Online Failure Prevention from Connected Heating Systems

Manuel Mourato[1], João Mendes-Moreira[2], and Tânia Correia[3]

[1] Department of Electrical and Computer Engineering, Faculty of Engineering, University of Porto, Rua Dr. Roberto Frias, s/n, 4200-465 Porto, Portugal,
ee10087@fe.up.pt
[2] LIAAD-INESC TEC, Faculty of Engineering, University of Porto, Rua Dr. Roberto Frias, s/n, 4200-465 Porto, Portugal,
jmoreira@fe.up.pt
[3] Bosch Thermotechnology, S.A, EN 16 - Km 3.7, 3800-533 Cacia Aveiro, Portugal,
tania.correia@pt.bosch.com

Abstract. Many water boiler manufacturers are not able to detect the occurrence of failures in the machines they produce before they can pose inconvenience and sometimes danger for costumers and workers. Moreover, the number of boilers that have to be monitored, are many times in the range of the thousands or even millions, proportionaly to the number of costumers a company possesses. The detection of these failures in real time, would provide a significant improvement to the perception that consumers have of a certain company, since, if these failures occur, maintenance services can be deployed almost as soon as a failure happens. In this paper, an application prototype capable of monitoring and preventing failures in domestic water boilers, on the fly, is presented. This application evaluates measurements which are performed by sensors within the boilers, and identifies the ones that greatly differ from those received previously, as new data arrives, detecting tendencies which might illustrate the occurrence of a failure. The incremental local outlier factor is used with an approach based on the interquatile range measure to detect the outlier factors that should be analysed.

Keywords: Incremental Local Outlier Factor, Resilient Distributed Dataset, Outlier Detection

1 Introduction and Background

One of the major problems faced by manufacturers of domestic equipment and components of any kind, is the lack of ways to detect the occurrence of failures in these machines before they can pose an inconvenience and sometimes danger for costumers and workers. Specifically, the detection of these instances in real time, would provide a significant improvement to the perception that a consumer has of a certain company, since, if these failures occur, maintenance services can be deployed almost as soon as a failure happens. Ideally, by using predictive techniques, the detection of these issues before they happen by analyzing the

metrics that define the behavior of a certain component, becomes a desirable goal. Another issue faced by many companies, especially the ones that provide products for a household, is the fact that the number of machines that have to be monitored, are many times in the range of the thousands or even millions. This poses a problem, since the amount of data that has to be analyzed can be in the order of several Gigabyte, and needs to be processed in mere seconds.

In the specific case of domestic and industrial water boilers, as well as other heating equipment, the nature of these failures has to do with disruptions in the operation of these machines, in both a domestic and industrial context. In order to overcome this monitoring problem, sensors can be installed within water boilers, in order to record data related to several parameters and how they change over time. The main objective for this paper, therefore, was the development of an analytical backend application, capable of processing streams of data from measurements from boiler sensors, in real time, detecting outliers within them, which signify the occurrence of failures in a water boiler in different contexts. After this detection has been achieved, an analysis on the components responsible for that failure has to be done, ideally also in real time, so that this information can be passed on to maintenance services, so they act to solve these issues. It should also be able to deal with significant amounts of data in a short amount of time from different equipment/boilers.

After this introduction, a list of different implemented approaches related to outlier detection in boilers is mentioned in section 2. Section 3 presents the proposed application for outlier detection in data streams by analyzing its individual constituent parts, as well as an overview of an offline phase to assert the relevance of the different parameters. Finally, the results for this application performance are analyzed in section 4, first regarding the used methodology for outlier detection in data streams, then the overall advantages of using the proposed module for data processing, and finally the results of the offline phase and the knowledge added to the application derived from it. In section 5, conclusions taken from these results are presented, as well as suggestions for future work.

