








# Machine Learning Techniques for Detecting Supercomputer Applications with Abnormal Behavior

Alexander Bezrukov<sup>1</sup> , Mikhail Kokarev<sup>2</sup> , Denis Shaykhislamov<sup>2</sup> ,  
Vadim Voevodin<sup>2</sup> , and Sergey Zhumatiy<sup>2</sup> 

<sup>1</sup> Plekhanov Russian University of Economics, Moscow, Russia  
Bezrukov.AV@rea.ru

<sup>2</sup> Research Computing Center, Lomonosov Moscow State University, Moscow, Russia  
mikhail.kokareff@gmail.com, sdenis1995@gmail.com,  
{vadim,serg}@parallel.ru

**Abstract.** There are different approaches that help to solve the issue of low efficiency of modern supercomputer usage. One of them is based on constant monitoring of a supercomputer job flow in order to promptly detect inefficient programs. The execution dynamics of such programs usually differs from the “normal” behavior of common programs; however, it is very difficult to establish exact criteria for determining abnormal behavior. Machine learning methods are therefore used in this study for detecting abnormal jobs. This paper deals with an important aspect of working with machine learning methods, namely data preparation. The solution proposed herein was evaluated on the Lomonosov-2 supercomputer.

The issue of optimal input data selection is one of the key steps for transferring the methods suggested in the paper to other supercomputers. The analysis described in the article has served as a starting point for developing a methodology for applying overall solutions to other supercomputers, which is also described in this paper.

**Keywords:** Supercomputer · High-performance computing  
Task flow · Anomaly detection · Program efficiency · Machine learning

## 1 Introduction

High-performance computing is becoming more and more large-scale: the number of scientists from different research areas that use supercomputer technologies for solving scientific problems is constantly growing. This is definitely a positive trend which hopefully will continue in the future. But this has an unobvious

---

This work was partially funded by the Russian Foundation for Basic Research (grants 16-07-00972 and 17-07-00719) and a study grant from the Russian Federation President’s Fund (SP 1981.2016.5).

drawback. A lot of new scientists entering this area are usually skillful specialists in their research areas, such as computational physics, molecular dynamics, weather forecasting, drug design, etc., but they are by no means experienced enough in parallel computing. Taking also into consideration that the architecture of modern supercomputers is highly complex, it becomes really difficult to develop efficient parallel applications that consider all the peculiarities of underlying hardware. And this results in a substantial amount of parallel programs with really low execution efficiency [1].

One can say that more people involved in parallel computing means efficient off-the-shelf application packages being developed, and this is partially true. New ready-to-use packages appear as well as existing packages enhance their functionality, although unfortunately they are often neither very scalable nor efficient in practice. But such packages play significant role in forming the overall efficiency of using supercomputer centers, so their behavior should be monitored and analyzed, which is the goal of another research being conducted at the Research Computing Center of the Lomonosov Moscow State University (RCC MSU) [2].

There are many different approaches as to how the efficiency of a particular parallel program can be analyzed and optimized. Various profilers, trace analyzers, and debuggers have been developed and successfully used to address this task. But before an application can be studied thoroughly, we would have to become aware that this application has low execution efficiency and that it needs to be analyzed. And it turns out that in many cases not only users but also system administrators do not know that an application has some performance issues. This means that a constant monitoring of all programs running on a supercomputer is needed in order to find inefficient applications with possible performance issues.

This work is aimed at solving this particular task. The main goal is to detect abnormal applications, i.e. applications with abnormal behavior which significantly differentiates from the standard behavior of the tasks in a supercomputer job flow. The behavior of applications is described using system monitoring data. Owing to the fact that it is currently almost impossible to precisely establish criteria for abnormal behavior, machine learning (ML) methods are used for that purpose. But tuning machine learning techniques to maximize its performance can be quite tricky. One of the main difficulties that are encountered on this path is to correctly select and prepare needed input data, which can drastically influence the overall accuracy of machine learning methods.

The main contribution of this paper is a description of methods for determining a suitable input data set which can lead to improvements in classification accuracy. These methods were implemented and evaluated using an anomaly detection method developed previously. Furthermore, the conducted data preparation analysis served as an entry point for developing a methodology for applying overall anomaly detection approaches to other supercomputer centers. This newly developed methodology is also presented in this paper.

