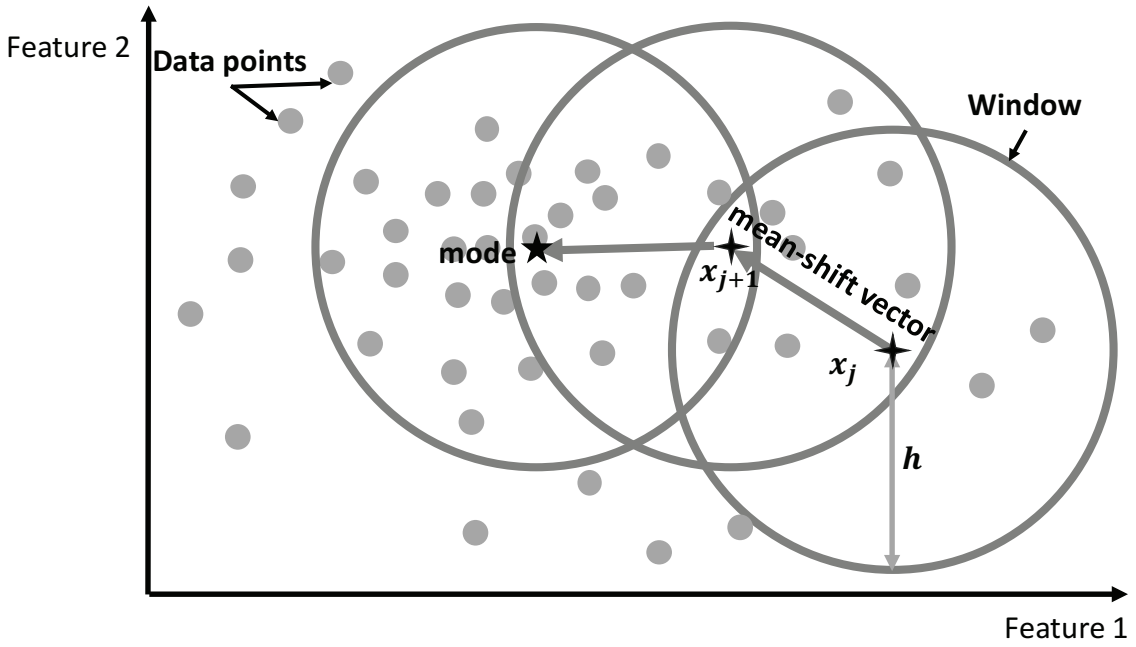

Figure 4.2: Mean-Shift migration from $\mathbf{x}_j$ to mode

where $k\left(\|\mathbf{u}\|^2\right)$ is called $profile$ of $K\left(\bullet\right)$. c is the positive normalization constant that assures the integration of the kernel function $K\left(\mathbf{u}\right)$ equals one. By utilizing the profile, we have

$$p\left(\mathbf{x}_j\right) = \frac{c}{n^{(j)}h^M}\sum_{i=1}^{n^{(j)}} k\left(\left\|\frac{\mathbf{x}_j - \mathbf{x}_i}{h}\right\|^2\right) \tag{4.3}$$

This is the kernel density estimator at $win(\mathbf{x}_j, h)$.

Calculating Mean-shift Vector of Window by using Density Gradient

To calculate the mean-shift vector of a window, we calculate the density gradient of $p\left(\mathbf{x}_j\right)$, and we set $g\left(s\right) = -k'\left(s\right)$.

$$\begin{aligned}\nabla p\left(\mathbf{x}_j\right) &= \frac{2c}{h^{M+2}}\sum_{i=1}^{n^{(j)}}\left(\mathbf{x}_i - \mathbf{x}_j\right) g\left(\left\|\frac{\mathbf{x}_j - \mathbf{x}_i}{h}\right\|^2\right) \\ &= \frac{2c}{h^{M+2}}\left[\sum_{i=1}^{n^{(j)}} g\left(\left\|\frac{\mathbf{x}_j - \mathbf{x}_i}{h}\right\|^2\right)\right] \times \left[\frac{\sum_{i=1}^{n^{(j)}}\mathbf{x}_i g\left(\left\|\frac{\mathbf{x}_j-\mathbf{x}_i}{h}\right\|^2\right)}{\sum_{i=1}^{n^{(j)}} g\left(\left\|\frac{\mathbf{x}_j-\mathbf{x}_i}{h}\right\|^2\right)} - \mathbf{x}_j\right]\end{aligned} \tag{4.4}$$

The second term of Eq. 4.4 is mean-shift vector $\mathbf{m}\left(\mathbf{x}_j\right)$ in $win(\mathbf{x}_j, h)$.

$$\mathbf{m}\left(\mathbf{x}_j\right) = \frac{\sum_{i=1}^{n^{(j)}} g\left(\left\|\frac{\mathbf{x}_j-\mathbf{x}_i}{h}\right\|^2\right)\mathbf{x}_i}{\sum_{i=1}^{n^{(j)}} g\left(\left\|\frac{\mathbf{x}_j-\mathbf{x}_i}{h}\right\|^2\right)} - \mathbf{x}_j \tag{4.5}$$

The mean-shift vector always points in the direction of the increasing maximum density as shown in Fig. 4.2. Since $\mathbf{x}_j$ and the mean-shift vector are known, the next candidate center point of a window is calculated as follows:

$$\begin{aligned} \mathbf{x}_{j+1} &= \mathbf{m}\left(\mathbf{x}_j\right) + \mathbf{x}_j \\ &= \frac{\sum_{i=1}^{n^{(j)}} g\left(\left\|\frac{\mathbf{x}_j - \mathbf{x}_i}{h}\right\|^2\right)\mathbf{x}_i}{\sum_{i=1}^{n^{(j)}} g\left(\left\|\frac{\mathbf{x}_j - \mathbf{x}_i}{h}\right\|^2\right)} \end{aligned} \tag{4.6}$$

Hence, the next window is $win(\mathbf{x}_{j+1}, h)$. Moreover, according to Cheng [34], no matter from which data point the calculation starts, the final result is convergent at the mode of probability density of the observed data.

4.2 Local Outlier Detection Method

In this section, we introduce our local outlier detection method. We assume that the WSN in our algorithm is a standard class[1] -based WSN. In accordance with the similar environment, the sensor nodes and class head (CH) are distributed into different classes. Sensor nodes communicate with their CH, which transmits the gathered sensing data points to the base station.

Supposing a WSN contains P classes and one class has $W^{(p)}, (p \in [1, \cdots, P])$ sensor nodes, each sensor node transmits G data points to CH in time period t. Hence, each CH receives a set of data points, whose size is $N^{(p)} = W^{(p)} \times G$. One data point $\mathbf{x}_i$ contains M features, $\mathbf{x}_i = (x_{i1}, \ldots, x_{iM}), i = (1, \ldots, N^{(p)})$.

The goal of the method is to cluster collected sensing data points of CH into different clusters and then find which cluster is an outlier in the sensing dataset. We add two main features to accompany the mean-shift algorithm: (1) anchor data points to fix the density of sensing dataset for each time period to efficiently utilize the mean-shift algorithm and (2) a labeling technique to classify the properties of cluster as "normal" or "outliers" in an unsupervised manner. The algorithm is divided into three steps.

4.2.1 Step 1: Fixing Density of Sensing Data and Detecting Global Outliers

We define the density of a collected sensing dataset at time period t as follows:

$$dens^{(p)} = \frac{N^{(p)}}{\Pi_{m=1}^{M} R_m^{(p)}}, \tag{4.7}$$

[1] In WSNs, a group of sensor nodes is called a 'cluster.' In this paper, we call it a 'class' to distinguish it from 'cluster' in the mean-shift algorithm.

where $R_m^{(p)}$ is the difference between the maximum and minimum value of the data points' feature m of class p at time period t. The value range of feature m of the sensing dataset is different in different time periods because the environment is different in different time periods. Thus, the density changes along with the time period.

Moreover, when the density is changing, it is not appropriate to use the mean-shift algorithm because mean-shift is sensitive to the density of a dataset, and variable density of the sensing dataset reduces the accuracy of the clustering result of the mean-shift algorithm. Furthermore, an incorrect clustering result will reduce the accuracy of outlier detection. To avoid the density changes in such a situation, we define the anchor data points, low anchor $L_m^{(p)}$, and high anchor $H_m^{(p)}$ for each feature m of class p. The user decides the value of the anchor data point of $L_m^{(p)}$ and $H_m^{(p)}$ for each feature m arbitrarily. Thus, a fixed density uses anchor data points as follows

$$\hat{dens}^{(p)} = \frac{N^{(p)}}{\Pi_{m=1}^{M}(H_m^{(p)} - L_m^{(p)})}, \tag{4.8}$$

These anchor data points can also remove global, e.g., if a data point is lower than $L_m^{(p)}$ or larger than $H_m^{(p)}$. For example, in an office, the normal temperature range is from 20 °C to 30 °C. We set two anchor data points to 15°C and 35°C. A measurement of 10°C would be a global outlier.

4.2.2 Step 2: Clustering with Mean-Shift Algorithm

The purpose of this step is to cluster the collected sensing data of class p at time period t into different clusters by mean-shift algorithm. Moreover, we have to update radius h_t at every time period to guarantee the accuracy of the clustering result. Algorithm 1 shows the procedure.

Algorithm 3: Mean-Shift based Clustering

for *sensing dataset at each* t **do**
  calculating radius h_t at time period t ;
end
for *data point* $\boldsymbol{x}_i, i \in (1, \cdots, N^{(p)})$ **do**
  execute the mean-shift algorithm ;
  by moving $win(\mathbf{x}_i, h_t)$ to $win(\mathbf{c}_l^{(p)}, h_t)$;
  data swept by $win(\mathbf{c}_l^{(p)}, h_t)$ is defined as cluster $C_l^{(p)}$;
end
if *some windows share the same* $\boldsymbol{c}^{(p)}$ **then**
  merge the clusters generated by those windows;
end

As explained in Sect. 3.2, the mean-shift algorithm can find the mode of a cluster. First, CH calculates radius h_t in time period t according to [37]. Then, the mean-shift algorithm clusters the sensing dataset by moving $win(\mathbf{x}_i, h_t), i \in (1, \cdots, N^{(p)})$ to $win(\mathbf{c}_l^{(p)}, h_t)$, where l indicates the number of clusters. If window $win(\mathbf{x}_j, h_t)$ finally stops at $\mathbf{c}_l^{(p)}$, that data points that are swept by the window is considered as cluster $C_l^{(p)}$. Moreover, if the distance between some modes of clusters is very small, we consider that these clusters share the same mode and merge those clusters. The new mode of merged cluster is the average of mode of each cluster before merging.

4.2.3 Step 3: Local Outlier Labeling Technique

We define two distances with the mode of each cluster and the median value of the collected sensing dataset, respectively. WSNs use these two distances to detect outliers. The detail of the two distances and how to detect outliers are as follows.

We define a Euclidean distance of cluster l that is the average distance from the mode $\mathbf{c}_l^{(p)}$ of cluster l to every data point in the collected sensing dataset of class p. We write this Euclidean distance as

$$Dis_l^{(p)} = \frac{\sum_{i=1}^{N^{(p)}} \left\| (\mathbf{x}_i^{(p)} - \mathbf{c}_l^{(p)}) \right\|}{N^{(p)}} \tag{4.9}$$

$\mathbf{M}_t^{(p)}$ is the median value of the collected sensing dataset of class p at time period t. We define another Euclidean distance that is the average distance from $\mathbf{M}_t^{(p)}$ to every data point in the collected sensing dataset of class p. We write it as

$$DIS^{(p)} = \frac{\sum_{i=1}^{N^{(p)}} \left\| \mathbf{x}_i^{(p)} - \mathbf{M}_t^{(p)} \right\|}{N^{(p)}} \tag{4.10}$$

We also find that $Dis_l^{(p)}$ is always larger or equal to $DIS^{(p)}$. The proof is as follows. The sensing dataset contains two parts. $\mathbf{x}_i : i = 1, \cdots, N$ is the normal part of the dataset, and $\mathbf{y}_j : j = 1, \cdots, n$ is the outlier part of the dataset. $\mathbf{M}_t$ is the median value of the dataset, and $N \gg n$. For the normal part, $\hat{\rho} = E(|\mathbf{x}_i - \mathbf{M}_t|)$ is the average deviation of the normal data points, and $\rho = max\{|\mathbf{x}_i - \mathbf{M}_t|\}$. For the outlier part, $\hat{R} = E(|\mathbf{y}_j - \mathbf{M}_t|)$ is the average deviation of outliers, and $R = min\{|\mathbf{y}_j - \mathbf{M}_t|\}$. $\mathbf{c}^{(l)}$ is the mode of cluster l, and the distance from every data point to $\mathbf{c}^{(l)}$ is:

$$\begin{aligned} Dis_l^{(p)} &= \sum_{i=1}^{N} \left|\mathbf{x}_i - \mathbf{c}^{(l)}\right| + \sum_{j=1}^{n} \left|\mathbf{y}_j - \mathbf{c}^{(l)}\right| \\ &\geq \sum_{i=1}^{N} \left(\left|\mathbf{c}^{(l)} - \mathbf{M}_t\right| - |\mathbf{x}_i - \mathbf{M}_t|\right) \\ &\geq N(R - \hat{\rho}) \end{aligned} \tag{4.11}$$

On the other hand, the distance from every data point to $\mathbf{M}_t$ is:

$$\begin{aligned} DIS^{(p)} &= \sum_{i=1}^{N} |\mathbf{x}_i - \mathbf{M}_t| + \sum_{j=1}^{n} \left|\mathbf{y}_j - \mathbf{M}_t\right| \\ &= N\hat{\rho} + n\hat{R} \end{aligned} \tag{4.12}$$

Then, the difference between $Dis_l^{(p)}$ and $DIS^{(p)}$ satisfies:

$$Dis_l^{(p)} - DIS^{(p)} \geq N(R - 2\hat{\rho}) - n\hat{R} \tag{4.13}$$

We suppose $N(R - 2\hat{\rho}) - n\hat{R} \geq 0$, then:

$$\frac{R - 2\hat{\rho}}{\hat{R}} \geq \frac{n}{N} \tag{4.14}$$

Since $R \gg \hat{\rho}$ and $N \gg n$, then $\frac{R}{\hat{R}} - 2\frac{\hat{\rho}}{\hat{R}} \gg 0$ and $\frac{R}{\hat{R}} - 2\frac{\hat{\rho}}{\hat{R}} \geq \frac{n}{N}$. Thus, our assumption that $\frac{R-2\hat{\rho}}{\hat{R}} \geq \frac{n}{N}$ is true. We get $Dis_l^{(p)} \geq DIS^{(p)}$.

According to our assumption that data from a similar environment is generated by the same PDF, the sensing data of every sensor node in the same class has the same PDF because sensor nodes in similar environments are classified into the same class. Hence, the center of every cluster (the mode of each cluster) is similar to the center of the entire sensing dataset (the median value of the entire dataset) of the class. Thus, if cluster l is normal, $Dis_l^{(p)}$ should be close to $DIS^{(p)}$. In other words, the ratio of $Dis_l^{(p)}$ to $DIS^{(p)}$ should be close to 1. Moreover, because $Dis_l^{(p)} \geq DIS^{(p)}$, we set threshold ϵ, which is a

very small empirical value, and use discrimination $\frac{Dis_l^{(p)}}{DIS^{(p)}} - 1 \leq \epsilon$ to detect outliers. The algorithm for detecting outliers is as follows.

Algorithm 4: Outlier detection of cluster ;

```
for each cluster Dis_l^(p) do
    if Dis_l^(p) / DIS^(p) − 1 ≤ ε then
        cluster l is labeled as normal ;
    else
        cluster l is labeled as outlier
    end
end
```

4.3 Simulations

In this section, we show our simulation results based on a real dataset from the Intel Berkeley Research Laboratory [7] and a synthetic dataset. We also compare our simulation results with those of Zhang et al. [121]. They detected outliers on the basis of an unsupervised method, since they used the same real dataset as we did, and we generate synthetic dataset using the same method. Moreover, we show the generality of the proposed method and compare simulation results with and without setting the anchor data since this is an important characteristic of our method.

4.3.1 Simulation Results of Real Dataset

In this subsection, we simulate our method on the real dataset from Intel Berkeley Research Laboratory as shown in Fig. 4.4. Each sensor node in the WSN records temperature, humidity, light, and voltage once every 31 seconds. We choose the same sensor nodes $1, 2, 33, 34, 35, 36$, and 37 inside the circle (35 is the CH), and we also use two features, the temperature and humidity of $5th$ March 2004, which are the same as Zhang's work. In the simulation, we only considered two features for each data point: temperature and humidity. Each sensor node contained 5000 data points, which are shown in Fig. 4.3

The normal data ranges and the averages of temperature and humidity are shown in Table 4.1. According to the settings of Table 4.1, we set four types of outliers, which are shown in Table 4.2. The four types of outlier cover the cases where outliers are close to or far away from the normal data range. Moreover, we respectively generate datasets containing 5%, 10%, 15%, 20%, and 25% outliers for every type of outlier.

- Outlier1 is near the normal data, some outliers are even inside the normal range.

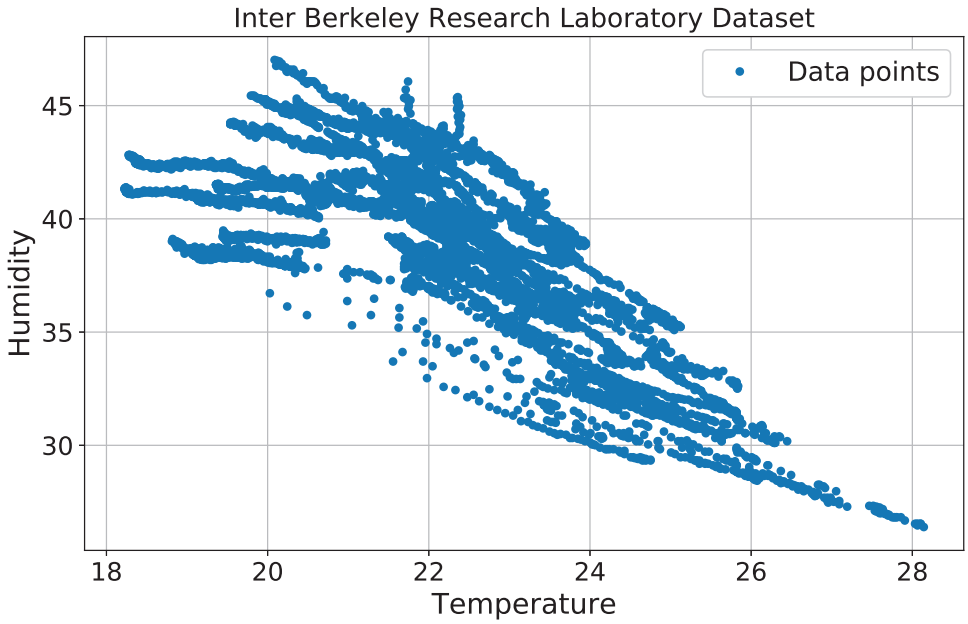

Figure 4.3: Dataset from Intel Berkeley Research Laboratory

Table 4.1: Normal Data Setting

	Range	Average
Temperature ($^{\circ}C$)	21.32 - 28.14	23.14
Humidity ($\%$)	26.39$\sim$44.02	37.69

- Outlier2 is far from the normal data; however, they cannot be removed by anchor data.
- Outlier3 is such that the value of temperature is normal; however, the value of the humidity is abnormal.
- Outlier4 is the opposite setting of Outlier3.

The following terms are used to access our method.

- True Positives (TPs) are true outliers that were detected as outliers by our method.
- False Positives (FPs) are true normal samples that are wrongly detected as outliers.
- True Negatives (TNs) are true normal samples that were detected as outliers.
- False Negatives (FNs) are true outliers that are detected as normal samples.