2 Related work

A considerable amount of work has been done on the topic of fault detection in several types of boilers. In [1–3], different cases of boiler faults and data mining techniques for their detection are presented. The biggest limitation for these cases however, is that none of these deal with outlier detection in data streams, or data sequences that arrive continuously and have to be analyzed in a real time manner. All of these approaches take time to assemble data, analyze it in order to create training datasets and only then construct a model capable of classifying or clustering boiler measurements. Therefore, even if the data has a temporal component, it is not analyzed in an on the fly manner. The work presented in [4] shows an interesting and relevant difference. It deals with the detection of fouling problems that lead to boiler pluggage, which in turn causes unscheduled shutdowns, for which a temporal data analysis is necessary. This

is a situation where the order of the data is relevant for the detection of the failure. However, once again, this classification is done in a static context after retrieving the necessary measurements.

Due to this lack of cases for real time outlier detection in boilers, a more general approach was taken, analyzing outlier detection in data streams for data from different contexts, as presented in [5, 6]. From the analysis of the available approaches that were implemented for data stream analysis, it can be seen that there is a much greater variety of supervised learning approaches. For this particular case, the domain of the boiler measurements, unsupervised technique is required since no labeled data is available. Such cases are presented in [7, 8], the last of which will be later discussed.

3 Proposed Methodology

In this section, an overview of the entire application is provided. Figure 1 ilustrates the different phases within the application.

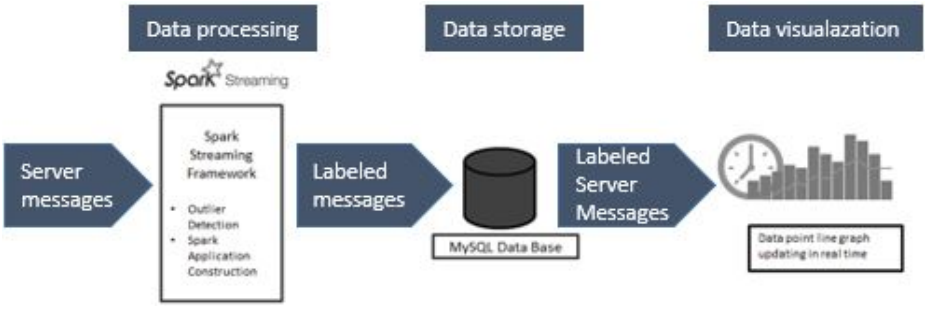

Fig. 1. High level application architecture

3.1 Data Stream Processing

The dataset used to test this application consisted on a collection of measurements received by the sensors within a domestic water boiler, collected over a period of 24 hours, and in which the parameters were controlled at will, in order to simulate different types of real world situations, namely equipment faults. Each observation was provided in JavaScript Object Notation (JSON) format, and it contained four kinds of attributes:

– Numerical: these are continuous, quantitative values supplied by the water boiler sensors, which indicate physical measurements, for example the water temperature within the boiler at a specific point in time;

- Categorical: these on the other hand, are attributes that have qualitative values, and indicate the state of water boiler components, work modes that the boiler might have (for example if at a certain point in time, the boiler is being used to heat water from a shower, or if it is being used for central heating) and also general descriptive information about the characteristics of the boiler, operation time, as well as many other parameters;
- Timestamp: a especially important attribute since messages did not arrive at fixed time intervals;
- Boiler ID: an identifier of the water boiler the present observation belongs to. For this project only one water boiler was available, so the ID was unique.

Another interesting characteristic of these data points is that each JSON message is only sent if some measurement value changes showing only the changed attributes. If, for example, the flame which warms up the water within the boiler was turned off at time t_0, and then turned on at time t_{10}, in the messages from $[t_1, t_9]$, the attribute relative to the state of this flame would not show up in the messages.

Because the domain of these boiler measurements is only known by a small group of people, and the water boiler measurements arrive in a data stream like fashion, it is not possible to present a throughout list of all specific attributes within these categories. Since each message only shows measurements that change, or that are different from previous ones, there is always a chance that a new attribute might show up.