The paper is organized as follows. Section 2 briefly describes the work that was previously done within this research, as well as related studies. Section 3 is devoted to the problem of data preparation for the machine learning method used for anomaly detection. A methodology for applying the developed solution for anomaly detection to other supercomputers is described in detail in Sect. 4. Section 5 contains the conclusions made as well as plans for future research.

## 2 Background and Related Work

The main goal of this work is to find abnormally inefficient applications in a supercomputer job flow using system monitoring data. There are several related works with similar goals that could be mentioned. Many of them are based on just static thresholds that help to determine abnormal behavior: this is how system monitoring tools like Nagios or Zabbix do. But this works well just for simple cases and it is required to accurately adjust these thresholds.

For more complex cases, machine learning methods are used. For example, in [3], ML techniques are used for program classification as well. But the authors of that paper use supervised methods for identifying specific applications (e.g., software packages like GROMACS) based on performance data. For detecting inefficient behavior, simple static thresholds were used.

Another example, which is the most related one to our research, is the paper [4]. The authors present a method for detecting performance anomalies in HPC systems. A system monitoring performance data is also used in this case for anomaly detection, although they are interested in detecting performance variations caused by resource contention or hardware/software problems on a node. A number of node-level anomalies like “orphan processes” and “hidden hardware problems” are specified, and are then detected using machine learning methods. So the main goal of this work is quite different, even though the approach used is very similar. It is interesting that Random Forest algorithm showed the best classification results, as in our study. The high performance achieved in paper [4] showed us that the methods we were planning to use in our case should lead to suitable results.

There are other works where machine learning techniques for performance analysis in the HPC area are used (for example, [5, 6]), but none of them aims at solving our task. Nevertheless, it should be mentioned that studying these works helped us to determine the methods suitable in our case.

As mentioned earlier, the anomaly detection method proposed is based on analyzing data collected with monitoring systems. At the MSU Supercomputing Center, a set of proprietary tools is currently being used, but it is planned to switch in the near future to the DiMMon monitoring system [7], which is being developed at the RCC MSU for use on exascale-level supercomputers. Data from processor counters (e.g., CPU user load), memory and communication network intensity (e.g., number of L1 cache misses per second, amount of bytes sent per second): all this information is collected for each job running on the supercomputer, forming the basis needed for the performance analysis of job efficiency.

Using these dynamic characteristics for performance description, each job can be classified as normal, suspicious or abnormal. In general, a job is classified as normal if no performance issues are found. A job is called abnormal if it is definitely working incorrectly, wasting computing resources; this could happen if a program stalls or a software/hardware error has been encountered. A job is classified as suspicious if we can detect some performance issues in its behavior but we cannot be sure that this behavior is definitely incorrect (abnormal), so a more detailed analysis is needed.

The overall job classification process is organized as follows (a detailed description can be found in [8]).

Each job is represented by the values of dynamic characteristics changing during the program runtime, that is, by a number of timelines, one timeline per characteristic. For each job, these timelines are divided into time intervals. An intellectual method is used for this purpose that tries to identify substantial changes in the behavior of the program, separating different logical stages of the program execution. In this case, the behavior of each interval is quite simple and can be therefore represented accurately using integral values (e.g., max, min, median, oscillation rate). After timelines are divided into time intervals, each interval is individually classified as abnormal, suspicious or normal using integral values for the chosen dynamic characteristics.

For interval classification purposes, we use a method based on the Random Forest algorithm (Scikit-learn [9] implementation). This is a supervised method that was trained on a set of 520 manually classified intervals (270 normal, 70 abnormal and 180 suspicious intervals), leading to a classifier accuracy of  $\sim 0.93$  on the Lomonosov-2 supercomputer. The accuracy is calculated as the ratio of correctly classified intervals to the number of all intervals in the set.

When each interval of a job runtime is classified, it is needed to assign a class to the job in general. It is done using a set of criteria that attempt to determine whether a substantial amount of processor time was consumed by intervals with abnormal/suspicious behavior. The results were validated on a test set of 110 applications (32 abnormal, 48 suspicious, 33 normal). The overall job classification accuracy achieved was  $\sim 0.92$ .

The resulting performance is quite high. However, the following should be noted. One of the most important points that influence the performance of machine-learning-based classification is data preparation. And the original selection of the feature set was based only on our initial sense of what, in our opinion, should be most useful for the classification. It was thus decided to study how much accuracy can be increased with a more intelligent approach to the choice of the input data, based on a rich existing analytical experience in this area. Within this paper, we describe a number of different methods we have tried for choosing the most suitable feature set, aimed at increasing the accuracy of the classifier we have developed.