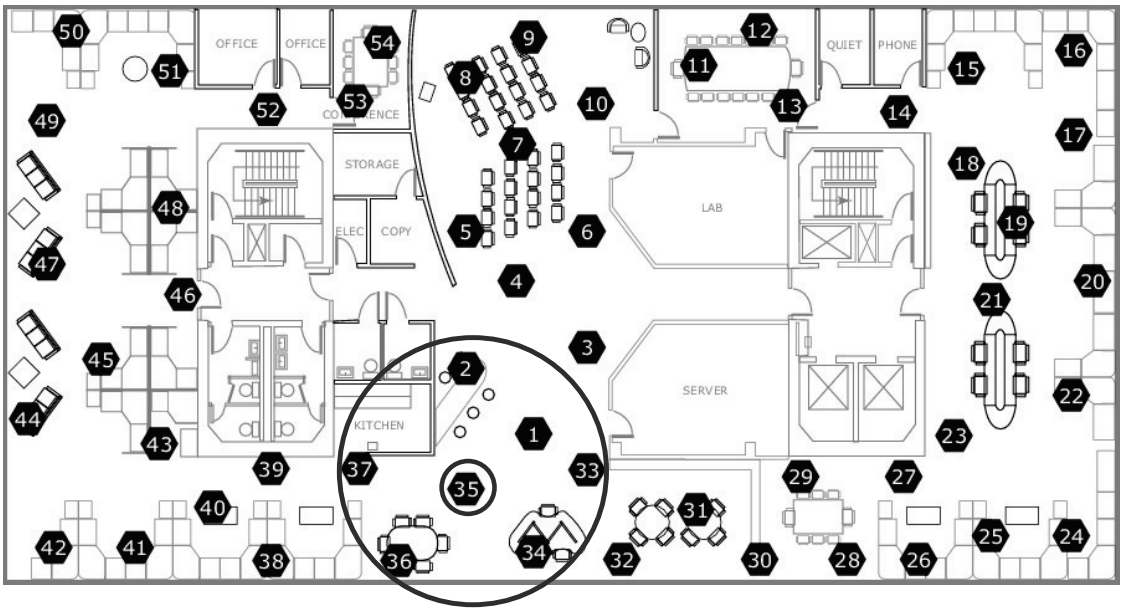

Figure 4.4: Sensor nodes deployed in Intel Berkeley Research Laboratory [7]

Table 4.2: Outlier Data Setting

Type of Outlier	Outlier1	Outlier2	Outlier3	Oulier4
Temperature ($°C$)	26∼30	31∼35	22∼28	31∼35
Humidity ($\%$)	42∼46	47∼52	47∼52	27∼44

The false positive rate (FPR) is the ratio of the normal data detected as outliers to the total true normal data, which is $\frac{FP}{FP+TN}$, and it estimates the ability of the algorithm to distinguish outliers and normal data. We compare our method's FPR with the FPR of Zhang's work; the result is shown in Fig. 4.5.

Fig. 4.5 shows that our method has ideal performance on $outlier2$, $outlier3$, and $outlier4$. However, $outlier2$ and $outlier3$ have similar curves so that $outlier2$ is blocked by $outlier3$. The FPR of $outlier2$, $outlier3$, and $outlier4$ kept below 3.3% when the outliers' percentage was less than or equal to 20%. Even in extreme conditions where a dataset contains 25% outliers, the worst case ($outliers1$) in our simulation has an FPR of about 12.8%. According to the results of the comparison in Fig. 4.5, we consider that Zhang's work had easier simulation conditions than ours in two ways: (1) they mentioned that outliers were distant from other data but did not mention how far away they were and (2) they did not test the performance when the partial feature of a data point is abnormal when performed on a real dataset, such as $outlier3$ and $outlier4$.

Moreover, $outlier2$, $outlier3$, and $outlier4$ have similar results to those of our simulation. $Outlier2$ can easily be detected as outliers because its temperature and humidity are

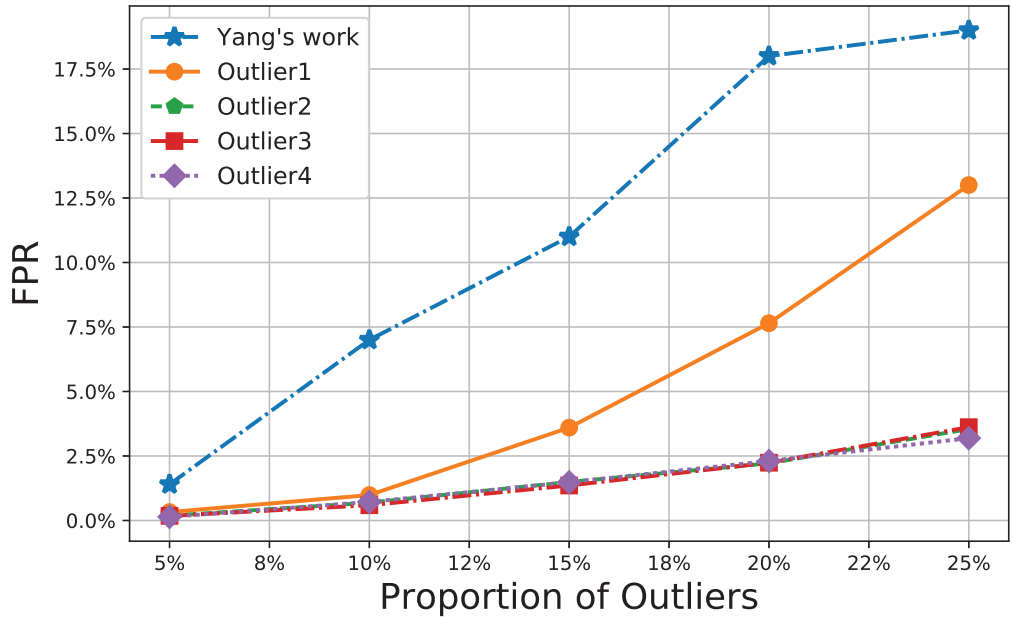

Figure 4.5: Simulation results using real dataset of Intel Berkeley Research Laboratory [7] compared with those of Zhang et al. [121].

both abnormal. Although features of $outlier3$ and $outlier4$ are partially normal, we can imagine that the distributions of $outlier3$ and $outlier4$ deviated from the normal range in 2-Dimension. The results of $outlier2$, $outlier3$, and $outlier4$ prove that our method can easily be adapted to different types of outliers.

Another fact (Fig. 4.5) is that more outliers significantly affect the FPR of our method. In $outlier1$, with the proportion of outliers increasing, more and more outliers appear in the normal range because some part of $outlier1$ overlaps the normal range. Hence, a lot of normal data points are easily detected as outliers. Similar results also appear in $outlier2$, $outlier3$, and $outlier4$ because with the proportion of outliers increasing, a great many outliers appear near to the normal range. The FPR of our method decreases when the proportion of outliers increases because normal data points are incorrectly detected as outliers. On the other hand, it can be seen that our method is sensitive to outliers, and we consider that it is a low tolerance feature for outliers of our method. The low tolerance for outliers can be estimated by another estimator called recall.

Recall is equal to $\frac{TP}{FN+TP}$ and acts as one estimator that evaluates how many true outliers are correctly detected. The recall of our simulation is shown in Fig. 4.6.

This figure shows that all types of outliers have recall near 98% when the proportion of outliers is 5%. The recalls of $outlier2$, $outlier3$, and $outlier4$ are around 96% with increasing proportion of outliers. Even the worst case with $outlier1$ with 25% outliers,

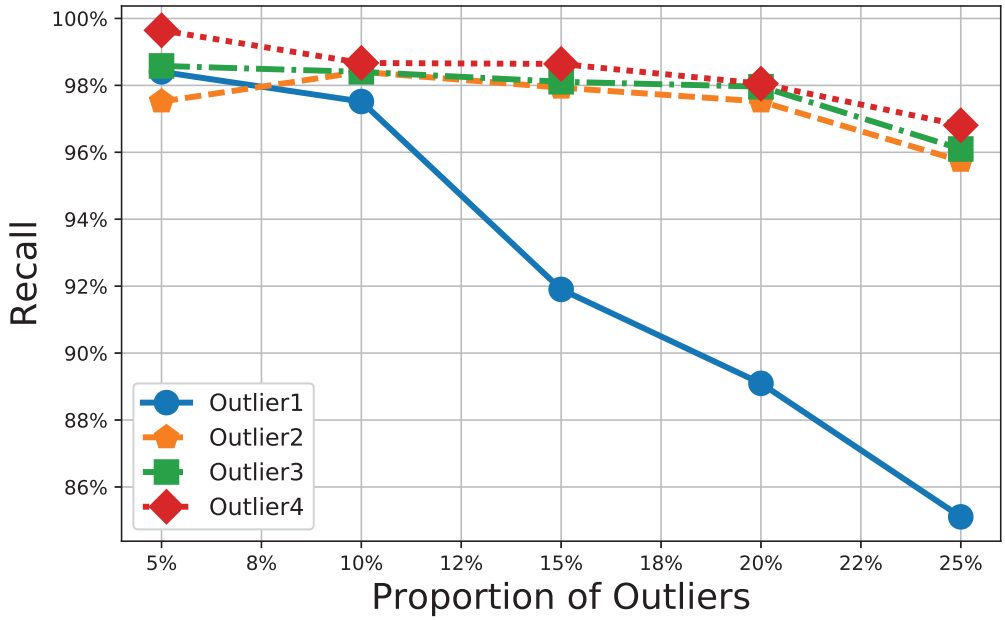

Figure 4.6: Simulation results of Recall for Fig. 4.6.

the recall is near 85%. The simulation results of every type show that our method has a very high accuracy for detecting outliers, and this is also evidence that our method has low tolerance for outliers.

Moreover, we also estimated the $precision$ and $F1$-$score$ of our method in Fig. 4.7 and Fig. 4.8. According to the simulation results, we can see that the precsion and F1-score are decresing when outliers are increasing. However, both of these measurements keep a high value on this testing dataset.

4.3.2 Simulation Results of Synthetic Datasets

We use the same method to generate synthetic data as Zhang et al. [121] did. Synthetic sensing data are generated by mixing three Gaussian distributions. The mean μ is randomly selected from $(0.3, 0.35, and 0.45)$, and the standard deviation is $\sigma = 0.03$. Outliers are generated by uniform distribution, which is distributed in an interval of $[0.5, 1]$. According to the empirical rules of Gaussian, the value range of Gaussian distributions is $\mu \pm 3\sigma$, and the normal range of the synthetic data is $[0.21, 0.54]$. This synthetic dataset blends all the conditions we discussed in real data, which are outliers overlapping normal data, outliers near to normal data, and partial feature values are normal. Hence, the generality is improved because the synthetic dataset mixed most possible outlier conditions.

Figure 4.9 compares the simulation results for FPR between our algorithm and that of Zhang's work. Because the synthetic data blends all types of outliers and the outliers were

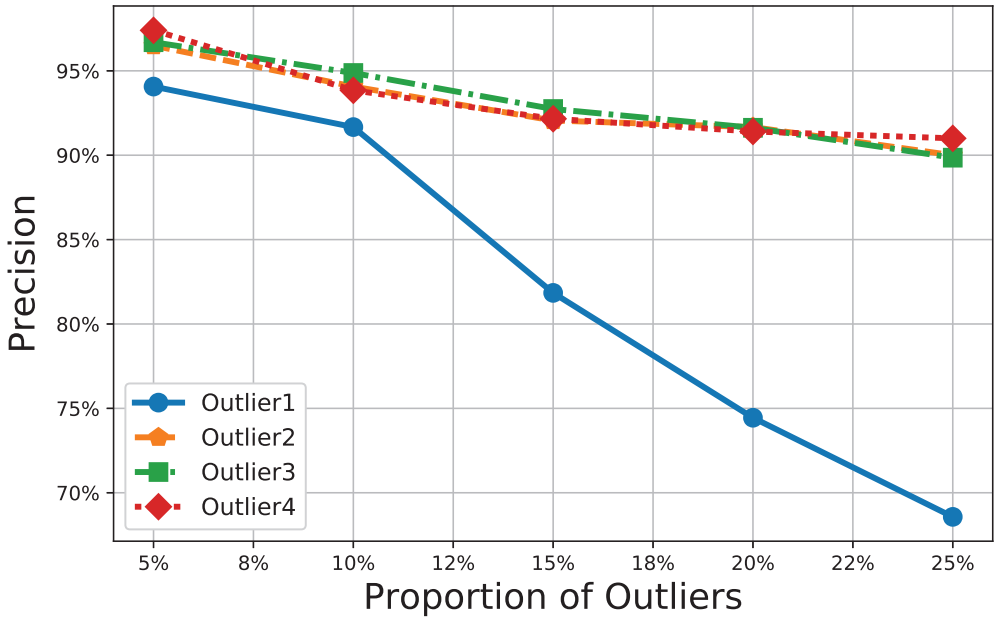

Figure 4.7: Simulation results of Precision

randomly generated, sometimes more outliers fall into or near the normal range. Thus, we can only control the quantity of outliers; however, we cannot decide where the outliers falls. This leads to the FPR of our method being higher than that of the real data, and this is the reason that the FPR is higher when the proportion of outliers is 15%.

We also calculate the recall of our method performed on the synthetic data to confirm the effect of outliers, which is shown in Fig. 4.10. The result shows that the recall of our method fluctuates because the randomly generated outliers sometimes fall inside the normal range. When outliers fall inside the normal range, they significantly affect our results. However, the recall of synthetic data has a similar trend, which is decreasing with increasing outliers, with the recall of real data. Moreover, because the probability that outliers occur is low, a dataset that contains 25% outliers is an extreme case. Even in the extreme case, the recall almost keeps to around 80% (Fig. 4.10). Hence, we conclude that our proposed method also has ideal performance in the more general cases.

4.3.3 Simulation Results Affected by Anchor Data

As mentioned in Sect. 4.1, the mean-shift algorithm may cluster the normal data into several clusters because the density of the dataset is changing with time, which leads to normal data being detected as outliers. Since using anchor data points is a feature of this work, to evaluate this aspect, we performed the following simulation where an outlier-free dataset is distributed in a 2-D Gaussian distribution. As shown in Fig. 4.11(a), two anchor

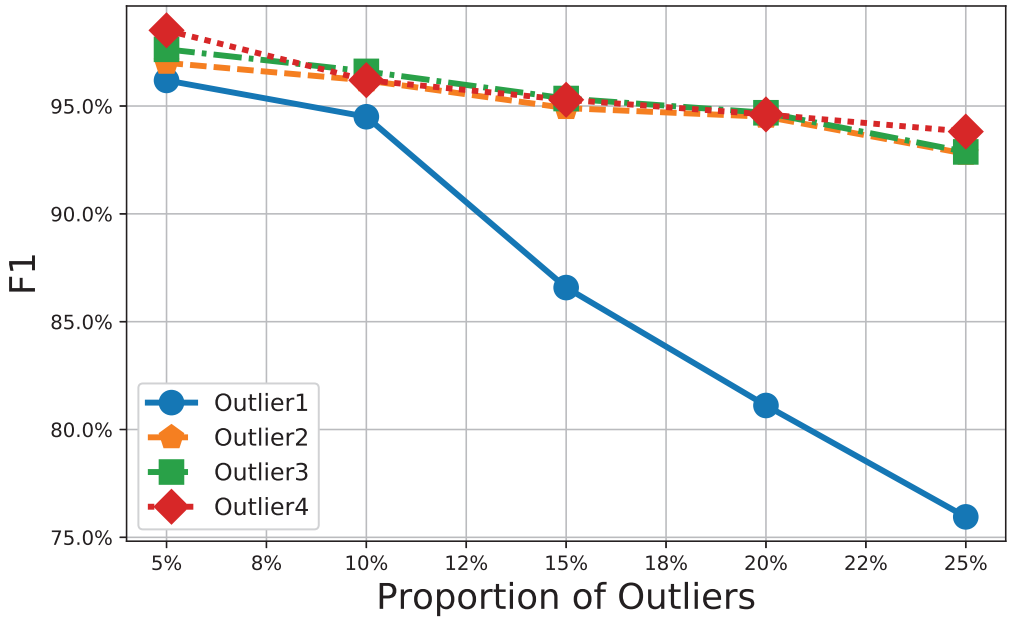

Figure 4.8: Simulation results of F1-score

data points were inserted at each point L and H (Fig. 4.12(b)). The simulation results in Fig. 4.11(a) show that, without setting anchor data points, the dataset were clustered into four classes, and two of them were determined as outliers, as shown in Fig. 4.11(a). On the other hand, the simulation results in Fig. 4.12(b) show that, taking advantage of the anchor data points, the normal data were clustered as one class and were correctly determined as "normal."

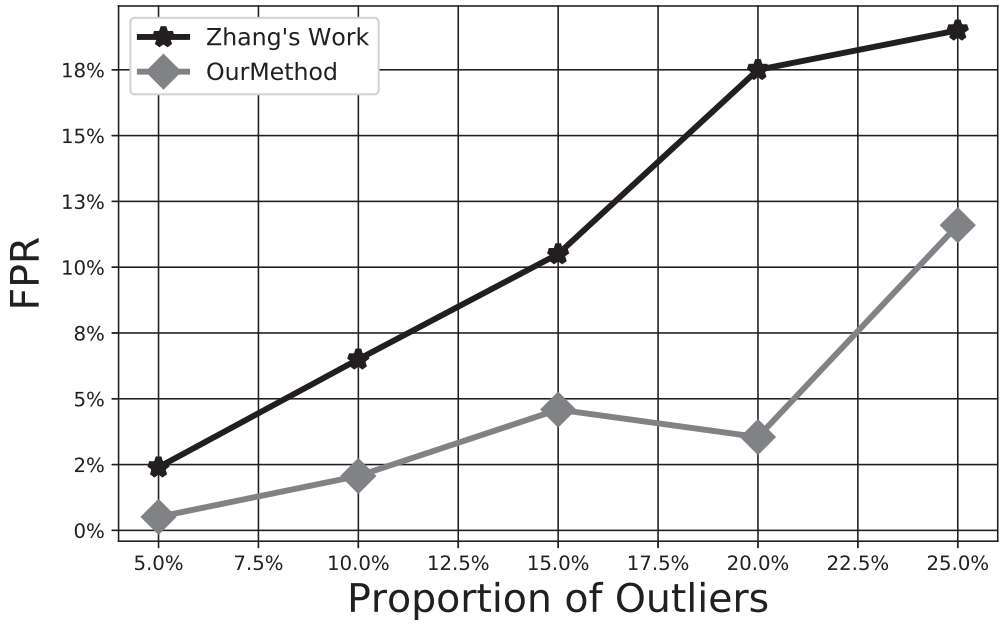

Figure 4.9: Simulation results for FPR of proposed algorithm and that of Zhang et al. [121]

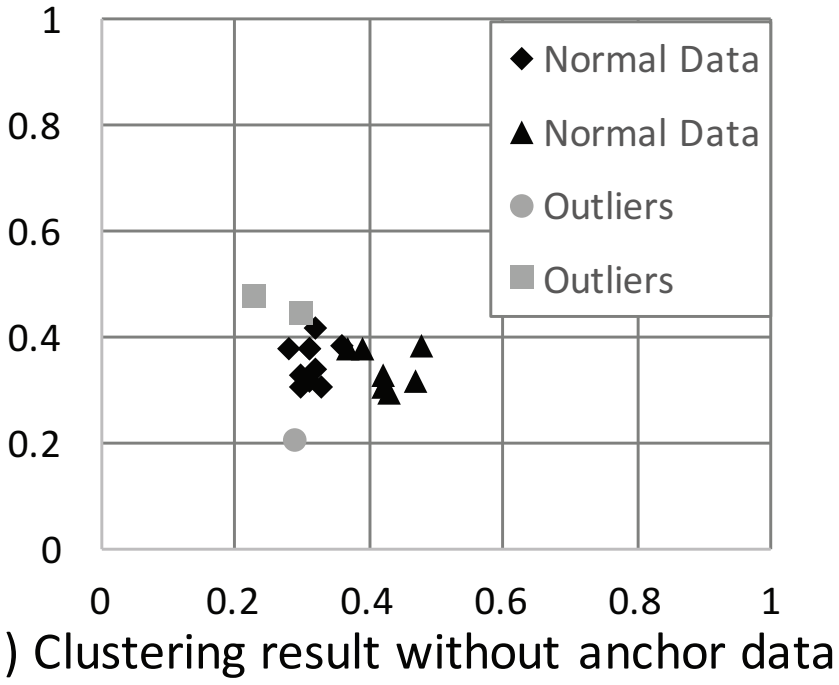

Figure 4.11: Clustering results with and without anchor data.