The continuous reception of JSON messages was simulated with a server that send JSON messages at specified time intervals via a local TCP/IP connection, to a single port. That is, a socket connection was created in the local machine, in order to send the data. It is important to notice that, in order to preserve simplicity, JSON messages were not sent according to their timestamps, as that would require additional calculations for the exact times that were not particularly relevant for this project. The time interval for these messages could be constant and controlled by the user, or randomized, depending on what tests had to be performed.

The algorithm chosen to perform the failure detection procedure in the described boiler data was the Incremental Local Outlier Factor (LOF) [8]. Its main idea is to assign to each data record a degree of being an outlier, by computing the local density for each data point. This degree is called the local outlier factor (LOF) of a data record. Data records (points) with a high LOF have local densities smaller than their neighborhood and typically represent stronger outliers, unlike data points belonging to uniform clusters that usually tend to have lower LOF values.

The proposed incremental LOF algorithm computes LOF values for each data record inserted in a sliding window with a fixed size and constantly updated values, and instantly determines whether the inserted data record is an outlier. Specifically, it specifies two different situations:

(i) If $LOF(q) < 1$, then q is an inliner

(ii) If $LOF(q) \gg 1$, then q is an outlier

As the reader may deduce, the cases when LOF values are equal or just slightly bigger than 1 cannot be accurately labeled by relying solely on the incremental LOF, since it only provides a clear binary distinction for the cases above. The solution for this problem is addressed later on.

Despite this algorithm provides a simple yet effective way to detect outliers from dynamic data, it does present some flaws which need to be addressed. The first challenge is faced when using this algorithm with the provided data. Since each JSON message has both numeric and categorical attributes, the Euclidean Distance alone does not work because it does not deal with categorical attributes. Instead, the Overlap Distance Measure [14] is able to deal with categorical attributes. It is defined by the following equation:

$$\text{Overlap distance} = \begin{cases} 1 \text{ if } x_k = y_k \\ 0 \text{ otherwise} \end{cases} \tag{1}$$

Since in a distance measure as different the values are as larger the distance is, the distance measure had to be changed for this particular application: if the values are the same, 0 is given as a distance measurement, otherwise 1 is given. For a measurement with both numeric and categorical attributes, the Euclidean and the Overlap distances, respectively, are computed and added together [14].

Another major limitation for the incremental LOF algorithm is the fact that outliers cannot usually be well evaluated if LOF values are very close to 1, because there is a certain degree of subjectiveness depending on the window values that are presented. For example, a 1.2 value might be an outlier in a certain window, but it might not be one in a different window. In order to address this limitation, the notions of boxplot and interquartile range were used, as defined in [10].

The interquartile range (IQR), is the difference between the values of the third (Q_3) and first (Q_1) quartiles and it can be used to detect outliers [10]. This is done by multiplying this value by one and a half, and considering every data point from the data set that is either bigger than $Q_3 + 1.5 \times IQR$ or smaller than $Q_1 - 1.5 \times IQR$. Since in the incremental LOF algorithm outliers only exist for high values, the unique condition that must be verified is

$$Q_3 + 1.5 \times IQR \tag{2}$$

Armed with these conclusions, labeling data points based on their LOF values becomes trivial:

1. The LOF values within a given window W are sorted, and the median (Q_2) is determined;
2. Q_1 and Q_3 are determined as the 25% percentile and 75% percentile values in the window, respectively;
3. The IQR can then be calculated by $Q_3 - Q_1$;
4. Data points are then labeled as outliers, if their LOF values are bigger then $Q_3 + 1.5 \times IQR$

In order to apply this algorithm in a distributed scenario, in which measurements from a large number of boilers need to be processed at the same time, a suitable data processing engine has to be used. Spark is a fast and general-purpose cluster computing system for large-scale data processing, and was chosen to process the measurements from water boilers. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs [11]. The main abstraction which allows for distributed computing of large datasets is a structure called Resilient Distributed Dataset (RDD). By itself, Spark can only process static data and thus a specialized module to deal with data streams had to be used for this particular case. Spark Streaming is a micro-batching streaming module, which differs from traditional continuous processing of streams by its use of Discretized Streams. In simple terms, one can define a D-Stream as a collection of RDDs, in which each RDD possesses the data received by the application during a fixed batch time interval [12].