Moreover, the approach for data preparation described in this paper makes this overall classification process much more portable, since choosing the correct input data is one of the most challenging tasks for performing an accurate

classification. Following the methodology described in Sect. 4, one can try to implement the described classifier on a different supercomputer.

### 3 Data Preparation for the Anomaly Detection Approach

In a previous work, we selected and fixed an input feature set for the classifier according to our initial view of which data is the most important in our opinion (here and below, we will refer to this set as the “basic feature set”, and, accordingly, to the classifier based on it as the “basic classifier”). However, during the working process, we figured out that this can potentially be improved with useful information about job dynamic behavior that was not used at that time. So it was decided to rethink our data preparation process.

The data preparation stage for machine-learning-based algorithms consists of three steps: selection, preprocessing and transformation [10]. Usually a lot of different types of input data are available that can be used for classification purposes, but using all data not always results in the best performance and, also, it can be very computationally complex. So the **data selection** step is aimed at choosing the right subset of data types that will lead to the best accuracy and/or classification speed.

In our case, the machine learning algorithm uses system monitoring data as input, so we can potentially use all the information that can be collected from the system counters describing the utilization of CPUs, memory subsystem, communication network, etc. It should be noted that there are always hardware restrictions in a processor (which are not the same for different processor families), which allows us to collect only a small amount of processor counters simultaneously. For each supercomputer, we heuristically chose the most suitable data. For example, the set of data being collected on our Lomonosov-2 supercomputer is the following:

- *CPU utilization*: CPU user load, other types of CPU load (system, iowait), loadavg.
- *Memory usage intensity*: number of L1/L2/L3 cache misses per second; number of load/store operations per second.
- *Communication network usage intensity*: number of bytes/packets sent/received per second, separately for MPI and file system networks.
- *GPU utilization*: GPU user load, GPU memory load, GPU memory utilization.

Using all this data is not the best option, so we need to choose a suitable subset. This task turned out to be the most challenging in the data preparation stage; a description of methods used is further provided.

The second step in the data preparation stage is **data preprocessing**. This step includes such processes as data cleaning and formatting, which in our case is almost not needed at all: the monitoring system provides us with all the needed data in a suitable format.

The third step is **data transformation**. During this step, data scaling or decomposition as well as aggregation can be performed. In this work, a machine learning algorithm is used for time interval classification, and this is done with the Random Forest algorithm. It does not require data normalization, so no need for any data scaling in our case.

Data aggregation is done twice before passing the input feature set to the classifier. At first, 2-min approximation is used: raw data collected by the monitoring system is replaced every two minutes with integral values (max, min, average, etc.). This is necessary to reduce the amount of data that needs to be stored, and it was decided not to change this step in this work.

The next aggregation is done on our side, at the interval level. As mentioned earlier, we form time intervals in such a way that they show a simple behavior which can be accurately described using integral values. This means that each interval and each dynamic characteristic (like CPU user load) is described with only maximum, median, etc. values instead of using a time series of raw numbers (usually, there are 30 to 100 time points representing each interval). Furthermore, most jobs studied in our work are parallel, which means that we also need to aggregate the data across different processor cores in a node, as well as between nodes. So we use triple aggregation in this case: first by space (between cores in a node), then again by space (between nodes) and then by time (between time points).

The question is, what integral values should be used for interval description? We have made a list of possible variants that can be useful in practice, according to our experience. For each variant, three aggregation methods are provided (within a node/between nodes/time):

- *Minimum (within a node)/minimum (between nodes)/minimum (between time points)*. Maximum is not so interesting since it is often equal to the peak.
- *Average/average/median*. We have selected median for time aggregation due to its better resistance to outliers, which leads to a more accurate description in many cases.
- *Maximum/average/average*. This value helps to detect imbalance between nodes.
- *Minimum/(average-minimum)/average*. We have selected (average-minimum) for aggregation between nodes instead of just average, since using just average leads to values very close to the minimum, which was already described earlier.
- *Average/average/oscillation\_rate*. The oscillation rate is calculated as maximum-minimum range divided by the overall average. This number reflects the relative fluctuation of the max and min values of the characteristic around the average.
- *Maximum/maximum/oscillation\_rate*. This helps us to describe fluctuation of characteristics as well. We do not use minimum/minimum/oscillation\_rate because it is usually equal to zero.
- *(Maximum/maximum/maximum)/(average/average/average)*. Another way of describing fluctuation.