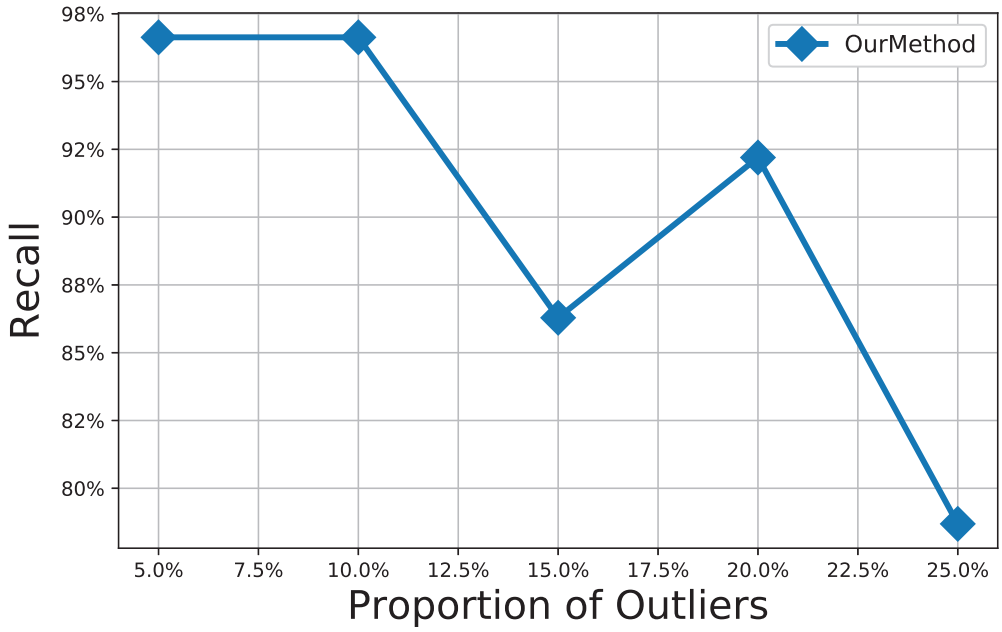

Figure 4.10: Simulation results for Recall on Synthetic Datasets

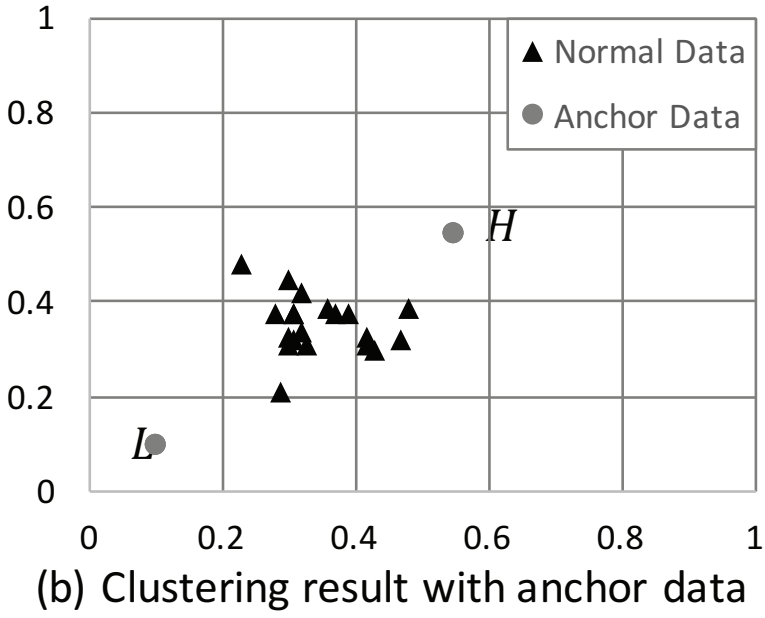

Figure 4.12: Clustering results with and without anchor data.

Chapter 5

A Peak Searching Based Cluster Method

We propose a new peak searching algorithm (PSA) that uses Bayesian optimization to find probability peaks in a dataset, thereby increasing the speed and accuracy of clustering algorithms. Wireless sensor networks (WSNs) are becoming increasingly common in a wide variety of applications that analyze and use collected sensing data. Typically, the collected data cannot be directly used in modern data analysis problems that adopting machine learning techniques, because such data lacks additional information (such as data labels) specifying its purpose of users. Clustering algorithms that divide the data in a dataset into clusters are often used when additional information is not provided. However, traditional clustering algorithms such as expectation-aximization (EM) and k-means algorithms require massive numbers of iterations to form clusters. Processing speeds are therefore slow, and clustering results become less accurate because of the way such algorithms form clusters. The PSA addresses these problems, and we adapt it for use with the EM and k-means algorithms, creating the modified PSEM and PSk-means algorithms. Our simulation results show that our proposed PSEM and PSk-means algorithms significantly decrease the required number of clustering iterations (by 1.99 to 6.3 times), and produce clustering that, for a synthetic dataset, is 1.69 to 1.71 times more accurate than it is for traditional EM and enhanced k-means (k-means++) algorithms. Moreover, in a simulation of WSN applications aimed at detecting outliers, PSEM correctly identified the outliers in a real dataset, decreasing iterations by approximately 1.88, and PSEM was 1.29 times more accurate than EM in maximum.

5.1 Introduction

Over the past decade, Wireless sensor networks (WSNs) have been widely applied in applications that involve analyzing collected data to improve quality of life or secure

property. For example, sensor nodes are present in homes, vehicle systems, natural environments, and even satellites and outer space. These sensors collect data for many different purposes, such as health monitoring, industrial safety and control, environmental monitoring, and disaster prediction [4], [52] [87] [117]. In such WSN applications, sensing data can be manually or automatically analyzed for specific purposes. However, in the age of big data, an increasing amount of sensing data is required for precise analysis in the WSN applications. Consequently, it is difficult or, in some cases, even impossible to manually analyze all of the collected data.

There are several conventional ways to automatically manage the collected data. The most typical and the easiest method is to set threshold values that correspond to sensing events. Events are triggered once the data exceed these thresholds. However, the thresholds in large-scale WSNs vary, and chances due to environment changes. Moreover, precise analysis results cannot be obtained through the use of thresholds alone.

A complementary approach uses supervised machine learning. In this approach, a model is trained that can categorize sensing data into the different states required by an application. However, because sensing data labels are required in the training phase, extra work is required to manage the data. This process is particularly difficult when the dataset is large. Moreover, if the sensing environment changes, certain labels must also change. It is difficult to maintain a functional model under conditions where labels change frequently; this affects the analysis results.

Unsupervised machine learning methods are feasible and well-studied, and are not associated with the data labeling problems described above. Clustering is an important and common method in such approaches. In clustering, the overall features of the dataset are extracted. Then, the data are divided into clusters according to their features. As a result, data labeling is not required, and the data-labeling difficulties that occur in supervised approaches can be avoided. However, in state-of-the-art clustering methods such as the $expectation\text{-}maximization$ (EM) [38] and k-means [58] algorithms, a massive number of iterations must be performed in order to form clusters, and a significant amount of computation time is required. Furthermore, because these algorithms use random start data points as initial center points to form clusters, and because the number of clusters is not precisely determined, the clustering results become less accurate. To address these problems, in this paper, we propose a peak searching algorithm (PSA) for improving clustering algorithm capabilities.

Our approach should be applicable to different dataset distributions. Therefore, the collected sensing dataset is considered to be generated by a Gaussian mixture model composed of several different Gaussian distributions. If the number of Gaussian distributions and appropriate initial center points are known, clustering algorithms can appropriately divide the dataset into different clusters, because each Gaussian distribution corresponds

to a cluster. The proposed PSA employs a Bayesian optimization (BO) strategy that uses a Gaussian process [98]. Bayesian optimization is typically used for hyper-parameter optimizations; to the best of our knowledge, our approach is the first to use BO to improve clustering.

Given a collected dataset, the PSA searches for the data points with the highest probability values (i.e., peaks in the dataset). A Gaussian distribution peak is a point that corresponds to the mean. By searching the peaks, we can obtain appropriate initial center points of Gaussian distributions, hence, the corresponding clusters. This method overcomes the difficulties associated with the hard determination of starting data points in traditional cluster algorithms, thereby reducing the number of iterations. By using the PSA, cluster algorithms can form clusters using peak points instead of random start points, which improves the clustering accuracy.

We used simulations to investigate the potential of the proposed PSA for improving algorithm performance. To measure performance improvements, we applied the PSA to the EM and k-means algorithms. We refer to these modified algorithms as $PSEM$ and $PSk\text{-}means$, respectively. The simulation results showed that for PSEM and PSk-means, the required numbers of clustering iterations were significantly reduced by 1.99 to 6.3 times. Additionally, for synthetic datasets, clustering accuracy was improved by 1.69 to 1.71 times relative to the traditional EM and enhanced version of k-means, i.e., k-means++ [2].

The proposed method can accurately group data into clusters. Therefore, any outliers in a dataset can be clustered together, making them possible to identify. Because outliers obviously reduce the capabilities of the WSN applications, we also conducted a simulation using a real WSN dataset from the Intel Berkeley Research lab. This allowed us to compare the outlier-detection capabilities of $PSEM$ and EM. Our simulation results showed that PSEM correctly identified outliers, decreased iterations by approximately 1.88 times, and improved accuracy by 1.29 times in maximum.

5.2 Bayesian Optimization

Before a dataset can be divided into clusters, the starting data points of clusters in the dataset must be determined. In particular, the number of peak points (a peak point is a data point corresponding to the maximum probability) in a dataset corresponds to the number of clusters. In this study, we use BO to identify peak points. Typically, we do not know the form of the probability density function $p(\boldsymbol{x})$. Nevertheless, we can obtain the approximate value $f(\boldsymbol{x})$ of $p(\boldsymbol{x})$ at data point $\boldsymbol{x}$, with some noise. For example, we can approximately compute the density of a certain volume. This density is an approximate value of the probability density (see Sec. 4). following subsection, we introduce the Gaussian process

used in BO. However, obtaining the maximum density can be computationally expensive, because of the large number of data points. To reduce computation costs, we used BO [30] [24], a very powerful strategy that fully utilizes prior experience to obtain the maximum posterior experience at each step. This allows the maximum density to be approached. Thus, fewer data points are required to obtain the maximum density. In the following subsection, we introduce the Gaussian process used in BO.

5.2.1 Gaussian Process

In BO, a Gaussian process (GP) is used to build a Gaussian model from the provided information. The model is then updated with each new data point. Assume that a set of data points contains t elements: $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_t\}$. We use the notation $\boldsymbol{x}_{1:t}$ to represent the set of data points. Each of these points exists in a $D\text{-}dimensional$ space. An example data point is $\boldsymbol{x}_i = (x_{i1}, \cdots, x_{iD})$.

There is an intuitive analogy between a Gaussian distribution and a GP. A Gaussian distribution is a distribution over a random variable. In contrast, the random variables of a GP are functions. The mean and covariance are both functions. Hence, function $f(\boldsymbol{x})$ follows a GP and is defined as follows:

$$f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')) \tag{5.1}$$

where $m(\boldsymbol{x})$ is the mean function, and $k(\boldsymbol{x}, \boldsymbol{x}')$ is the kernel function of the covariance function.

Suppose that we have a set of data points $\boldsymbol{x}_{1:t}$ and their corresponding approximate probability density $\{f(\boldsymbol{x}_1), f(\boldsymbol{x}_2), \cdots, f(\boldsymbol{x}_t)\}$. We assume that function $f(\boldsymbol{x}_i)$ can map a data point $\boldsymbol{x}_i$ to its probability density $p(\boldsymbol{x}_i)$ with some noise. For concision, we will use $\boldsymbol{f}_{1:t}$ to represent the set of functions for each data point $\{f(\boldsymbol{x}_1), f(\boldsymbol{x}_2), \cdots, f(\boldsymbol{x}_t)\}$. For the collected dataset, $\mathcal{D}_{1:t} = \{(\boldsymbol{x}_1, f_1), (\boldsymbol{x}_2, f_2), \cdots, (\boldsymbol{x}_t, f_t)\}$ is the given information. For convenience, we assume that $\mathcal{D}_{1:t}$ follows the GP model, which is given by an isotropic Gaussian $\mathcal{N}(\boldsymbol{0}, \boldsymbol{K})$ whose initial mean function is zero and covariance function is calculated using $\boldsymbol{K}$, as follows[1]:

$$\boldsymbol{K} = \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_t) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_t, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_t, \boldsymbol{x}_t) \end{bmatrix}.$$

Once, we have calculated $\boldsymbol{K}$, we build a GP model from the information provided.

[1] $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ consists of the kernel functions.

A new data point $\boldsymbol{x}_{t+1}$ also follows $f_{t+1} = f(\boldsymbol{x}_{t+1})$. According to the GP properties, $\boldsymbol{f}_{1:t}$ and f_{t+1} are jointly Gaussian:

$$\begin{bmatrix} \boldsymbol{f}_{1:t} \\ f_{t+1} \end{bmatrix} = \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} \boldsymbol{K} & \boldsymbol{k} \\ \boldsymbol{k}^T & k\left(\boldsymbol{x}_{t+1}, \boldsymbol{x}_{t+1}\right) \end{bmatrix}\right),$$

where

$$\boldsymbol{k} = \begin{bmatrix} k\left(\boldsymbol{x}_{t+1}, \boldsymbol{x}_1\right) & k\left(\boldsymbol{x}_{t+1}, \boldsymbol{x}_2\right) & \cdots & k\left(\boldsymbol{x}_{t+1}, \boldsymbol{x}_t\right) \end{bmatrix}.$$

Moreover, we want to predict the approximate probability density f_{t+1} of the new data point $\boldsymbol{x}_{t+1}$. Using Bayes' theorem and $\mathcal{D}_{1:t}$, we can obtain an expression for the prediction.

$$P\left(f_{t+1} | \mathcal{D}_{1:t}, \boldsymbol{x}_{t+1}\right) = \mathcal{N}\left(\mu_t(\boldsymbol{x}_{t+1}), \sigma_t^2(\boldsymbol{x}_{t+1})\right) \tag{5.2}$$

where

$$\begin{aligned} \mu_t\left(\boldsymbol{x}_{t+1}\right) &= \boldsymbol{k}^T \boldsymbol{K}^{-1} \boldsymbol{f}_{1:t} \\ \sigma_t^2\left(\boldsymbol{x}_{t+1}\right) &= k(\boldsymbol{x}_{t+1}, \boldsymbol{x}_{t+1}) - \boldsymbol{k}^T \boldsymbol{K}^{-1} \boldsymbol{k}. \end{aligned} \tag{5.3}$$

We can observe that μ_t and σ_t^2 are independent of f_{t+1} and, and that we can calculate f_{t+1} using the given information.

5.2.2 Acquisition Functions for Bayesian Optimization

Above, we briefly describe how to use the given information to fit a GP and update the GP by incorporating a new data point. At this point we must select an appropriate new data point $\boldsymbol{x}_{i+1}$ to use to update the GP, so that we can obtain the maximum value of $f(\boldsymbol{x}_{i+1})$. To achieve this, we could use BO to realize exploitation and exploration. Here, exploitation means that we should use the data point with the maximum mean in the GP, because that point fully uses the given information. However, this point cannot provide additional information about the unknown space. Exploration means that a point with a larger variance in the GP can provide additional information about the unknown area. The acquisition functions used to find an appropriate data point are designed on the basis of exploitation and exploration. There are three popular acquisition functions: probability of improvement, expectation of improvement, and upper confidence bound criterion.

The probability of improvement (PI) function is designed to maximize the probability of improvement over $f(\boldsymbol{x}^+)$, where $\boldsymbol{x}^+ = \mathrm{argmax}_{\boldsymbol{x}_i \in \boldsymbol{x}_{1:t}} f(\boldsymbol{x}_i)$ The resulting cumulated distribution function is:

$$\begin{aligned} PI(\boldsymbol{x}) &= P\left(f(\boldsymbol{x}) \geq f(\boldsymbol{x}^+) + \xi\right) \\ &= \Phi\left(\frac{\mu(\boldsymbol{x}) - f(\boldsymbol{x}^+) - \xi}{\sigma(\boldsymbol{x})}\right) \end{aligned} \tag{5.4}$$

where ξ is the exploration strength, which is provided by the user.

The expectation of improvement (EI) is designed to account for not only the probability of improvement, but also the potential magnitude of improvement that could be yielded by a point. The EI is expressed as

$$EI(\boldsymbol{x}) = \begin{cases} (\mu(\boldsymbol{x}) - f(\boldsymbol{x}^+) - \xi)\,\Phi(Z) + \sigma(\boldsymbol{x})\phi(Z) & if\ \sigma(\boldsymbol{x}) > 0 \\ 0 & if\ \sigma(\boldsymbol{x}) = 0 \end{cases} \tag{5.5}$$

$$Z = \begin{cases} \frac{\mu(\boldsymbol{x}) - f(\boldsymbol{x}^+) - \xi}{\sigma(\boldsymbol{x})} & if\ \sigma(\boldsymbol{x}) > 0 \\ 0 & if\ \sigma(\boldsymbol{x}) = 0 \end{cases} \tag{5.6}$$

The upper confidence bound (UCB) criterion uses the confidence bound, which is the area representing the uncertainty between the mean function and variance function in Eq. 5.3. The UCB is compared with the other two acquisition functions, and is relatively simple and intuitive. In detail, it directly uses the mean and variance functions obtained from the given information. A potential new data point is presented by the sum of (i) the mean function, and (ii) a constant ν times the variance function. That is, given several potential new data points, the data point with the largest UCB will be selected as the next new data point. Moreover, ν, which is greater than 0, indicates how much exploration is expected. The UCB formula is

$$\mathrm{UCB}(\boldsymbol{x}) = \mu(\boldsymbol{x}) + \nu\sigma(\boldsymbol{x}) \tag{5.7}$$

These three acquisition functions are suited to different datasets, and allow us to obtain an appropriate new data point. The BO algorithm is shown below.

Algorithm 5: BO

for $i = 1, 2, \ldots$ **do**
  Fit a GP to the given information $\mathcal{D}_{1:t}$;
  Use acquisition functions to find a data point $\boldsymbol{x}$ that has the maximum value $\mu(\boldsymbol{x}|\mathcal{D}_{1:t})$ over GP;
  Calculate the value of $f(\boldsymbol{x})$ at $\boldsymbol{x}_i$;
  Augment the dataset $\mathcal{D}_{1:t+1} = \{\mathcal{D}_{1:t}, (\boldsymbol{x}_i, f_i)\}$ and update the GP;
end

5.3 Peak Searching Algorithm

In this section, we first introduce some preliminary information related to our proposed algorithms. Then, we explain the algorithm.

5.3.1 Preliminary Investigations

In most cases, the environment can be represented as a collection of statuses that indicate whether or not certain events have occurred. Such events include fires, earthquakes, and invasions. The data points collected by the sensor nodes contain measurements that describe the statuses of these events. One can assume that the collected dataset is generated by a Gaussian mixture model (GMM), because the data points contained in the dataset are collected from the normal environment, or from natural events. Thus, before fitting a GMM, it is necessary to clarify the peaks of the GMM, because each peak is a point that has the largest probability corresponding to a Gaussian distribution. Therefore, we need to know the probability of each data point when we search for the dataset peaks. Although the probability density function is unknown, it can be approximated using alternative methods, which are shown as follows.

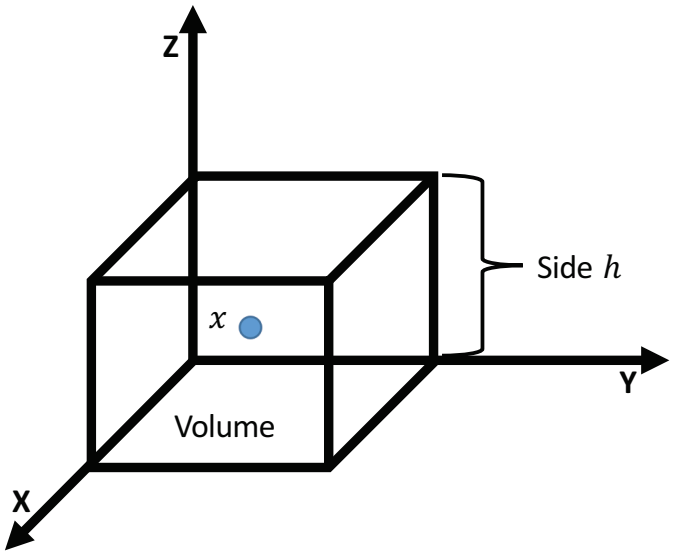

Figure 5.1: A volume in $3-dimensional$ space

One type of method assumes that the set of data points exists in a D-$dimensional$ space. The probability of data point $\boldsymbol{x}$ can then be approximated as follows. (i) Set $\boldsymbol{x}$ as the center of a volume with side h. Figure 5.1 shows an example of a volume in 3-D space, where the length of each side is h. (ii) The density of the volume with center $\boldsymbol{x}$, calculated using Eq. (5.8) [24], is approximately equal to the probability at data point $\boldsymbol{x}$. The density $p(\boldsymbol{x})$ in this formula depends on the length of side h in the volume and the number T_h (that is, the number of neighbors of data point $\boldsymbol{x}$ in the volume). N is the total number of data points in the dataset and h^D is the size of the volume. Thus, to search for the peaks, we must calculate the densities of all of the different data points using Eq. (5.8). However, this is computationally expensive.

$$p\left(\boldsymbol{x}\right) = \frac{T_h}{Nh^D} \tag{5.8}$$

Another method fixes h and applies a kernel density estimator [24]. In this case, the probability of data point $\boldsymbol{x}$ can be calculated as

$$p(\boldsymbol{x}) = \frac{1}{Nh^D} \sum_{i=1}^{T} \boldsymbol{K}\left(\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h}\right), \tag{5.9}$$

where $\boldsymbol{K}(\bullet)$ is the kernel function and T is the number of data points in a volume with side h. Then, the largest value of $p(\boldsymbol{x})$ occurs along the gradient of Eq.(5.9), which is

$$\nabla p(\boldsymbol{x}) = \frac{1}{Nh^D} \sum_{i=1}^{T} \boldsymbol{K}'\left(\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h}\right). \tag{5.10}$$

By setting Eq. (5.10) equal to zero, we can calculate the point along the gradient that has the largest $p(\boldsymbol{x})$. With this method, we do not need to search the unimportant data points, which reduces the time required to identify peaks. However, Eq. (5.9) and Eq. (5.10) are difficult to solve. Moreover, the length of side h affects the peak search results. Firstly, it supposes that all of the volumes are the same size, because they have the same h. Second an inappropriate h value will lead to an incorrect result. In particular, h values that are too large cause over-smoothing in high-density areas, while h values that are too small cause significant noise in low-density areas. To overcome these shortcomings, we introduce the PSA, which we describe in the following subsection.