3.2 Data Storage and visualization

Once the processed data points are properly labeled, in order to make them more interpretable for the user, they were stored in a MySQL Database, so that a visual representation of all data points can be provided in a near real time fashion (Figure 2). This way, outliers are more intuitive and observable for front-end users. Using the Node.js library provided by JavaScript server side applications, a server was created, with the intent of extracting the data points from the referred MySQL database. The extracted messages and respective labels are then requested to the Node.js server by a client, in order to graphically represent the data in our browser, at a specified local host port. The library used to achieve a real time graphical representation was D3.js.

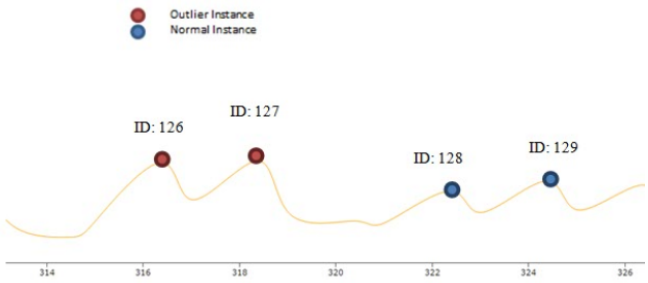

Fig. 2. Real time Visualization Example

3.3 Offline Learning

Because the knowledge of the domain was scarce, there was no information about which specific attributes were responsible for the results of the incremental LOF algorithm.

This offline learning phase uses the knowledge gained from the outlier detection phase, in order to determine which attributes are more significant in the labeling of the data points.

The Classification And Regression Tree (CART) algorithm [13] was chosen to determine which attributes were more significant for outlier labeling. However, one of the issues in training a decision tree classification model is the fact that it can be a lengthy process depending on the number of variables that need to be considered. This problem was mostly solved by using the Apache Spark MLlib, a library which uses the Spark engine to perform computations and a set of machine learning algorithms in parallel, thus reducing the model computation time to a few seconds.

4 Analysis of incremental LOF values

As shown in Table 1, the values of the incremental LOF algorithm can change greatly depending on the parameters. The left and central cases analyze the same 10 LOF values, for different parameters, and the last case compares the first 10 data points in 3 consecutive windows. Looking at both the left and central cases, it becomes apparent that it tends to be a stabilization of the LOF values as the parameter values increase. In the case of N, the larger the window size, the more faithfully statistics can be derived for the whole data stream, and thus differences between data points become less significant, unless there is a great discrepancy between the measured attributes. When the k nearest neighbor is increased, the resulting LOFs do not follow a clear tendency.

Table 1. Columns 1-3 present different LOF values for different K values and N=500; Columns 4-6 present different LOF values for different N values and K=10; columns 7-9 present LOF values for consecutive time intervals, N=500 and K=10.

K; N=500			K=10; N			N=500; K=10		
K=5	K=8	K=11	N=100	N=500	N=1000	T0	T1	T2
1.440	1.313	1.293	1.132	1.287	1.395	1.287	1.168	0.891
1.244	1.037	1.077	1.059	1.055	0.965	1.055	1.006	0.907
1.103	0.902	0.916	0.962	0.886	0.769	0.886	0.969	0.907
1.041	0.854	0.887	0.961	0.860	0.763	0.860	0.969	0.902
1.046	0.855	0.887	0.950	0.860	0.747	0.860	0.966	0.972
1.150	0.920	0.887	0.950	0.923	0.787	0.923	1.016	0.962
1.259	1.071	0.943	0.931	0.993	0.809	0.993	1.011	0.903
1.228	1.010	0.938	0.915	0.975	0.819	0.975	0.973	0.877
1.102	0.891	0.948	0.929	0.909	0.855	0.909	0.955	0.877
1.049	0.858	0.929	1.004	0.881	0.829	0.881	0.956	0.915