5.3.2 The Algorithm

We propose a peak searching algorithm (PSA) that does not consider parameter h. We will use simulations to investigate the details of the PSA, which can be used to improve the speed and accuracy of clustering algorithms such as EM and k-means.

In Eq. (5.10), $\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h}$ is a vector that starts at point $\boldsymbol{x}$ and ends at neighboring point $\boldsymbol{x}_i$. Because a kernel function is used to calculate the inner product of the vectors, in this case the inner product is equal to the length of vector. Moreover, it calculates the largest $p(\boldsymbol{x})$ and the location of data point $\boldsymbol{x}$ where $\boldsymbol{x}$ on the vector at $\frac{1}{Nh^D}$ times the length of the vector. Therefore, the largest probability for finding the peak lays on this vector. This allows us to concentrate only on the vector, without considering constants $\frac{1}{Nh^D}$ and h. Hence, we propose using $\boldsymbol{V}_{\boldsymbol{x}}$ to represent the vector in the PSA as

$$\boldsymbol{V}_{\boldsymbol{x}} = \sum_{i=1}^{T} \frac{(\boldsymbol{x} - \boldsymbol{x}_i)}{\|(\boldsymbol{x} - \boldsymbol{x}_i)\|} \tag{5.11}$$

In Eq. 5.11, only $\boldsymbol{V}_{\boldsymbol{x}}$ is searched. A significant amount of non-important space is not searched. However, many probabilities must be calculated along $\boldsymbol{V}_{\boldsymbol{x}}$. Moreover, because there are too many data points on the vector $\boldsymbol{V}_{\boldsymbol{x}}$, it becomes impossible to search for the best data point with the largest probability in a limited amount of time. Hence, we apply

BO when searching for the largest probability along $\boldsymbol{V}_{\boldsymbol{x}}$. BO optimizes the method for searching the maximum probability value mentioned in Algorithm 1. However, as we mentioned in Sect. 3, the form of probability function $p(\boldsymbol{x})$ is not known, and it can instead be represented by an approximate probability function, which is $f(\boldsymbol{x})$ in Algorithm 1 line 4. Therefore, in this paper, we use Eq. (5.8) to calculate the approximate probability function, which we use in the proposed algorithm. Eq. (5.8) is simpler and more practical for finding dataset peaks. The following describes the details of the proposed PSA.

Algorithm 6: PSA

1 Given a starting data point $\boldsymbol{x}^{(j)}$, and calculate the $\boldsymbol{V}_{\boldsymbol{x}}^{(j)}$;
2 $i = 0$;
3 **while** $True$ **do**
4 Search for Max $p\left(\boldsymbol{x}_i^{(j)}\right)$ along $\boldsymbol{V}_{\boldsymbol{x}}^{(j)}$ by using Algorithm 1 ;
5 Set $\boldsymbol{x}_i^{(j)}$ as a peak and calculate $\boldsymbol{V}_{\boldsymbol{x}_i}^{(j)}$ with $\boldsymbol{x}_i^{(j)}$'s K neighbors;
6 Search for Max $p\left(\boldsymbol{x}_{i+1}^{(j)}\right)$ along $\boldsymbol{V}_{\boldsymbol{x}_i}^{(j)}$ by using Algorithm 1 ;
7 **if** $\left|p\left(\boldsymbol{x}_i^{(j)}\right) - p\left(\boldsymbol{x}_{i+1}^{(j)}\right)\right| < \epsilon$ **then**
8 $\boldsymbol{x}_{i+1}^{(j)}$ is a peak of the dataset ;
9 break ;
10 **else**
11 Set $\boldsymbol{x}_{i+1}^{(j)}$ as a peak and calculate $\boldsymbol{V}_{\boldsymbol{x}_{i+1}}^{(j)}$ with $\boldsymbol{x}_{i+1}^{(j)}$'s T neighbors ;
12 $\boldsymbol{V}_{\boldsymbol{x}}^{(j)} \leftarrow \boldsymbol{V}_{\boldsymbol{x}_{i+1}}^{(j)}$;
13 $i \leftarrow i + 1$;
14 **end**
15 **end**

Next, we will explain how the PSA works in accordance with Algorithm 2. The initializing step requires a number of starting data points from which to begin the search for peaks, because the dataset may contain multiple peaks. Therefore, the PSA randomly selects M starting points, $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \cdots, and \boldsymbol{x}^{(M)}\}$. For convenience, we will use starting point $\boldsymbol{x}^{(j)}$ to describe the details of the method. Vector $\boldsymbol{x}^{(j)}$ is calculated using Eq. (5.11) in line 1. The peak searching process shown in Fig. 5.2 contains four steps. In Step 1, the PSA uses Algorithm 1 to search for the peak. That is, data point $\boldsymbol{x}_i^{(j)}$, which has a maximum probability along $\boldsymbol{V}_{\boldsymbol{x}}^{(j)}$. The probability denoted by $p\left(\boldsymbol{x}_i^{(j)}\right)$ is calculated using Eq. (5.8) as shown in line 4. In Step 2 in line 5, a new vector $\boldsymbol{V}_{\boldsymbol{x}_i}^{(j)}$ is calculated on the basis of $\boldsymbol{x}_i^{(j)}$ and its T neighboring data points. In Step 3, the method searches for the peak $\boldsymbol{x}_{i+1}^{(j)}$ along $\boldsymbol{V}_{\boldsymbol{x}_i}^{(j)}$ in line 6. Notice that data points $\boldsymbol{x}_i^{(j)}$ and $\boldsymbol{x}_{i+1}^{(j)}$ are possible dataset peaks. Step 4 starts from line 7 to 14, and the method repeats these steps until the difference

between $p\left(\boldsymbol{x}_i^{(j)}\right)$ and $p\left(\boldsymbol{x}_{i+1}^{(j)}\right)$ gets close enough to zero. At this point, data point $\boldsymbol{x}_{i+1}^{(j)}$ is selected as a dataset peak. The same four steps are used with the other starting data points to identify all peaks in the dataset.

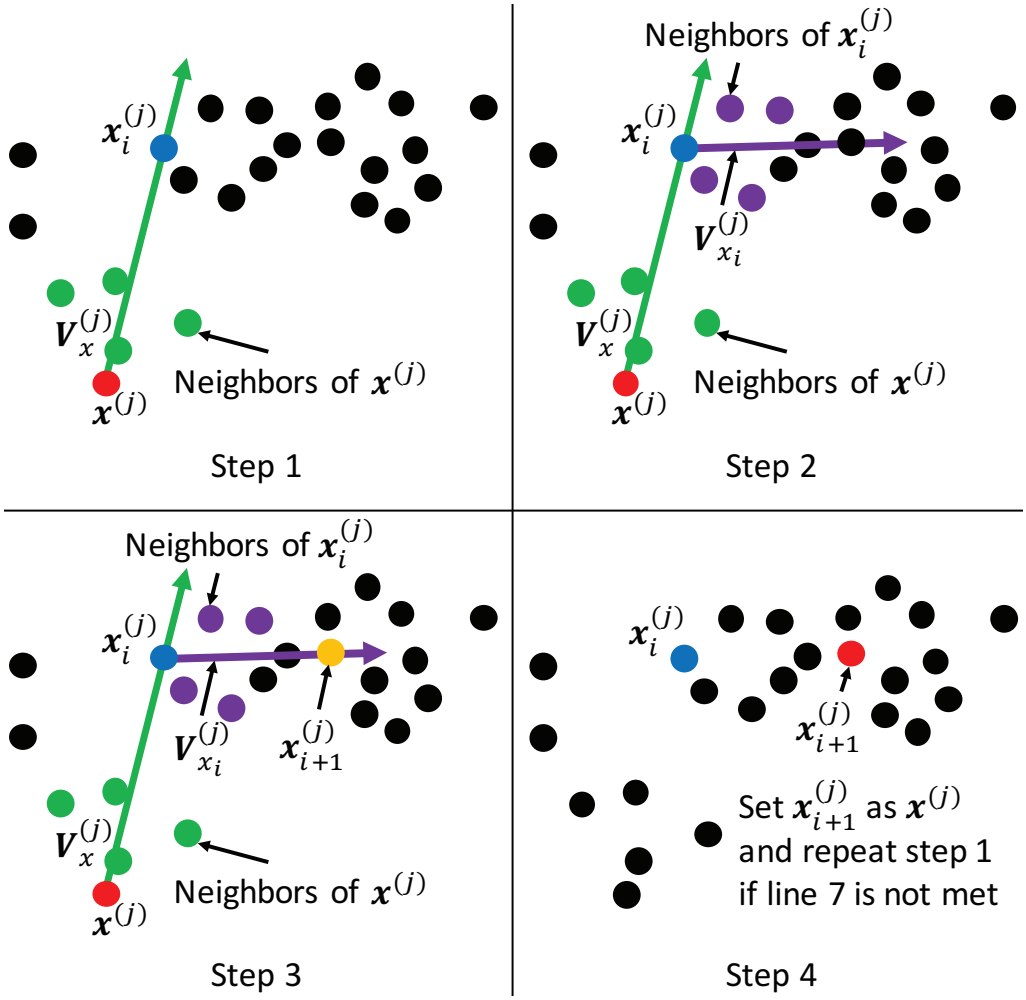

Figure 5.2: Peak searching

5.3.3 Peak Searching Test

In the following, we test this PSA algorithm to search the peak of a dataset that is a Gaussian mixture model that has the peaks of $(1, 1)$ and $(3, 3)$. For concision, we only use one starting data point to search the peak of one Gaussian. We also want to identify the affection of several parameters, which are the number of neighbors and the number of test data point during Gaussian process.

Case 1

In the case 1, we set the number of neighbors is 20, and at first the number of test points is 10. The result is shown in the following. We can see that the PSA randomly selected a staring point $(1.94, 3.66)$ and executes 4 times to find the final peak $(2.93, 3.04)$. The searched peak is very close to the true peak. During the peak searching, the $Vector$ that is a blue line changes its direction towards to the peak of $(3, 3)$, and there are 10 test points on the $Vector$. The diamond on this vector is a candidate peak before the searching process end.

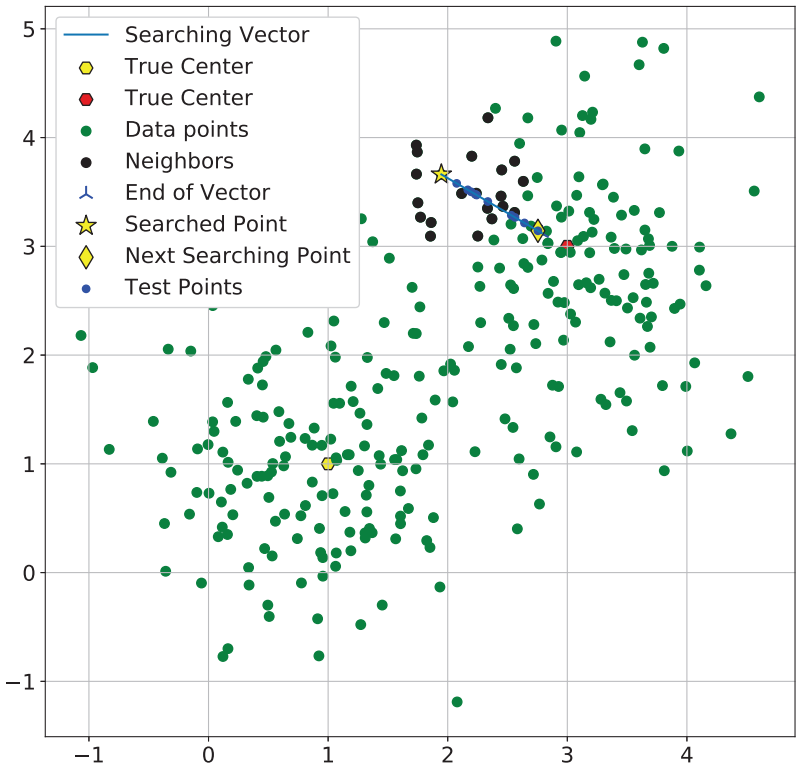

Figure 5.3: Peak searching, 20 neighbors and 10 test points (step 1)

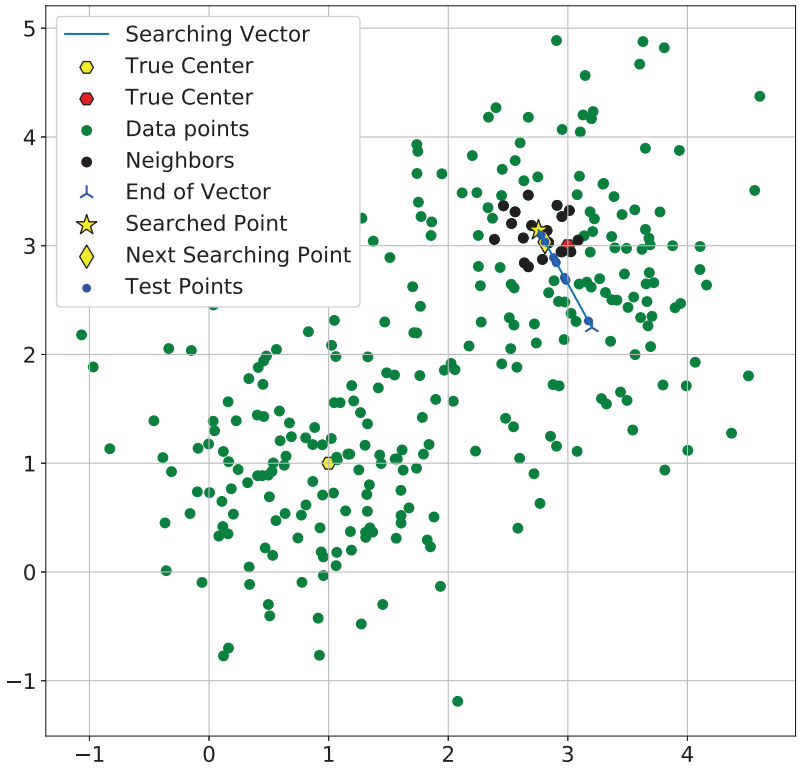

Figure 5.4: Peak searching, 20 neighbors and 10 test points (step 2)

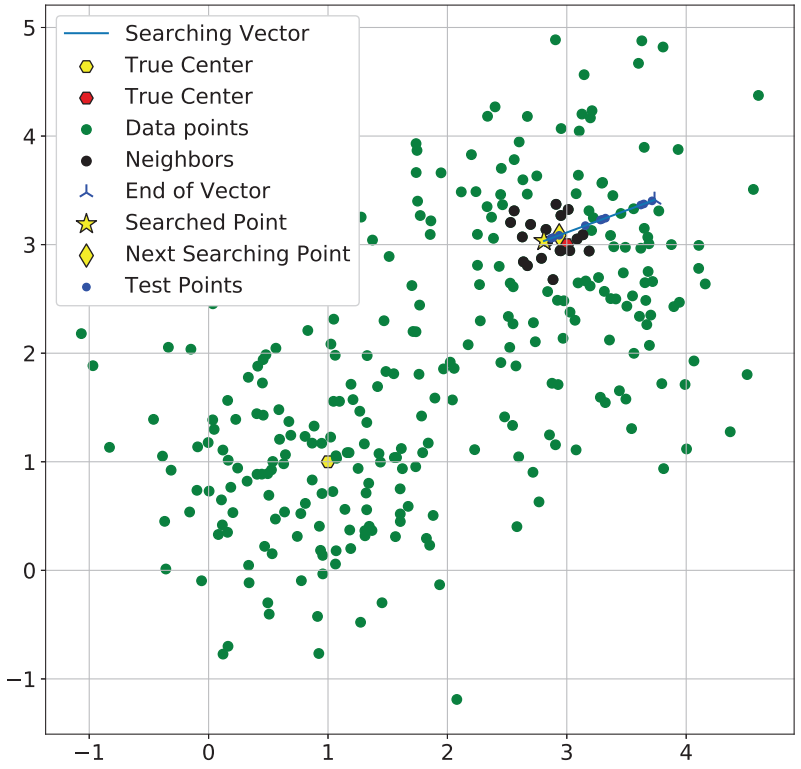

Figure 5.5: Peak searching, 20 neighbors and 10 test points (step 3)

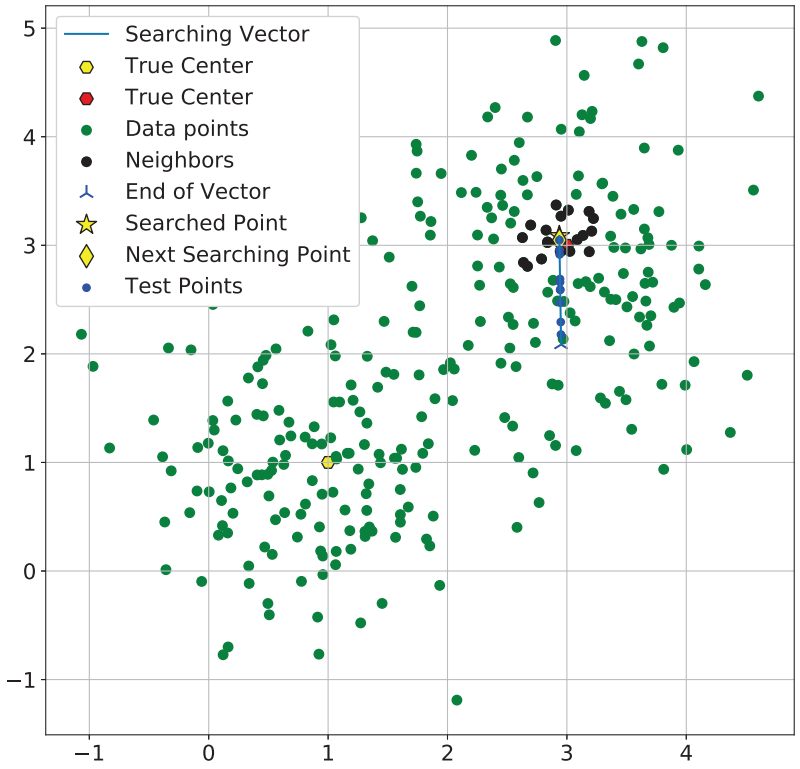

Figure 5.6: Peak searching, 20 neighbors and 10 test points (step 4)

Case 2

In the case 2, we only change the number of test point from 10 to 20. The PSA randomly selected a data point at $(3.703, 2.35)$, and it executes two times to find the peak at $(3.05, 3.01)$. The searched peak is much more close to the true peak. This result shows a more test data points may provide a more accuracy searching result. However, the drawback is that the calculation of PSA increased. Therefore, we think during the real application of PSA how to decide the number of test data is an issue for the WSNs, because the resource constraint sensor nodes have not enough batter.

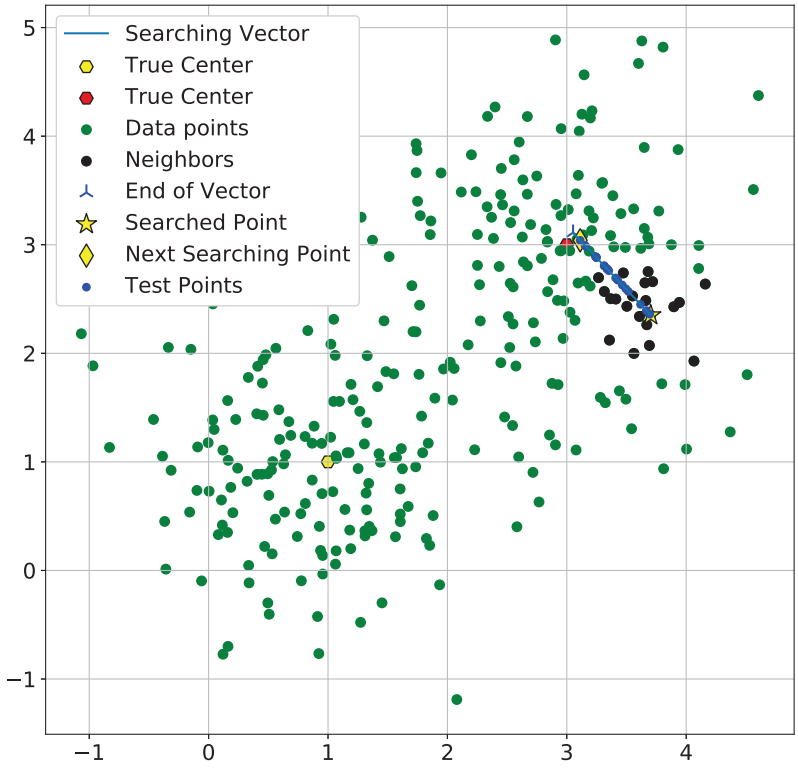

Figure 5.7: Peak searching, 20 neighbors and 20 test points (step 1)

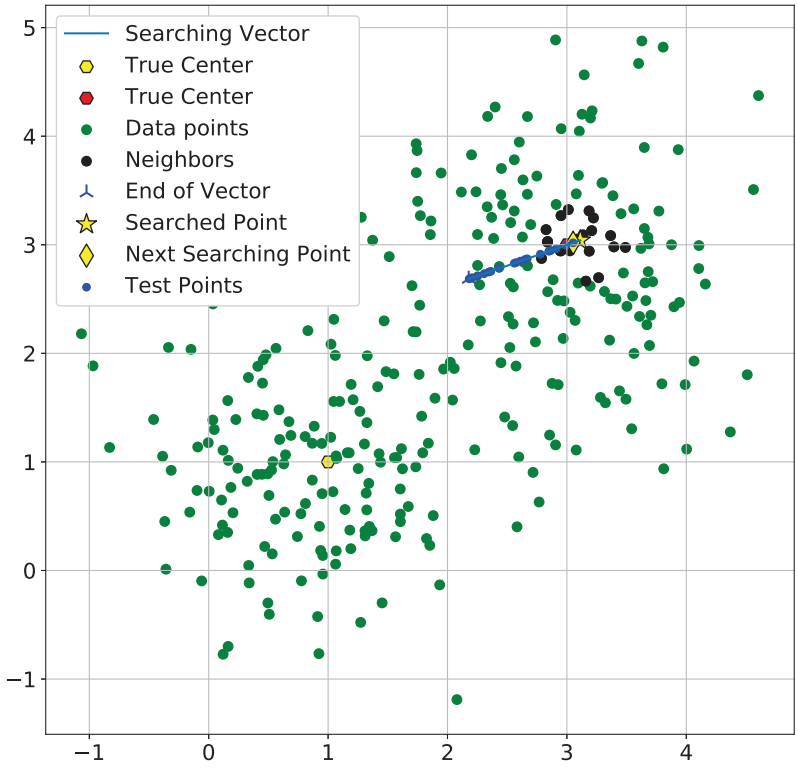

Figure 5.8: Peak searching, 20 neighbors and 20 test points (step 2)

Case 3

In the case 3, we want to identify the affection of the neighbors because the Vector is decided by the neighbors, which is very important during the peak searching process of PSA. Intuitively, more neighbors can provide more informations of the dataset. On the other hand, more data points may bring more noises of the dataset. Therefore, the number of neighbors is also very important for this PSA. The following results show that we using 20 test data points and different number of neighbors. In the first one, we using 60 neighbors, and the PAS randomly selected a point at $(3.58, 3.80)$ and it stops at $(3.10, 2.94)$. The searching processes are as follows.

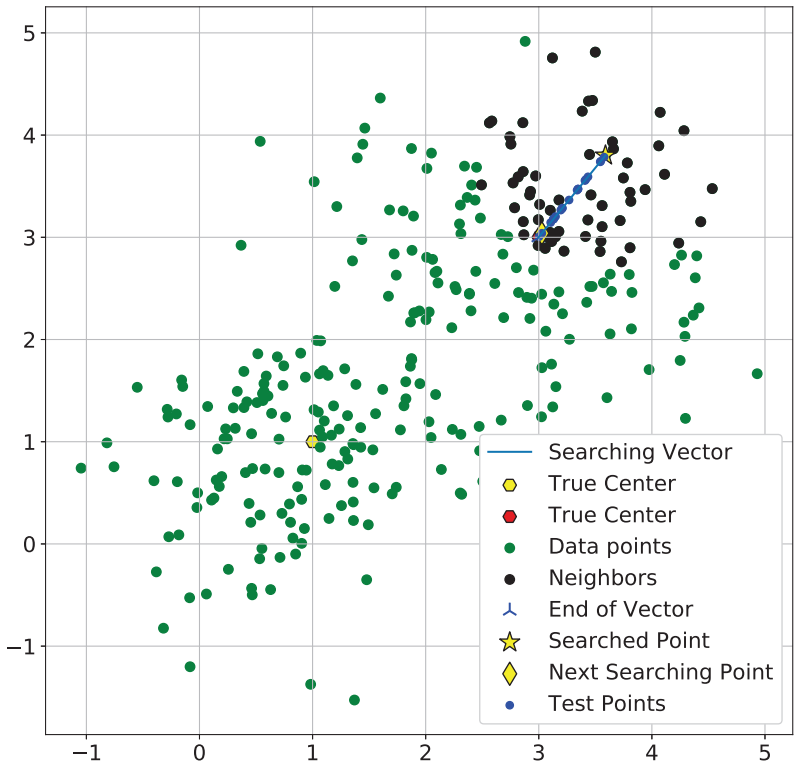

Figure 5.9: Peak searching, 60 neighbors and 20 test points (step 1)

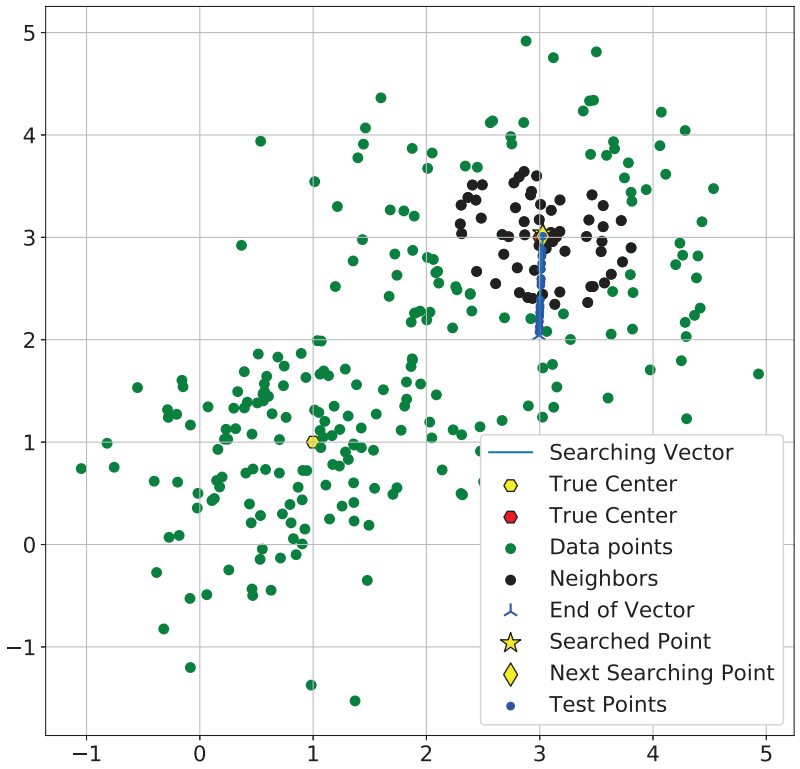

Figure 5.10: Peak searching, 60 neighbors and 20 test points (step 2)

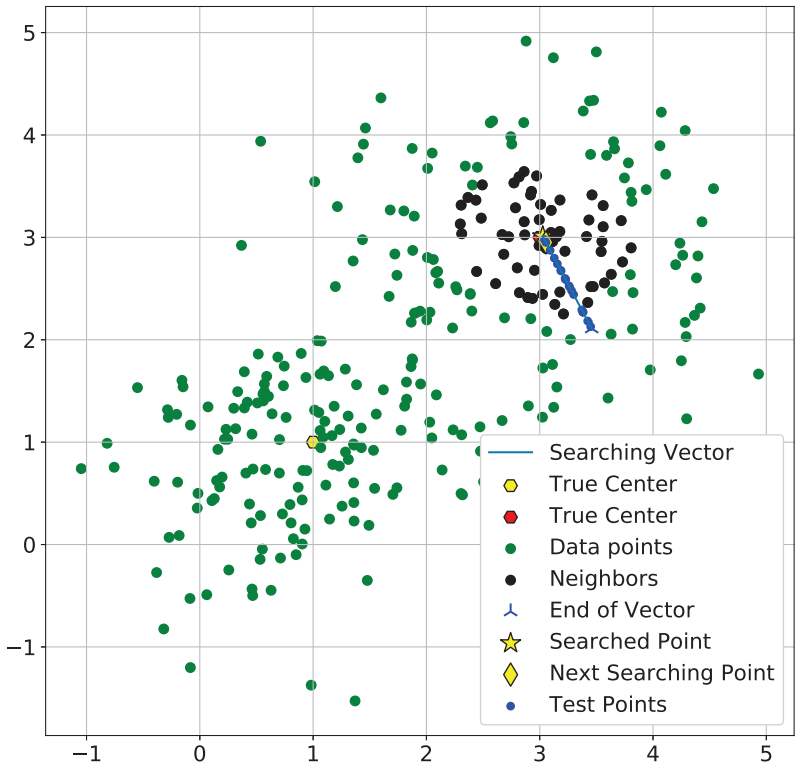

Figure 5.11: Peak searching, 60 neighbors and 20 test points (step 3)

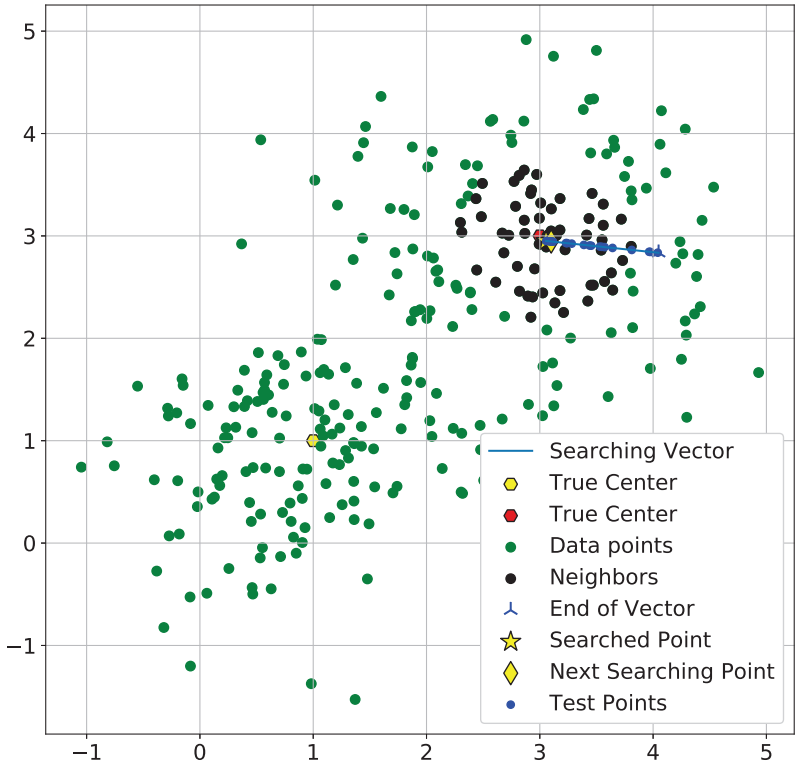

Figure 5.12: Peak searching, 60 neighbors and 20 test points (step 4)

Case 4

In the last case, we reduce the number of neighbors to 10 and keep the same number of test point. The PAS randomly select a data point at $(2.38, 2.45)$ and the searched peak is $(3.10, 2.99)$. Comparing with case 2 and case 3 the accuracy of peak searching result is not changes too much. According to these results of cases, we infer that the PSA has a robust property about the number of neighbor. The peak searching process of case 4 is shown as follows.

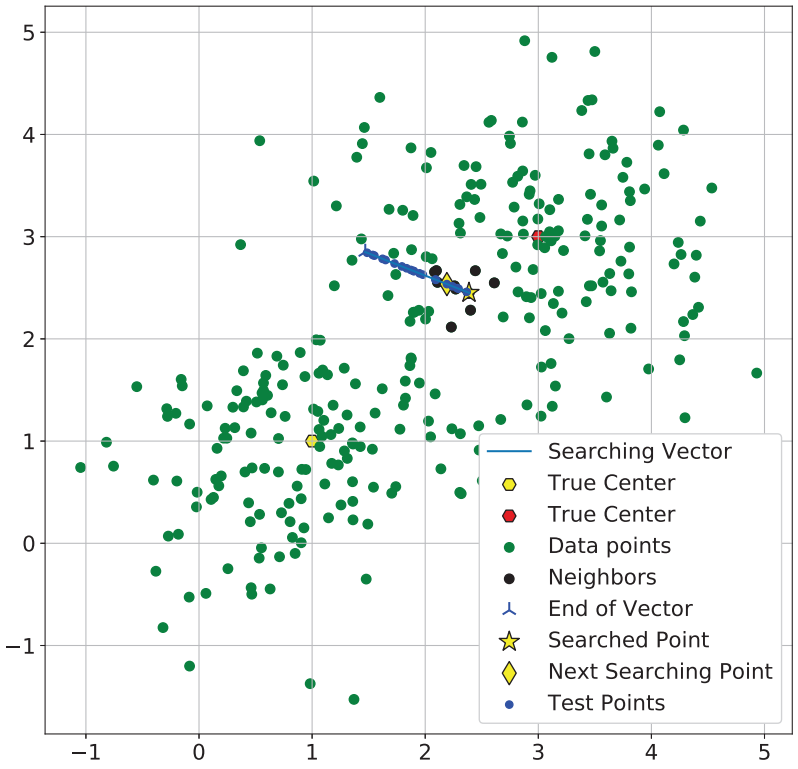

Figure 5.13: Peak searching, 60 neighbors and 20 test points (step 1)

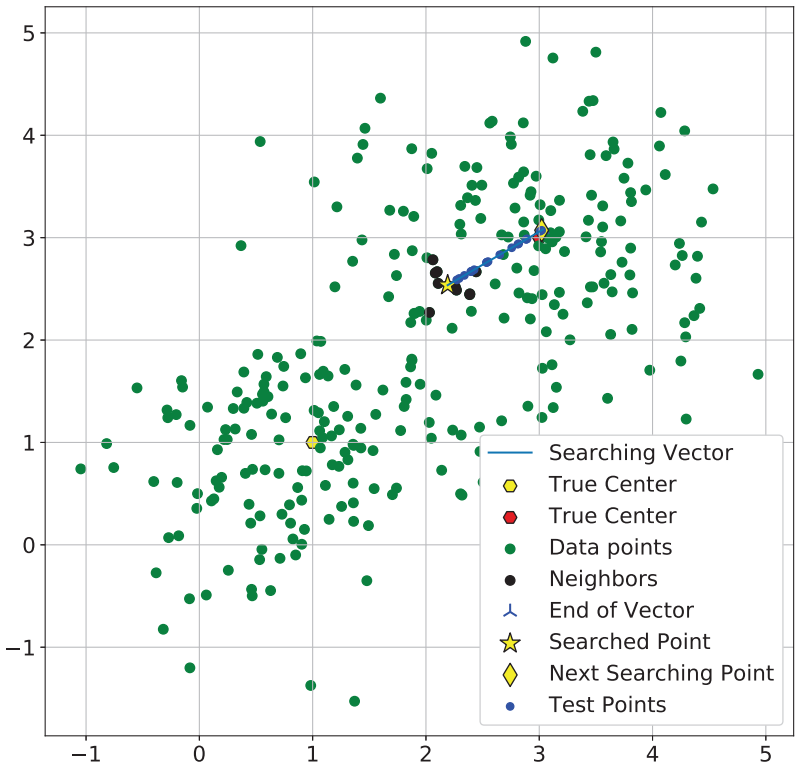

Figure 5.14: Peak searching, 60 neighbors and 20 test points (step 2)

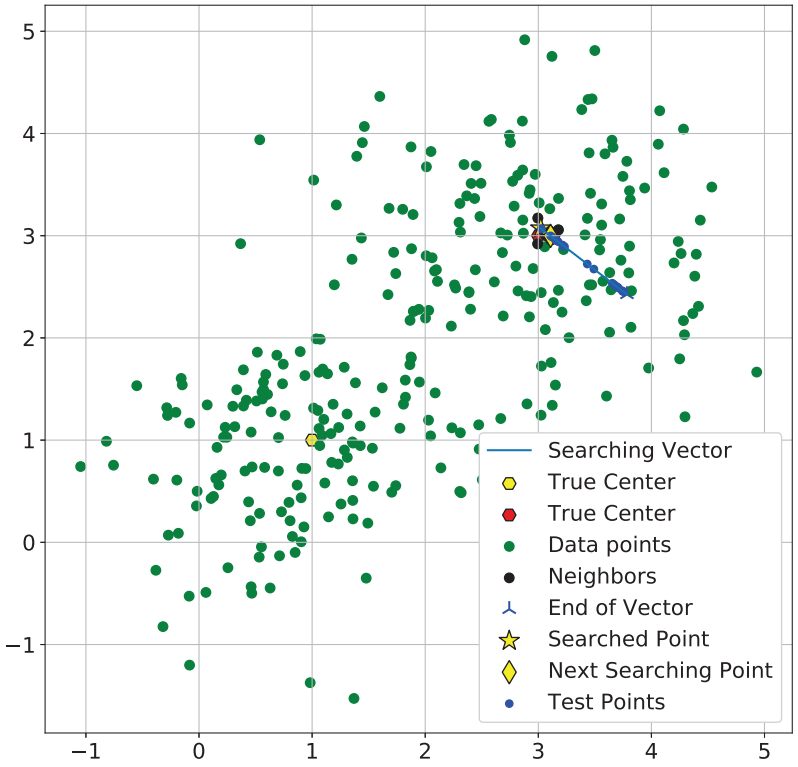

Figure 5.15: Peak searching, 60 neighbors and 20 test points (step 3)

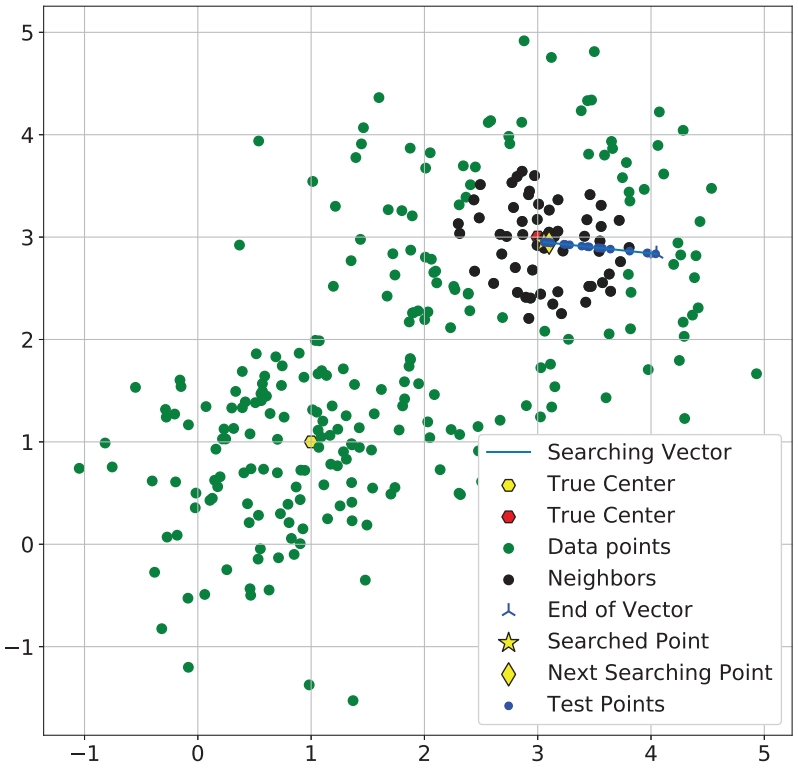

Figure 5.16: Peak searching, 60 neighbors and 20 test points (step 4)

5.4 Simulation and Analysis

In this section, we investigate the efficiency of the proposed PSA. Because the PSA is a method for improving clustering algorithms, we must use it in state-of-the-art clustering algorithms to evaluate the extent to which the PSA can improve those algorithms. As mentioned in Sect. 2, EM and $k\text{-}means$ are common clustering algorithms. Here, variations of those algorithms using the PSA are referred to as $PSEM$ and $PSk\text{-}means$, respectively. In $PSEM$ and $PSk\text{-}means$, the PSA first searches the peaks of the collected dataset. Then, EM and $k\text{-}means$ use the obtained peaks as the initial starting points to start clustering. In the simulations, we assume that the collected datasets follow GMMs, and that the number of peaks found by the PSA is equal to the number of Gaussian distributions.