4.1 Outlier Classification Results

As discussed in section 3, LOF values are classified by using the Interquartile Range concept for each sliding window of values. Figure 3 illustrates a series of boxplots for different windows created using the Minitab Sofware, one with only normal values, and another with both normal and outlier values. The window size N is of 500 and the nearest neighbor k value is of 10.

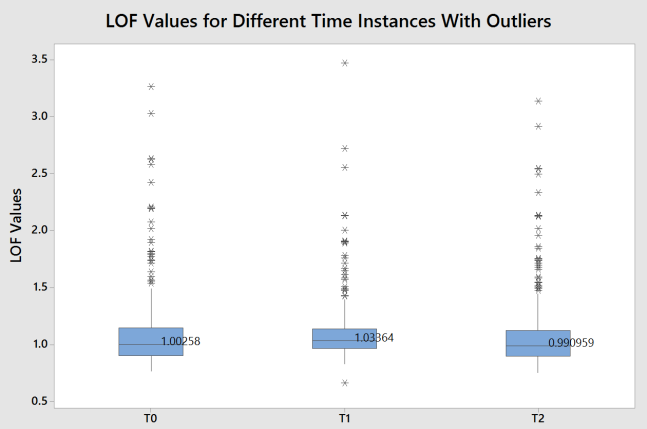

Fig. 3. Boxplot for LOF values with and without outliers.

Observing Figure 3 it is possible to notice the subjective nature of LOF values from time instance to time instance. Each boxplot contains the value of the median for each of the different windows, as well as the extreme values of the box, i.e., the values of the first and third quartiles. Table 2 presents the values of these three statistics, as well as their interquartile ranges for three different box plots.

Table 2. Statistics for different boxplots

First Quartile	Median	Third Quartile	Interquartile Range	Labeling Limit
0.904	1.003	1.147	0.243	**1.512**
0.967	1.034	1.139	0.172	**1.397**
0.898	0.991	1.126	0.228	**1.467**

By knowing these parameters, it becomes quite simple to establish a labeling value limit by using Equation 2. The LOF values at $T0$ are shown in Table 3.

Table 3. Sample of classification for 6 LOF values

Measurement Index	121	122	123	124	125	126
LOF Values	0.8451	0.8456	0.8448	0.8997	1.0726	1.6405
Outlier Label	0	0	0	0	0	1

4.2 Apache Spark Streaming Performance

Having presented the results from the used outlier detection algorithm in a data stream, it is now time to look at how the Spark Streaming module behaved, when it comes to implementing the incremental LOF algorithm, by analyzing its processing time for multiple cases.

Table 4. Sparks Performance for Different incremental LOF parameters

Case	Average Processing Time	Average Scheduling Delay
N=50;K=5	116 ms	0 ms
N=500;K=11	$>$ 800 ms	$> 1min$

By looking at these results, it is clear that data processing for algorithms of lower complexity works better in the Spark Streaming Module. In the case where N=50 and K=5, the average scheduling delay in the processing of boiler measurements, that is, the time each message needs to wait before it gets processed, is pretty much zero. This means that the chosen batch interval of 1 second was sufficient to handle the algorithm computations. In the case where N=500 and K=11 though, as shown in Table 4, it becomes evident that data processing is not viable, due to the fact that the processing time of each JSON message is much larger than the batch interval.

Specifically the processing time for N=500 and K=11 is exactly two times bigger than that of the defined batch interval. This of course causes an increasing scheduling delay because JSON messages arrive faster than the time that Spark Streaming takes to process them.