We conducted simulations using synthetic datasets and a real dataset. In simulations with synthetic datasets, we compared the accuracies and iterations of $PSEM$ and $PSk\text{-}means$ with those of the original EM (OEM), $k\text{-}means$, and $k\text{-}means$++ algorithms. Moreover, because recall and precision are important evaluation indicators, we also used the simulations to compare recalls and precisions. In the simulation using a real dataset, we simulated our methods in order to detect outliers. Because a real dataset could be either isotropic or anisotropic, and because $k\text{-}means$ has a weak effect on anisotropic datasets, we only compared $PSEM$ to OEM for the real dataset.

5.4.1 Simulation on Synthetic Datasets

Synthetic Dataset

We generated two synthetic datasets, whose data points contained two features. Each dataset was generated using a GMM that contained two different Gaussian distributions. The Gaussian distributions in the first dataset were isotopically distributed; their true peaks (means) were $(1, 1)$ and $(2, 2)$ and their variances were 0.6 and 0.5, respectively. The Gaussian distributions in the second synthetic dataset were transformed using the following matrix to create anisotropic ally distributed datasets:

$$\begin{bmatrix} 0.6 & -0.6 \\ -0.4 & 0.8 \end{bmatrix}.$$

The two synthetic datasets are shown in Fig. 5.17. The two synthetic datasets are appropriate for these types of simulations, because they can represent both easy and difficult clustering situations. This allows us to evaluate the effects of our algorithm.

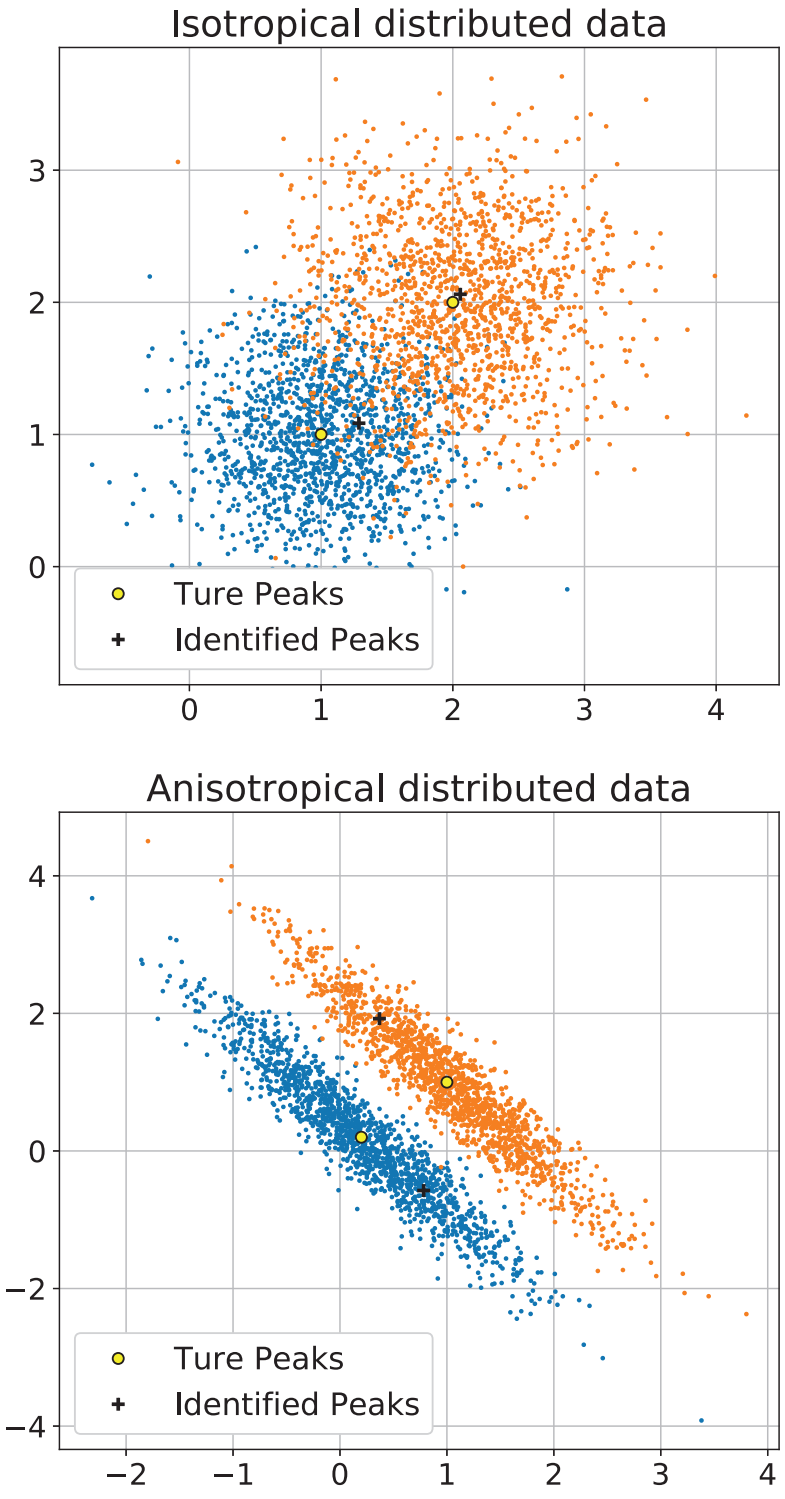

Figure 5.17: Synthetic Dataset

Simulations and Results

To estimate the extent to which the PSA can improve clustering capabilities, we compared $PSEM$ with the original EM (OEM) algorithm. Both $PSEM$ and OEM use EM to

fit a GMM, and have a time complexity of $O(N^3)$, where N is the number of data points. Hence, we cannot use time complexity to compare $PSEM$ and EM. Computational efficiency can also be measured from the number of iterations. The EM algorithm contains two steps: the E-step and the M-step. These two steps are iteratively executed to fit a GMM, and are the core calculations of this algorithm. Hence, we compared the number of iterations in $PSEM$ (i.e., how many E-steps and M-steps were executed) with the number of iterations in OEM. Note that the OEM algorithm does not use PSA, so its calculations start at randomly selected initial starting points.

$PSEM$ and OEM were executed 200 times for the two different datasets. Fig. 5.17 shows 200 peak searching results for PSA. The dark crosses indicate the peaks identified by PSA. We can see that in the isotropically distributed dataset, the identified peaks are very close to the true peaks. In the anisotropically distributed dataset, the identified peaks are also close to the true peaks. Fig. 5.18 illustrates the number of iterations (y-axis) for each size of dataset (x-axis). In the peak searching step, three different acquisition functions are used (UCB, EI, and PI), and their calculation efficiencies are compared. According to the results shown in Fig. 5.18, there were 3.06 to 6.3 times fewer iterations for $PSEM$ than OEM. In other words, the PSA improved the calculation efficiency of OEM by 73.9% to 86.3%. Moreover, we can see that there is no obvious difference between the three acquisition functions.

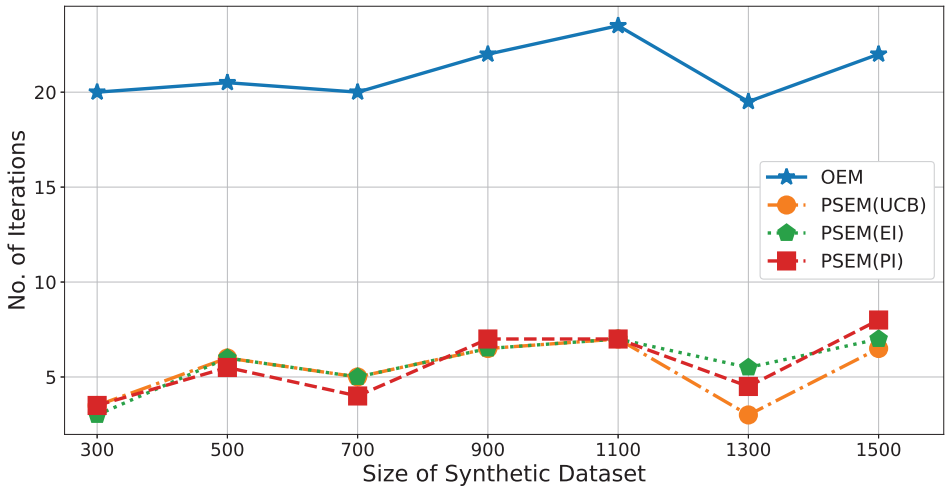

Figure 5.18: Comparison of iterations: OEM

Because we wanted to fairly estimate the extent to which the proposed PSA improves clustering capabilities, we compared the PSA to k-$means$++ in another simulation. k-$means$++ uses a special method to calculate its initial points, and its clustering method increases the speed of convergence. Note that both PSk-$means$ and k-$means$++ are

based on k-$means$, which has a time complexity $O(N^2T)$, where N is the number of data points and T is the number of iterations. Similarly, we cannot use time complexity to compare calculation efficiencies. However, we can compare the number of iterations required for PSk-$means$ to that required for k-$means$++. Both of these algorithms were executed 200 times with the two different datasets, and the results are shown in Fig. 5.19.

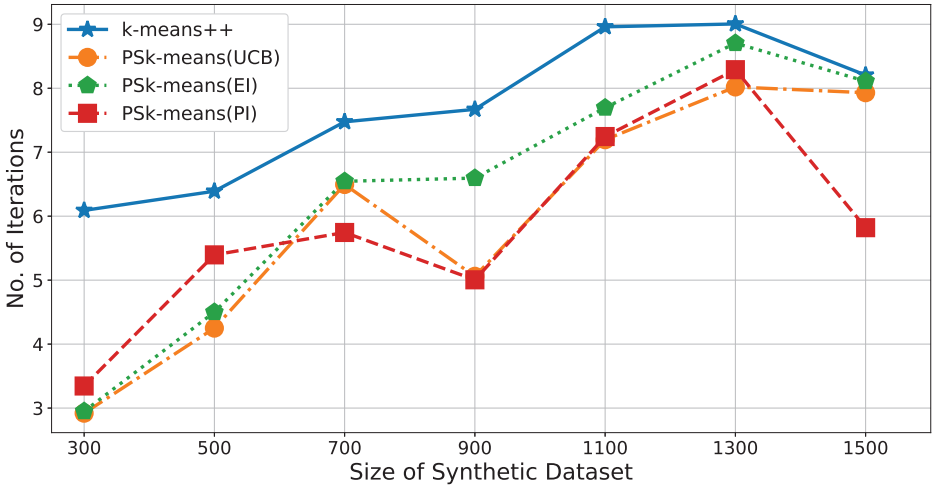

Figure 5.19: Comparison of iterations: k-$means$++

The simulation results are shown in Fig. 5.19. The average number of iterations for PSk-$means$ is reduced by 1.04 to 1.99 times compared with the number of iterations for k-$means$++. In other words, the PSA improved the calculation efficiency of OEM by 51% to 67%. Additionally, there was no obvious difference between the three acquisition functions.

Performance Estimation of Clustering

$Accuracy$, $precision$, and $recall$ are three commonly used measurements for estimating machine learning algorithm performance. Therefore, we adopt these measurements to quantify the performances of our proposed algorithm. In simulations, a dataset containing two clusters is generated by GMM. To explain these measurements, we assume that the two clusters are cluster A and cluster B. Data points belonging to cluster A are considered to be positive instances, while those that belong to cluster B are considered to be negative instances. If a data point from cluster A is correctly clustered into cluster A, it is a true positive (TP) result. Otherwise, it is a false positive result (FP). Similarly, if a data point from cluster B is correctly clustered into cluster B, that is a true negative (TN); otherwise,

it is a false negative (FN). Overall $accuracy$ can be calculated as follows:

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN}. \tag{5.12}$$

$Recall$ is equal to the ratio of TP to the total number of positive instances. It is based on the total positive instances, and shows how many positive instances can be detected by the algorithm. It is calculated as

$$recall = \frac{TP}{TP + FN}. \tag{5.13}$$

From a prediction standpoint, $Precision$ indicates how many TPs occur in the detected positive instances. It presents the proportion of TP to the total number of data points that are detected as positive, which is equal to TP + FP. $Precision$ is calculated as

$$precision = \frac{TP}{TP + FP}. \tag{5.14}$$

We estimated the $accuracy$, $precision$, $recall$ and $F1 - score$ of the $PSk\text{-}means$ and $PSEM$ clustering algorithms, and compared the values with those for $k\text{-}means$, $k\text{-}means$++, and OEM. We repeated this estimation 200 times for each dataset; the average accuracy of each algorithm is shown in Figs. 5.20 and 5.21

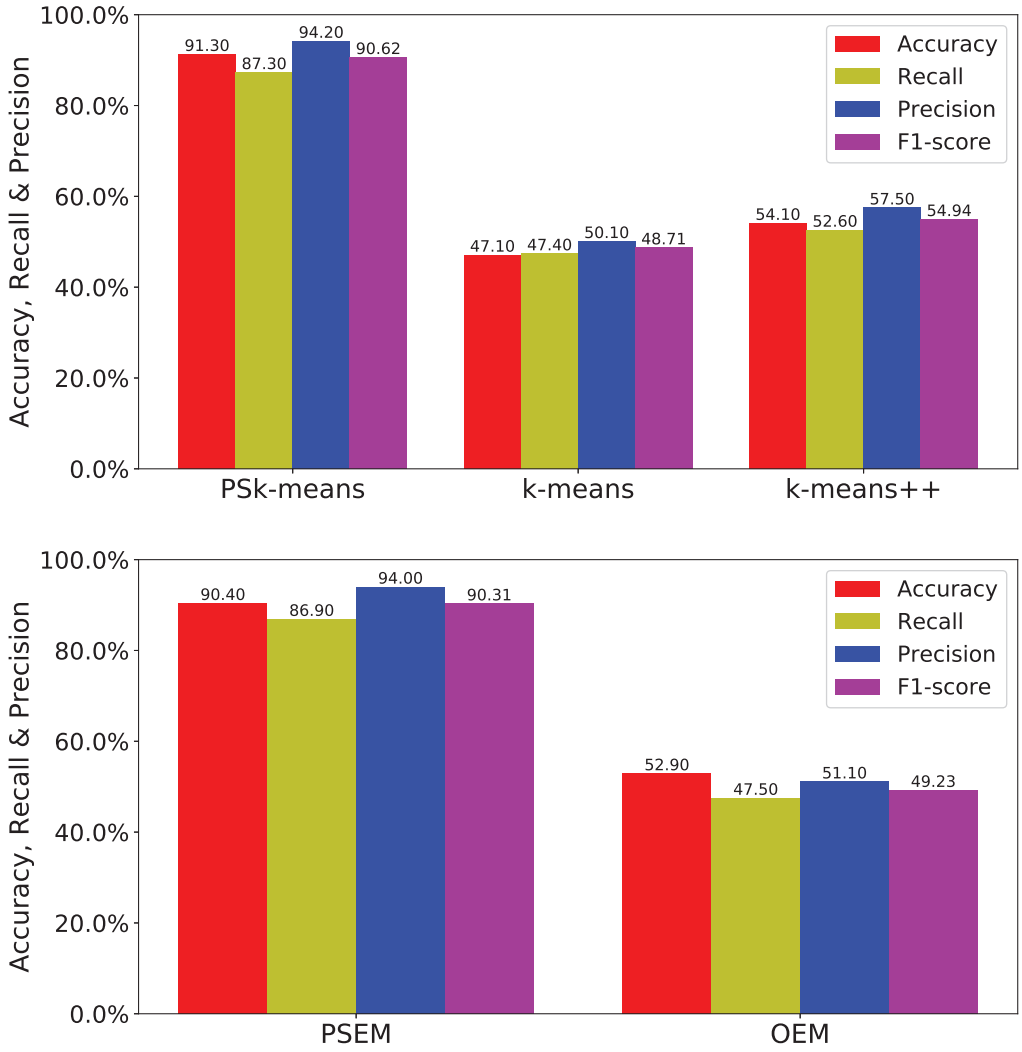

Figure 5.20: Measurements for isotropic dataset

The isotropic datasets shown in Fig. 5.17 are difficult to cluster, because the two clusters partially overlap and their centers are very close together. We can see from the simulation results shown in Fig. 5.20 that the estimations of k-$means$, k-$means$++, and OEM are similar. However, PSk-$means$ and $PSEM$ show a great improvement over their original algorithms. The accuracy of PSk-$means$ is 1.69 times higher than that of k-$means$++, while that of $PSEM$ is 1.71 times higher than that of OEM. The recall of PSk-$means$ is 1.66 times higher than that of k-$means$++, and the recall of $PSEM$

is 1.83 times higher than that of OEM. Moreover, the precision of $PSk\text{-}means$ is 1.64 times higher than that of $k\text{-}means$++'s. The precision of $PSEM$ is 1.84 times higher than that of OEM. Moreover, the $F1-score$ of $PSEM$ is 1.83 times higher than that of OEM. The $F1-score$ of $PSk-\text{-}means$ is 1.86 times higher than that of $k\text{-}means$, and 1.64 times higher than that of $k\text{-}means$++.

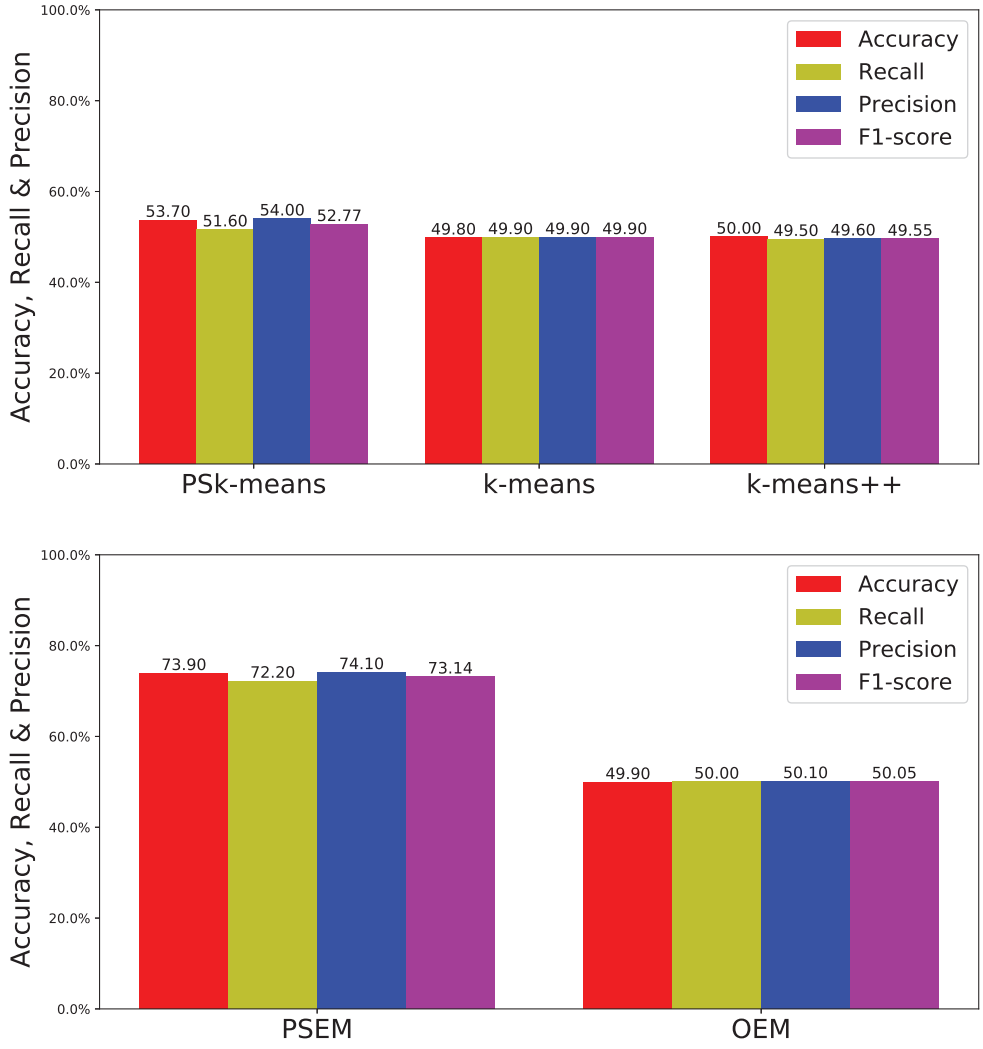

Figure 5.21: Measurements for anisotropic dataset

The results for the anisotropic datasets are shown in Fig. 5.21. Because the anisotropic datasets are elliptical, as shown in Fig. 5.17, and the two datasets are very close together,

the datasets are very difficult to cluster. As a result, $k\text{-}means$ and $k\text{-}means$++ exhibit low estimation performance, and $PSk\text{-}means$ yields little improvement. However, the accuracy of $PSEM$ was 1.48 times higher than that of OEM, and its recall and precision were 1.44 and 1.48 times higher, respectively, than they were for OEM. Moreover, the $F1 - score$ of $PSEM$is 1.46 times higher than that of OEM. The $F1 - score$ of $PSk\text{-}means$ is 52.77 that is higher than that of $k\text{-}means$ and $k\text{-}means$++, respectively

Accordingly, we can see that the PSA can improve clustering accuracy.