The reason why the data processing time takes this long for bigger values of the N and k parameters, is mainly because of the sequential nature of the incremental LOF algorithm. Since all the operations depend on themselves, one operation cannot be completed without concluding the previous one, which defeats the paradigm of data parallelization that Spark uses. Even though a lot of partitions might be formed, they cannot be executed in parallel effectively, because one specific operation can create a bottleneck, and because all operations have to be done in sequence, the processing time is completely dependent on that bottleneck.

As can be seen in Table 5, in the case in which the input is kept at 1 record per second, the number of partitions within each RDD is not that significant. Because of this fact, having more cores available for data processing is irrelevant, since there are not enough partitions to divide among the nodes, resulting in

Table 5. Sparks Performance for Different Input Volumes

Case	Average Processing Time	Average Scheduling Delay
1 Message	1 core: 107 ms	1 core: 0 ms
Per second	3 cores: 110 ms	3 cores: 0 ms
5000 Messages	1 core: 370 ms	1 core: 0 ms
Per second	3 cores: 283 ms	3 cores: 0 ms

a very similar performance. This situation changes when the input number is increased, namely, data processing becomes significantly faster when all cores from the local machine are used with a difference of about 90 ms when there are 5000 input messages.

It is important to notice that because Spark was deployed in local mode, the advantages of data parallelization are smaller than executing it in a distributed environment with multiple JVMs and nodes.

Another relevant aspect to take into account is that, because only measurements from a single water boiler were provided, the parallelization by source suggested in section 3 and the possible benefits it would have for the processing speed were impossible to test in this use case.

4.3 Decision Tree Classifier Analysis

As discussed in section 3, a key interest for this application, would be to determine what are the most significant attributes from each measurement, that lead to their classification as being outlier or normal. In order to accomplish this, a Classification And Regression Tree was built as shown in Figure 4 and Table 6.

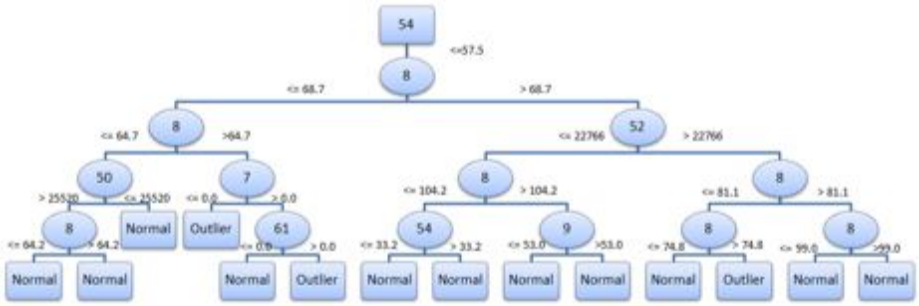

Fig. 4. Sample of the obtained Decision Tree Classifier for the boiler data

Some interesting and meaningful variables were deemed relevant, like the primary temperature in the water boiler. Ultimately, the results of this decision tree model can only be evaluated accurately if one possesses labeled measurements in order to test the model predictive capacity. Unfortunately, there were no labeled measurements available to do it.

Table 6. Relevant attributes for data classification

Attribute Index	Attribute Meaning
7	Boiler power in Watts
8	Primary water boiler temperature
9	Primary water boiler default temperature
50	Operation time for water heating
52	Hot water heating start number
54	Hot water outside temperature
61	Pump operation mode

5 Conclusions and Future Work

The goal of this paper was to develop a backend application capable of detecting outliers in data stream measurements from sensors of multiple water boilers.

Regarding the use of the incremental LOF algorithm, the overall result was fairly reasonable, considering that there was no labeled data available. The developed prototype is able to detect both single and collective outliers in a time interval of one second, if the window size and k neighbors is not too high. The subjectiveness of the original incremental LOF values was solved by continuously determining statistics based in boxplots for each window of values, for each new measurement that arrived.