5.4.2 Simulation on a Real Dataset from Intel Berkeley Research Laboratory

We used a real sensor dataset from the Intel Berkeley Research Laboratory[7] to assess outlier detection performance. In the simulation, we only considered two features for each data point: temperature and humidity. Each sensor node contained 5000 data points.

Because the original dataset did not provide any outlier information or labels, we manually cleaned the data by removing values that fell outside a normal data range. All of the remaining data points were considered to be normal. Table 5.1 lists the normal data ranges.

Table 5.1: Normal data ranges

	Range	Average
Temperature ($^{\circ}C$)	21.32 - 28.14	23.14
Humidity ($\%$)	26.39 - 44.02	37.69

After completing this step, a uniform distribution was used to generate artificial outliers. Temperature outliers were generated within a range of $(27\text{-}30)^{\circ}C$, and humidity outliers were generated within a range of $(42\text{-}46)\%$. Thus, some outliers can fall inside the normal range with the same probability. Outliers were then inserted into the normal dataset. We produced four different cases, in which the outliers accounted for 5%, 15%, 20%, and 25% of the total normal data points.

Setting of WSNs

$PSEM$ and OEM were run for a real dataset from the Intel Berkeley Research Laboratory. There were 54 sensor nodes, each of which had a $Mica2Dot$ sensor for collecting humidity, temperature, light, and voltage values. Temperatures were provided in degrees Celsius. Humidity was provided as temperature-corrected relative humidity, and ranged from

0-100%. Light was expressed in Lux (1 Lux corresponds to moonlight, 400 Lux to a bright office, and 100,000 Lux to full sunlight), and voltage was expressed in volts, ranging from 2-3. The batteries were lithium ion cells, which maintain a fairly constant voltage over their lifetime; note that variations in voltage are highly correlated with temperature. We selected data from 10 sensor nodes (nodes 1 to 10) to test our method, and used only humidity and temperature values.

In this simulation, we assumed that the WSN was hierarchical and consisted of classes.[2] Each class contained one class head (CH) and other member sensor nodes (MSNs). The MSNs sent the data points collected over a certain time period to the CH, which used the proposed method to monitor whether the dataset collected from its members contained outliers. The configuration of the WSNs is shown in Fig. 5.22.

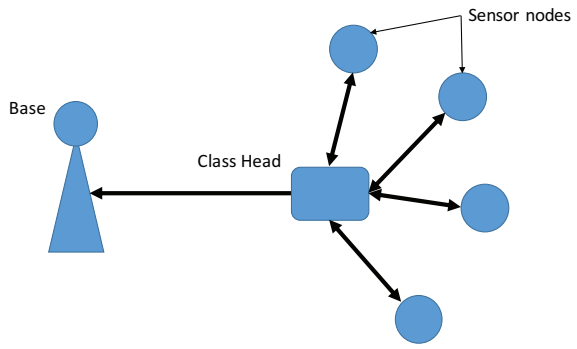

Figure 5.22: WSN configuration

Results

Using the real dataset, we tested the proposed $PSEM$ and compared it with the OEM. The CH executed the $PSEM$ or OEM to detect outliers, and sent outlier reports to the base station. We generated four different datasets, containing 5%, 15%, 20%, and 25% outliers.

It was relatively easy to detect outliers in the test dataset containing only 5% outliers, because the proportion of outliers was so low. Thus, the accuracy of our method approached 100% for 5% outliers. In contrast, the accuracy of the OEM was only approximately 85%. In the other datasets, more outliers fell within the normal dataset. In such cases, it was difficult to detect the outliers; the accuracies of both methods decreased as the proportion of outliers increased. However, $PSEM$ remained more accurate than OEM. In the worst case, with 25% outliers in the test dataset, its accuracy of $PSEM$ was approximately 80%, while the accuracy of OEM was only approximately 60%. That is, $PSEM$ was about

[2]"Cluster" is used in the WSNs to describe a group of sensor nodes. However, "cluster" can also refer to a group of similar data points in data mining. In this paper, we use "class" instead of cluster to describe a group of sensor nodes.

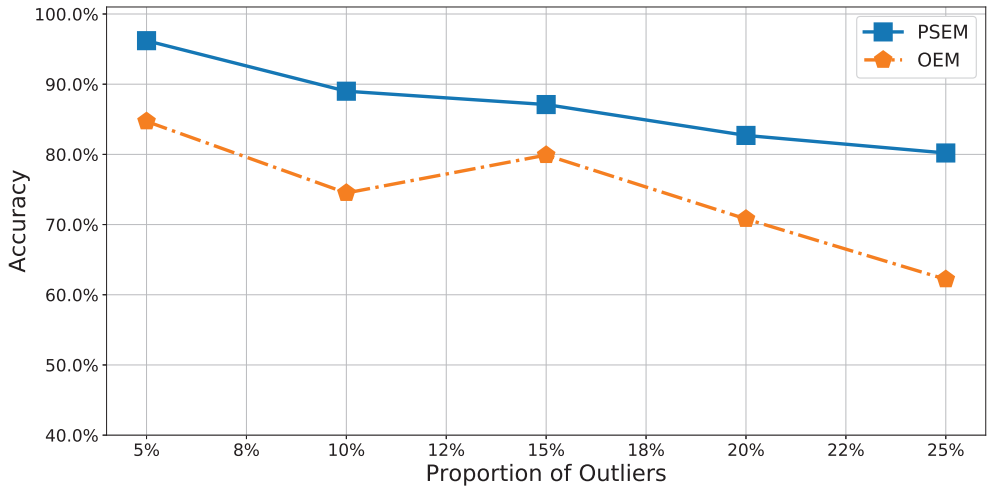

Figure 5.23: Accuracy of real dataset

1.09 to 1.29 times more accurate than OEM. Moreover, Fig. 5.24 shows the number of iterations was 1.52 to 1.88 times lower for $PSEM$, meaning that $PSEM$ improved the calculation efficiency of OEM by 60% to 65.2%. Because accuracy and iteration numbers are very important metrics for assessing the clustering algorithm efficiency, this simulation result demonstrated the practical significance of $PESM$, and therefore, of the PSA.

Simulation Platform

Moreover, we use a raspberry pi 3B model as a sensor node, which uses a quad core 1.2GHz Broadcom BCM2837 64bit CPU and a 1GB RAM, and we measured the executing time of this peak searching algorithm on it, we also measured the executing times of the same clustering algorithms, and we compare the executing time with k-means and EM that are not using the peak searching algorithm. The executing time is shown in the Table 5.2.

Table 5.2: Executing time on Raspberry pi 3B

Size of Dataset	300	500	700	900	1100	1300	1500
k-means	0.09s	0.12s	0.12s	0.13s	0.14s	0.14s	0.133s
PSk-means	0.011s	0.01s	0.012s	0.007s	0.011s	0.012s	0.008s
EM	0.064s	0.065s	0.072s	0.099s	0.10s	0.051s	0.091s
PSEM	0.03s	0.037s	0.035s	0.039s	0.043s	0.046s	0.047s

According to the measured executing time on raspberry, we can see algorithms using the peak searching algorithm cost less time than that not using the peak searching algorithm.

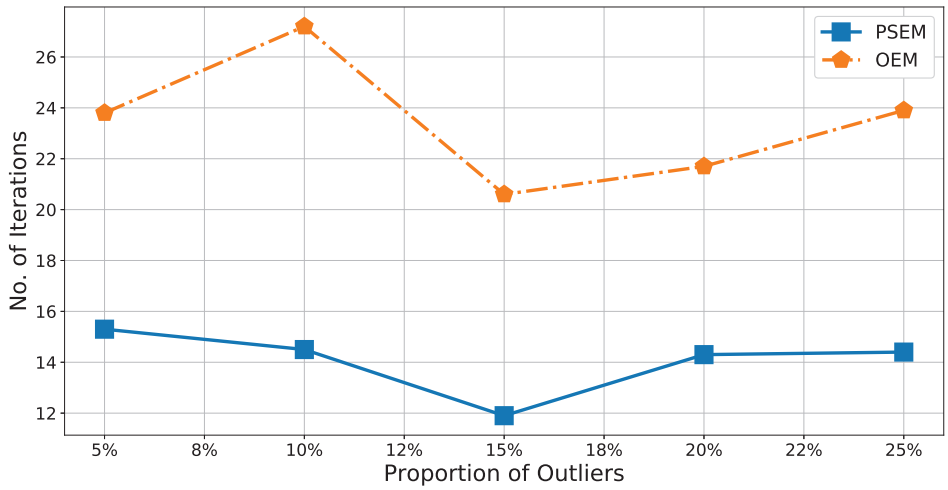

Figure 5.24: No. of iterations for dataset

Therefore, if the sensor nodes have enough extra power support, we consider this peak searching algorithm can be deployed on WSNs to reduce the executing time of clustering.

5.5 Discussions

In this section, we describe other important aspects of the WSNs, such as WSN power consumptions and lifetime. We also discuss the advantages and disadvantages of the proposed method.

Because most sensor nodes in the WSNs are powered by batteries, sensor node power consumptions, WSN lifetime, and energy efficiency are also important problems affecting the quality of a WSN. H. Mostafaei et al. [83] proposed an algorithm PCLA to schedule sensors into active or sleep states, utilizing learning automata to extend network lifetime. Our previous work attempted to extend battery life by reducing peak power consumption. We scheduled sensor execution times [93], and used optimized wireless communication routes to reduce energy consumption, with the goal of prolonging network lifetimes[129][130]. If the proposed PSA can be applied in such approaches to analyze data using clustering methods, then energy consumption can be further reduced. Because the PSA can reduce clustering iterations, the required computational power decreases, leading to energy savings.

The proposed algorithm has advantages and disadvantages. In conventional clustering methods such as EM and k-means, cluster-forming procedures are started at random data points. There are two disadvantages associated with this. First, correct clusters may not be able to form from random starting points. Second, because random staring points may

not occur near cluster centers, massive iterations may be needed to update random points to approach the cluster centers. However, because the PSA can identify the peak points near cluster centers, it is a better approach for forming clusters than an algorithm starting from a random point. Therefore, clustering algorithms using the PSA can form clusters more accurately. Moreover, using peak points as the starting points to form clusters can significantly reduce clustering iterations, because peak points are the desired points.

There are some disadvantages associated with the PSA. The PSA use BO, and are, therefore, affected by the problems associated with BO. A particular issue is that $a\ priori$ design is critical to efficient BO. As mentioned in Sect. 3, BO uses GPs to build Gaussian models with Gaussian distributions, making the resulting datasets transcendental. If a dataset does not have a Gaussian distribution, the PSA may be less efficient. Another weak point of the PSA is that it is centralized. It is not suited for highly distributed WSNs where data analyses are conducted at each sensor node.

5.6 Conclusion

In this paper, we proposed a new PSA for improving the performance of clustering algorithms (i.e., for improving accuracy and reducing clustering iterations). BO is used to search for the peaks of a collected dataset in the PSA. To investigate the efficiency of the PSA, we used the PSA to modify EM and $k\text{-}means$ algorithms. The new algorithms were named $PSEM$ and $PSk\text{-}means$, respectively.

Using simulations, we investigated the performance of $PSEM$ and $PSk\text{-}means$ relative to that of OEM and $k\text{-}means$++. We conducted simulations using both synthetic datasets and a real dataset. For synthetic datasets, $PSEM$ and $PSk\text{-}means$ reduced iterations by approximately 6.3 and 1.99 times, respectively, in maximum. Moreover, they improved clustering accuracy by 1.71 times and 1.69 times, respectively, in maximum. On a real dataset for outliers detection purpose, $PSEM$ reduced iterations about 1.88 times, and improved clustering accuracy by 1.29 times in maximum. These results show that our proposed algorithm significantly improves performance. We obtained the same conclusions by illustrating the recall and precision improvements for $PSEM$ and $PSk\text{-}means$.

In the future, we will improve this method so that it can be used with high-dimensional data, such as images collected by a camera. Moreover, we would like to deploy the peak searching algorithm with sensor nodes, to allow CHs to obtain peak searching results from their neighbors; this will reduce the calculation time required for the peak search. Thus, clustering can be implemented in the sensor node and communication costs can be reduced.

Chapter 6

Conclusion

The purpose of this doctoral thesis is to distribute outliers detection method into each sensor node of a WSN. Therefore, a sensor node can provide a real-time data analysis service. In this doctoral thesis, we adopt machine learning algorithms into WSNs for outliers detection. This thesis contains three main parts. In the first part, we distributed a supervised learning based method into a WSN. In the second part, we developed an unsupervised method based on mean-shift algorithm. The last part, we proposed peak searching algorithm to improve the capability of clustering algorithm.

6.1 A Brief Summary of Each Method

The preliminary experiment deployed a logistic regression algorithm for detecting outliers in WSNs. In detail, we divided this algorithm into learning step and executing step. We deployed the learning step into a sink node and the executing step into every senor node of a WSN. In preliminary experiment, the proposed method can accurately detect outliers. Moreover, we found the logistic regression does not need so many training data. Therefore, we think that this experiment can reduce the computation cost of sensor nodes. However, the weak points of a supervised learning method is that it needs enough training data to establish a model. Moreover, a more important issue is that preparing training data is very time cost and a bad training data cannot provide a good performance of outliers detection. Moreover, training data may lose effectiveness when environment changed. Therefore, we proposed an unsupervised learning based method.

In the first method, we described the necessity for detecting outliers in WSNs and presented an unsupervised learning based outlier detection method to solve this problem. In our method, we first fixed the density of the dataset to utilize the mean-shift algorithm efficiently by using anchor data. Then, the mean-shift algorithm was used to cluster the collected sensing dataset into clusters. As we mentioned that preparing training data is time cost, finally, we proposed a labeling technique to label those clusters as “normal” or

"outliers,"; hence, outliers in the sensing dataset can be detected. In the simulations, we showed the performance of our proposed method and compared our work with related work [121]. The results showed that our method has a lower FPR than that of the related work, and when outliers are far away from the normal data, our method obtained an FPR below 3.3%, which is quite low. Moreover, even in datasets where the distributions of outliers are close to the normal data or a substantial number of outliers are in the dataset, our method can still keep FPR at a low rate. Extended simulations also showed the generality of our method.

In the second method, we proposed a new peak searching algorithm. The algorithm calculates a vector according to kernel density function in order to provide a direction for searching for the peaks of the dataset. We adopted the strategy of Bayesian optimization, which can reduce the computations of the peak search. In simulations, we compared our algorithm with other unsupervised cluster methods, the OEM algorithm and k-means++ algorithm. The results indicated that PSA improves the clustering accuracy and reduces the iterations of the calculation. In particular, PSEM had 70% fewer iterations compared with OEM, while its accuracy improved 1.72 times compared with that of the original algorithm on an isotropic dataset.

6.2 Limitations and Future Works

Our proposed methods are all focused on outliers detection of WSNs and we only tried our algorithms in a 2-dimensional dataset. However, WSNs are used in many purposes, such as object detection and intrusion detection. Many applications usually need a high dimensional data information. In the future, we need to improve our algorithms so that they can deal with high dimensional data, such as images collected by a camera. Moreover, the peak searching algorithm in simulation is deployed in a sink node. We also want to develop a method that can deploy this algorithm into every sensor node, so that the class head can get the peak searching result from its neighbors; this will reduce the calculation time of the peak search. Thus, the clustering can be implemented in the sensor node and thereby reduce the communication cost.

Although cloud computing and edge computing are widely used for resource limited devices, limited bandwidth and weak wireless signal increase the time of data process. Moreover, cloud computing and edge computing are very valuable for big firms because they have enough money to deploy powerful devices and learn characters of their customers. On the other hand, individual will pay less money on devices and they have different demands when using data analysis tools. Therefore, we still pay a lot of attention to develop simple and efficient data process tools that can distributed executing on many resource limited devices.

Moreover, from the QoS perspective of WSNs, to keep the WSN working properly, when outliers in the sensing data are discovered, approaches such as how to tolerate the outliers or how to detect outliers on the sensor node side should be considered. Therefore, part of our future work is methods for tolerating outliers and distributed outlier detection in sensor nodes. Moreover, our method can be used for event detection because outliers are an event in the dataset.

Acknowledgements

First, I would like to express my deep gratitude to my Ph.D advisor, Professor Dr. Yukikazu Nakamoto, for his professional and careful guidance and specialist advices, leading me deeply exploring the exciting unknown word of research. I greatly appreciate the freedom you have given me to find my own path and the guidance and support you offered when needed. At the same time, I am also great appreciative to Professor Dr. Shin for extremely precious advices on my researches. I would also want to thank Professor Ojima for giving me advices of my Pd.D thesis. Moreover, I have to thank Professor Danny during the three years given me a lot of helps when I needed. I would also thank the members in my University for their patience and support in overcoming numerous obstacles I have been facing through my research. I would like to thank my family they give me enormous support and specially my wife, Zhao qian, she gives me a great help and encouragement during the studies.

Bibliography

[1] *Outliers in statistical data*, volume 3. 1994.

[2] k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[3] A taxonomy framework for unsupervised outlier detection techniques for multi-type data sets. Technical report, Centre for Telematics and Information Technology, University of Twente, 2007.

[4] Multi-sensors data fusion system for wireless sensors networks of factory monitoring via bpn technology. *Expert Systems with Applications*, 37(3):2124–2131, 2010.

[5] Joint event detection & identification: A clustering based approach for wireless sensor networks. In *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*, pages 2333–2338. IEEE, 2013.

[6] Towards safety from toxic gases in underground mines using wireless sensor networks and ambient intelligence. *International Journal of Distributed Sensor Networks*, 2013, 2013.

[7] `http://db.csail.mit.edu/labdata/labdata.html`, 2017.

[8] A wsn for monitoring and event reporting in underground mine environments. *IEEE Systems Journal*, 2017.

[9] Abu Zafar Abbasi, Noman Islam, Zubair Ahmed Shaikh, et al. A review of wireless sensors and networks' applications in agriculture. *Computer Standards & Interfaces*, 36(2):263–270, 2014.

[10] Mohammad Abu Alsheikh, Shaowei Lin, Dusit Niyato, and Hwee-Pink Tan. Machine learning in wireless sensor networks: Algorithms, strategies, and applications. *Communications Surveys & Tutorials, IEEE*, 16(4):1996–2018, 2014.

[11] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. *Computer networks*, 38(4):393–422, 2002.

[12] Jamal N Al-Karaki and Ahmed E Kamal. Routing techniques in wireless sensor networks: a survey. *IEEE wireless communications*, 11(6):6–28, 2004.

[13] Rocio Arroyo-Valles, Rocio Alaiz-Rodriguez, Alicia Guerrero-Curieses, and Jesús Cid-Sueiro. Q-probabilistic routing in wireless sensor networks. In *Intelligent Sensors, Sensor Networks and Information, 2007. ISSNIP 2007. 3rd International Conference on*, pages 1–6. IEEE, 2007.

[14] Joshua N Ash and Randolph L Moses. Outlier compensation in sensor network self-localization via the em algorithm. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings.(ICASSP'05). IEEE International Conference on*, volume 4, pages iv–749. IEEE, 2005.

[15] Taiwo Oladipupo Ayodele. Types of machine learning algorithms. In *New advances in machine learning*. InTech, 2010.

[16] Majid Bahrepour, Nirvana Meratnia, and Paul JM Havinga. Sensor fusion-based event detection in wireless sensor networks. In *Mobile and Ubiquitous Systems: Networking & Services, MobiQuitous, 2009. MobiQuitous' 09. 6th Annual International*, pages 1–8. IEEE, 2009.