Apache Spark and Spark Streaming provide an easy, compact syntax to perform map reduce operations. An analysis was conducted on the effects of the applications level of parallelism, which showed the potential for future work in a distributed computation environment. Because there were only measurements from one water boiler, parallelization by data source was not achieved, but nevertheless data was divided into partitions successfully, proving that it is possible to implement this in future projects.

The offline phase was also a very relevant step for understanding the domain of the measurements, and provide insights on which parameters from the sensor measurements were more relevant for outlier classification.

The proposed algorithm was too complex to process measurements from water boilers, not being fast enough when the number of samples and nearest neighbors in the incremental LOF algorithm is large. This is due to the sequential nature of incremental LOF. To adapt it to parallel processing is a promising research line.

Another challenge is to use labeled test datasets to evaluate the performance of this work. Approaches for outlier detection using supervised learning should also be tested.

Regarding the Apache Spark engine and the Spark Streaming module, since all tests were performed in local mode, the data processing speed was reasonable. However, Spark could be deployed in a distributed computation scenario with multiple nodes (JVMs) and even more cores.

The offline phase could be done online using the Very Fast Decision Tree algorithm instead of the CART algorithm that was used in this work.

Acknowledgements This work is partially financed by the ERDF European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme within project POCI-01-0145-FEDER-006961, and by National Funds through the FCT Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) as part of project UID/EEA/50014/2013.

References

1. Karim Salahshoor, Mojtaba Kordestani, Majid S. Khoshro: Fault detection and diagnosis of an industrial steam turbine using fusion of SVM (support vector machine) and ANFIS (adaptive neuro-fuzzy inference system) classifiers. Energy, 35(12): 5472-5482 (2010).
2. Zhe Song, Andrew Kusiak: Constraint-Based Control of Boiler Efficiency: A Data-Mining Approach. IEEE Transactions on Industrial Informatics, 3(1): 73-83, (2007).
3. Jeffrey Schein, Steven T. Bushby: A hierarchical rule-based fault detection and diagnostic method for HVAC systems. HVAC&R Research, 12(1): 111125 (2006).
4. Andrew Kusiak, Alex Burns Mining temporal data: A coal-fired boiler case study. International Conference on Knowledge-Based and Intelligent Information and Engineering Systems, 953958 (2005).
5. David J. Hill, Barbara S. Minsker, Eyal Amir: Real-time Bayesian anomaly detection in streaming environmental data. Water Resources Research, 45 (2009).
6. Cao Lijun, Liu Xiyin, Zhou Tiejun, Zhang Zhongping, Liu Aiyong: A Data Stream Outlier Detection Algorithm Based on Reverse K Nearest Neighbors. International Symposium on Computational Intelligence and Design 236-239 (2010).
7. Fabrizio Angiulli, Fabio Fassetti: Detecting distance-based outliers in streams of data. Proceedings of the sixteenth ACM conference on Conference on information and knowledge management, pp. 811820 (2007).
8. Dragoljub Pokrajac, Aleksandar Lazarevic and Longin Jan Latecki: Incremental local outlier detection for data streams. IEEE Symposium on Computational Intelligence and Data Mining, 2007 504515 (2007).
9. David F. Williamson, Robert A. Parker and Juliette S. Kendrick: The box plot: a simple visual method to interpret data. Annals of internal medicine, 110:916921 (1989).
10. Navidi, William Cyrus: Statistics for engineers and scientists, vol.1, McGraw-Hill New York (2006).
11. Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica: Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (2012).
12. Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker and Ion Stoica: Discretized streams: fault-tolerant streaming computation at scale. Proceedings of the SOSP'13, 423-438 (2013).
13. Leo Breiman, Jerome H. Friedman, Richard A. Olshen, Charles J. Stone: Classification and regression trees. Chapman and Hall/CRC (1984).
14. D. Randall Wilson and Tony R. Martinez: Improved heterogeneous distance functions. Journal of artificial intelligence research, 6: 1-34 (1997).