[17] Majid Bahrepour, Nirvana Meratnia, Mannes Poel, Zahra Taghikhaki, and Paul JM Havinga. Distributed event detection in wireless sensor networks for disaster management. In *Intelligent Networking and Collaborative Systems (INCOS), 2010 2nd International Conference on*, pages 507–512. IEEE, 2010.

[18] Majid Bahrepour, Nirvana Meratnia, Mannes Poel, Zahra Taghikhaki, and Paul JM Havinga. Distributed event detection in wireless sensor networks for disaster management. In *Intelligent Networking and Collaborative Systems (INCOS), 2010 2nd International Conference on*, pages 507–512. IEEE, 2010.

[19] Majid Bahrepour, Berend Jan van der Zwaag, Nirvana Meratnia, and Paul Havinga. Fire data analysis and feature reduction using computational intelligence methods. In *Advances in Intelligent Decision Technologies*, pages 289–298. Springer, 2010.

[20] Richard J Beckman and R Dennis Cook. Outlier. s. *Technometrics*, 25(2):119–149, 1983.

[21] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[22] Luís MA Bettencourt, Aric A Hagberg, and Levi B Larkey. Separating the wheat from the chaff: Practical anomaly detection schemes in ecological applications of distributed sensor networks. In *International Conference on Distributed Computing in Sensor Systems*, pages 223–239. Springer, 2007.

[23] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is nearest neighbor meaningful? In *International conference on database theory*, pages 217–235. Springer, 1999.

[24] C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[25] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[26] Alessio Botta, Walter De Donato, Valerio Persico, and Antonio Pescapé. Integration of cloud computing and internet of things: a survey. *Future Generation Computer Systems*, 56:684–700, 2016.

[27] George EP Box and George C Tiao. *Bayesian inference in statistical analysis*, volume 40. John Wiley & Sons, 2011.

[28] Joel W Branch, Chris Giannella, Boleslaw Szymanski, Ran Wolff, and Hillol Kargupta. In-network outlier detection in wireless sensor networks. *Knowledge and information systems*, 34(1):23–54, 2013.

[29] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *ACM sigmod record*, volume 29, pages 93–104. ACM, 2000.

[30] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[31] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys*, 41(3):15, 2009.

[32] Jinran Chen, Shubha Kher, and Arun Somani. Distributed fault detection of wireless sensor networks. In *Proceedings of the 2006 workshop on Dependability issues in wireless ad hoc networks and sensor networks*, pages 65–72. ACM, 2006.

[33] Yu Sheng Chen, Yu Sheng Qin, Yu Gui Xiang, Jing Xi Zhong, and Xu Long Jiao. Intrusion detection system based on immune algorithm and support vector

machine in wireless sensor network. In *Information and Automation*, pages 372–376. Springer, 2011.

[34] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE transactions on pattern analysis and machine intelligence*, 17(8):790–799, 1995.

[35] Jin-Hee Cho, Ray Chen, and Phu-Gui Feng. Effect of intrusion detection on reliability of mission-oriented mobile group systems in mobile ad hoc networks. *IEEE Transactions on Reliability*, 59(1):231–241, 2010.

[36] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 24(5):603–619, 2002.

[37] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. The variable bandwidth mean shift and data-driven scale selection. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pages 438–445. IEEE, 2001.

[38] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[39] Ethan W Dereszynski and Thomas G Dietterich. Spatiotemporal models for data-anomaly detection in dynamic environmental monitoring campaigns. *ACM Transactions on Sensor Networks (TOSN)*, 8(1):3, 2011.

[40] T Devi, N Saravanan, et al. Development of a data clustering algorithm for predicting heart. *International Journal of Computer Applications*, 48(7), 2012.

[41] Min Ding, Dechang Chen, Kai Xing, and Xiuzhen Cheng. Localized fault-tolerant event boundary detection in sensor networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 2, pages 902–913. IEEE, 2005.

[42] Shaoqiang Dong, Prathima Agrawal, and Krishna Sivalingam. Reinforcement learning based geographic routing protocol for uwb wireless sensor network. In *Global Telecommunications Conference, 2007. GLOBECOM'07. IEEE*, pages 652–656. IEEE, 2007.

[43] Alex HB Duffy. The" what" and" how" of learning in design. *IEEE Expert*, 12(3):71–76, 1997.

[44] Eleazar Eskin. Anomaly detection over noisy data using learned probability distributions. In *In Proceedings of the International Conference on Machine Learning*. Citeseer, 2000.

[45] E Eskinand and S Stolfo. Modeling system call for intrusion detection using dynamic window sizes. In *Proceedings of DARPA Information Survivability Conference and Exposition*, 2001.

[46] Anna Forster and Amy L Murphy. Froms: Feedback routing for optimizing multiple sinks in wsn with reinforcement learning. In *Intelligent Sensors, Sensor Networks and Information, 2007. ISSNIP 2007. 3rd International Conference on*, pages 371–376. IEEE, 2007.

[47] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[48] Keinosuke Fukunaga and Larry Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on information theory*, 21(1):32–40, 1975.

[49] João Gama and Pedro Pereira Rodrigues. Data stream processing. *Learning from Data Streams-Processing Techniques in Sensor Networks*, pages 25–39, 2007.

[50] Carlos F García-Hernández, Pablo H Ibarguengoytia-Gonzalez, Joaquín García-Hernández, and Jesús A Pérez-Díaz. Wireless sensor networks and applications: a survey. *International Journal of Computer Science and Network Security*, 7(3):264–273, 2007.

[51] Shuo Guo, Heng Zhang, Ziguo Zhong, Jiming Chen, Qing Cao, and Tian He. Detecting faulty nodes with data errors for wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 10(3):40, 2014.

[52] Gregory Hackmann, Weijun Guo, Guirong Yan, Zhuoxiong Sun, Chenyang Lu, and Shirley Dyke. Cyber-physical codesign of distributed structural health monitoring with wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 25(1):63–72, 2014.

[53] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[54] Douglas M Hawkins. *Identification of outliers*, volume 11. Springer, 1980.

[55] Robert Hecht-Nielsen et al. Theory of the backpropagation neural network. *Neural Networks*, 1(Supplement-1):445–448, 1988.

[56] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[57] Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial intelligence review*, 22(2):85–126, 2004.

[58] Zhexue Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data mining and knowledge discovery*, 2(3):283–304, 1998.

[59] Krontiris Ioannis, Tassos Dimitriou, and Felix C Freiling. Towards intrusion detection in wireless sensor networks. In *Proc. of the 13th European wireless conference*, pages 1–10. Citeseer, 2007.

[60] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[61] Dharanipragada Janakiram, VA Reddy, and AVU Phani Kumar. Outlier detection in wireless sensor networks using bayesian belief networks. In *Communication System Software and Middleware, 2006. Comsware 2006. First International Conference on*, pages 1–6. IEEE, 2006.

[62] Prem Prakash Jayaraman, Arkady Zaslavsky, and Jerker Delsing. Intelligent processing of k-nearest neighbors queries using mobile data collectors in a location aware 3d wireless sensor network. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 260–270. Springer, 2010.

[63] George H John. Robust decision trees: Removing outliers from databases. In *KDD*, pages 174–179, 1995.

[64] Sophia Kaplantzis, Alistair Shilton, Nallasamy Mani, and Y Ahmet Sekercioglu. Detecting selective forwarding attacks in wireless sensor networks using support vector machines. In *Intelligent Sensors, Sensor Networks and Information, 2007. ISSNIP 2007. 3rd International Conference on*, pages 335–340. IEEE, 2007.

[65] Samuel Kaski, Jari Kangas, and Teuvo Kohonen. Bibliography of self-organizing map (som) papers: 1981–1997. *Neural computing surveys*, 1(3&4):1–176, 1998.

[66] Myong Kim and Man-Gon Park. Bayesian statistical modeling of system energy saving effectiveness for mac protocols of wireless sensor networks. *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pages 233–245, 2009.

[67] Woojin Kim, Jaemann Park, Jaehyun Yoo, H Jin Kim, and Chan Gook Park. Target localization using ensemble support vector regression in wireless sensor networks. *IEEE transactions on cybernetics*, 43(4):1189–1198, 2013.

[68] Raghavendra V Kulkarni and Ganesh K Venayagamoorthy. Neural network based secure media access control protocol for wireless sensor networks. In *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, pages 1680–1687. IEEE, 2009.

[69] Sanjeev R Kulkarni, Gábor Lugosi, and Santosh S. Venkatesh. Learning pattern classification-a survey. *IEEE Transactions on Information Theory*, 44(6):2178–2206, 1998.

[70] Longin Jan Latecki, Aleksandar Lazarevic, and Dragoljub Pokrajac. Outlier detection with kernel density functions. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 61–75. Springer, 2007.

[71] Bill CP Lau, Eden WM Ma, and Tommy WS Chow. Probabilistic fault detector for wireless sensor network. *Expert Systems with Applications*, 41(8):3703–3711, 2014.

[72] Myeong-Hyeon Lee and Yoon-Hwa Choi. Fault detection of wireless sensor networks. *Computer Communications*, 31(14):3469–3475, 2008.

[73] Thomas W Lee and Terence R Mitchell. An alternative approach: The unfolding model of voluntary employee turnover. *Academy of Management Review*, 19(1):51–89, 1994.

[74] Richard Lippmann. An introduction to computing with neural nets. *IEEE Assp magazine*, 4(2):4–22, 1987.

[75] Mohammad Ahamdi Livani and Mahdi Abadi. Distributed pca-based anomaly detection in wireless sensor networks. In *in Proc. IEEE International Conference for IEEE Internet Technology and Secured Transactions*, pages 1–8. IEEE, 2010.

[76] Ching-Hu Lu and Li-Chen Fu. Robust location-aware activity recognition using wireless sensor network in an attentive home. *IEEE Transactions on Automation Science and Engineering*, 6(4):598–609, 2009.

[77] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[78] Yuyi Mao, Jun Zhang, and Khaled B Letaief. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE Journal on Selected Areas in Communications*, 34(12):3590–3605, 2016.

[79] Anthony Marcus, Ionut Cardei, Borko Furht, Osman Salem, and Ahmed Mehaoua. A mobile device prototype application for the detection and prediction of node faults in wireless sensor networks. *arXiv preprint arXiv:1401.5306*, 2014.

[80] Xin Miao, Kebin Liu, Yuan He, Dimitris Papadias, Qiang Ma, and Yunhao Liu. Agnostic diagnosis: Discovering silent failures in wireless sensor networks. *Wireless Communications, IEEE Transactions on*, 12(12):6067–6075, 2013.

[81] Tom M Mitchell et al. Machine learning. wcb, 1997.

[82] Mark R Morelande, Bill Moran, and Marcus Brazil. Bayesian node localisation in wireless sensor networks. In *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pages 2545–2548. IEEE, 2008.

[83] Habib Mostafaei, Antonio Montieri, Valerio Persico, and Antonio Pescapé. A sleep scheduling approach based on learning automata for wsn partial coverage. *Journal of Network and Computer Applications*, 80:67–78, 2017.

[84] Azzam Moustapha, Rastko R Selmic, et al. Wireless sensor network modeling using modified recurrent neural networks: Application to fault detection. *Instrumentation and Measurement, IEEE Transactions on*, 57(5):981–988, 2008.

[85] Azzam I Moustapha and Rastko R Selmic. Wireless sensor network modeling using modified recurrent neural networks: Application to fault detection. *IEEE Transactions on Instrumentation and Measurement*, 57(5):981–988, 2008.

[86] Gerhard Münz, Sa Li, and Georg Carle. Traffic anomaly detection using k-means clustering. In *GI/ITG Workshop MMBnet*, 2007.

[87] Luís ML Oliveira and Joel JPC Rodrigues. Wireless sensor networks: A survey on environmental monitoring. *JCM*, 6(2):143–151, 2011.

[88] Themistoklis Palpanas, Dimitris Papadopoulos, Vana Kalogeraki, and Dimitrios Gunopulos. Distributed deviation detection in sensor networks. *ACM SIGMOD Record*, 32(4):77–82, 2003.

[89] Themistoklis Palpanas, Dimitris Papadopoulos, Vana Kalogeraki, and Dimitrios Gunopulos. Distributed deviation detection in sensor networks. *ACM SIGMOD Record*, 32(4):77–82, 2003.

[90] Clifton Phua, Vincent Lee, Kate Smith, and Ross Gayler. A comprehensive survey of data mining-based fraud detection research. *arXiv preprint arXiv:1009.6119*, 2010.

[91] Diethelm Ostry Prasant Misra, Salil Kanhere. Safety assurance and rescue communication systems in high-stress environments: A mining case study. *IEEE Communications Magazine*, 48(4), 2010.

[92] Kyoritsu Publishers. *Anomaly Detection with Data Mining, Kyoritsu Publishers*. Elsevier, 2009.

[93] Shimpei Yamada Koutaro Yamamura Makoto Iwata Masayoshi Kai Qian Zhao, Yukikazu Nakamoto. Sensor scheduling algorithms for extending battery life in a sensor node. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E96-A(6):1236–1244, 2013.

[94] SP Rahayu, SW Purnami, and A Embong. Applying kernel logistic regression in data mining to classify credit risk. In *Information Technology, 2008. ITSim 2008. International Symposium on*, volume 2, pages 1–6. IEEE, 2008.

[95] Sutharshan Rajasegarar, Christopher Leckie, Marimuthu Palaniswami, and James C Bezdek. Distributed anomaly detection in wireless sensor networks. In *in Proc. IEEE Singapore International Conference on Communication Systems*, pages 1–5. IEEE, 2006.

[96] Sutharshan Rajasegarar, Christopher Leckie, Marimuthu Palaniswami, and James C Bezdek. Quarter sphere based distributed anomaly detection in wireless sensor networks. In *Communications, 2007. ICC'07. IEEE International Conference on*, pages 3864–3869. IEEE, 2007.

[97] Sutharshan Rajasegarar, Christopher Leckie, Marimuthu Palaniswami, and James C Bezdek. Quarter sphere based distributed anomaly detection in wireless sensor networks. In *in Proc. IEEE International Conference on Communications*, volume 7, pages 3864–3869, 2007.

[98] Carl Edward Rasmussen and Christopher KI Williams. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, 2006.

[99] Kay Romer and Friedemann Mattern. The design space of wireless sensor networks. *IEEE wireless communications*, 11(6):54–61, 2004.

[100] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

[101] Erich Schubert, Arthur Zimek, and Hans-Peter Kriegel. Generalized outlier detection with flexible kernel density estimates. In *Proceedings of the 2014 SIAM International Conference on Data Mining*, pages 542–550. SIAM, 2014.

[102] Nauman Shahid, Ijaz Haider Naqvi, and Saad Bin Qaisar. Characteristics and classification of outlier detection techniques for wireless sensor networks in harsh environments: a survey. *Artificial Intelligence Review*, 43(2):193–228, 2015.

[103] Ali Shareef, Yifeng Zhu, and Mohamad Musavi. Localization using neural networks in wireless sensor networks. In *Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*, page 4. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[104] Ali Shareef, Yifeng Zhu, and Mohamad Musavi. Localization using neural networks in wireless sensor networks. In *Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*, page 4. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[105] Belur V Sheela and Belur V Dasarathy. Opal: A new algorithm for optimal partitioning and learning in non parametric unsupervised environments. *International Journal of Parallel Programming*, 8(3):239–253, 1979.

[106] Yu-Ju Shen and Ming-Shi Wang. Broadcast scheduling in wireless sensor networks using fuzzy hopfield neural network. *Expert systems with applications*, 34(2):900–907, 2008.

[107] Bo Sheng, Qun Li, Weizhen Mao, and Wen Jin. Outlier detection in sensor networks. In *Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing*, pages 219–228. ACM, 2007.

[108] A Snow, P Rastogi, and G Weckman. Assessing dependability of wireless networks using neural networks. In *Military Communications Conference, 2005. MILCOM 2005. IEEE*, pages 2809–2815. IEEE, 2005.

[109] Ingo Steinwart and Andreas Christmann. *Support vector machines*. Springer Science & Business Media, 2008.

[110] Sharmila Subramaniam, Themis Palpanas, Dimitris Papadopoulos, Vana Kalogeraki, and Dimitrios Gunopulos. Online outlier detection in sensor data using non-parametric models. In *in Proc. International Conference on Very Large Data Bases*, pages 187–198. VLDB Endowment, 2006.

[111] Bo Sun, Lawrence Osborne, Yang Xiao, and Sghaier Guizani. Intrusion detection techniques in mobile ad hoc and wireless sensor networks. *IEEE Wireless Communications*, 14(5), 2007.

[112] Ruoying Sun, Shoji Tatsumi, and Gang Zhao. Q-map: A novel multicast routing method in wireless ad hoc networks with multiagent reinforcement learning. In *TENCON'02. Proceedings. 2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering*, volume 1, pages 667–670. IEEE, 2002.

[113] Duc A Tran and Thinh Nguyen. Localization in wireless sensor networks based on support vector machines. *IEEE Transactions on Parallel and Distributed Systems*, 19(7):981–994, 2008.

[114] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[115] Julian Winter, Yingqi Xu, and W-C Lee. Energy efficient processing of k nearest neighbor queries in location-aware sensor networks. In *Mobile and Ubiquitous Systems: Networking and Services, 2005. MobiQuitous 2005. The Second Annual International Conference on*, pages 281–292. IEEE, 2005.

[116] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[117] Che-I Wu, Hsu-Yang Kung, Chi-Hua Chen, and Li-Chia Kuo. An intelligent slope disaster prediction and monitoring system based on wsn and anp. *Expert Systems with Applications*, 41(10):4554–4562, 2014.

[118] Weili Wu, Xiuzhen Cheng, Min Ding, Kai Xing, Fang Liu, and Ping Deng. Localized outlying and boundary data detection in sensor networks. *IEEE transactions on knowledge and data engineering*, 19(8):1145–1157, 2007.

[119] Miao Xie, Song Han, Biming Tian, and Sazia Parvin. Anomaly detection in wireless sensor networks: A survey. *Journal of Network and Computer Applications*, 34(4):1302–1325, 2011.

[120] Bin Yang, Jianhong Yang, Jinwu Xu, Debin Yang, et al. Area localization algorithm for mobile nodes in wireless sensor networks based on support vector machines. *LECTURE NOTES IN COMPUTER SCIENCE*, 4864:561, 2007.

[121] Zhang Yang, Nirvana Meratnia, and Paul Havinga. An online outlier detection technique for wireless sensor networks using unsupervised quarter-sphere support vector machine. In *in Proc. IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, pages 151–156. IEEE, 2008.

[122] Zhang Yang, Nirvana Meratnia, and Paul Havinga. An online outlier detection technique for wireless sensor networks using unsupervised quarter-sphere support vector machine. In *Intelligent Sensors, Sensor Networks and Information Processing, 2008. ISSNIP 2008. International Conference on*, pages 151–156. IEEE, 2008.

[123] Feng Yin, Abdelhak M Zoubir, Carsten Fritsche, and Fredrik Gustafsson. Robust cooperative sensor network localization via the em criterion in los/nlos environments. In *Signal Processing Advances in Wireless Communications (SPAWC), 2013 IEEE 14th Workshop on*, pages 505–509. IEEE, 2013.

[124] Liyang Yu, Neng Wang, and Xiaoqiao Meng. Real-time forest fire detection with wireless sensor networks. In *Wireless Communications, Networking and Mobile Computing, 2005. Proceedings. 2005 International Conference on*, volume 2, pages 1214–1217. IEEE, 2005.

[125] Hao Yuan, Xiaoxia Zhao, and Liyang Yu. A distributed bayesian algorithm for data fault detection in wireless sensor networks. In *Information Networking (ICOIN), 2015 International Conference on*, pages 63–68. IEEE, 2015.

[126] Ke Zhang, Yuming Mao, Supeng Leng, Quanxin Zhao, Longjiang Li, Xin Peng, Li Pan, Sabita Maharjan, and Yan Zhang. Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks. *IEEE Access*, 4:5896–5907, 2016.

[127] Yang Zhang, Nirvana Meratnia, and Paul Havinga. Outlier detection techniques for wireless sensor networks: A survey. *IEEE Communications Surveys & Tutorials*, 12(2):159–170, 2010.

[128] Yang Zhang, Nirvana Meratnia, and Paul JM Havinga. Distributed online outlier detection in wireless sensor networks using ellipsoidal support vector machine. *Ad hoc networks*, 11(3):1062–1074, 2013.

[129] Qian Zhao and Yukikazu Nakamoto. Algorithms for reducing communication energy and avoiding energy holes to extend lifetime of wsns. *IEICE Transactions on Information and Systems*, E97-D(12):2995–3006, 2014.

[130] Qian Zhao and Yukikazu Nakamoto. Topology management for reducing energy consumption and tolerating failures in wireless sensor networks. *International Journal of Networking and Computing*, 6(1):107–123, 2016.