We also thought about adding more integral values, such as skewness or kurtosis [11], but it seems like it would not add any new information to the list of considered values, so it was decided not to expand the list, which was already quite big.

Taking into account the aforesaid, there are two options that we can adjust in order to try to improve classification accuracy, namely what data types should be selected and what integral values should be used. But we have 20 different data types, with 7 integral values to represent each of them, which makes  $2^{140}$  different possible feature subsets. This means that we need some intellectual method to choose one close to the optimal for our purpose.

But before starting to choose the appropriate set of features, it is necessary to understand whether it is worth doing in principle. We need to evaluate whether the possible accuracy gain is statistically significant. If no, there is no sense in performing any analysis at all; the current feature set would then be quite suitable.

For checking the statistical significance of the model accuracy ratio increase, we use Student's  $t$ -criterion (1):

$$T = max + 3 * (st\_error), \quad (1)$$

where  $max$  is the maximum accuracy obtained using the basic feature set, and  $st\_error$  is the standard error of the mean for the accuracy with the basic classifier. We use the  $t$ -test as it is commonly applied to check the significance of the difference between two values, in our case, the model accuracy values. If the accuracy obtained with new feature sets is higher than this value, we can say that we found a statistically significant better result.

We ran the basic classifier 1000 times and analyzed the cross-validation accuracy in the intervals of real-life applications from the Lomonosov-2 supercomputer. The max was 0.9398, while the average was 0.9324 (most of the accuracy variation is related to the random nature of the classifier due to the use of the Random Forest algorithm). The  $t$ -criterion in this experiment is then equal to 0.94. The analysis showed that this value can be exceeded using new feature sets. For example, the average accuracy for the most complete feature set (all 140 features considered) is 0.9434, which is higher than the  $t$ -criterion. It should be noted that this set does not suit us owing to the following reasons: (1) the result obtained is possibly not the highest one; (2) using so many characteristics is likely to lead to an overfitted model (and also to issues with classification speed), so this amount should be reduced.

This analysis proved that it is worthwhile to search for more optimal feature sets for our classifier. The standard approach for solving this issue is to use discriminant function analysis. The next section describes in detail how it was used in our research.

### 3.1 Discriminant Function Analysis

There are three possible approaches for choosing an appropriate feature set:

1. Use all possible features and manually select the most important ones (standard method).
2. Backward stepwise.
3. Forward stepwise.

All these approaches are usually quite similar in terms of the accuracy that can be achieved as a result. Both backward and forward stepwise methods were attempted in order to quantitatively evaluate the significance of the features and obtain the preliminary list of features that remain in the model after the completion of both algorithms, as well as to gain an a priori insight into the features' influences.

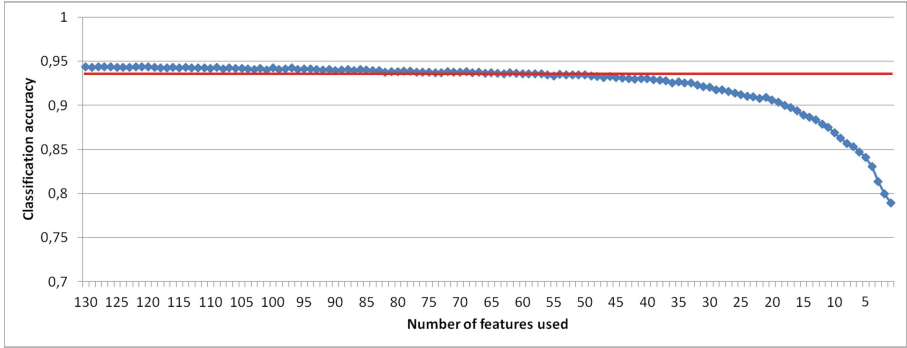
**Backward Stepwise Method.** The main idea of this algorithm is quite simple: we perform a number of steps, and at each step, the feature leading to the least accuracy loss is removed. Scikit-learn provides its own implementation of the backward stepwise algorithm [12] but it calculates the accuracy using only one cross-validation at a time, which in our case leads to rather unstable results. So we decided to implement five cross-validations and take the average result. The overall backward stepwise algorithm looks as follows:

- While the break condition is not satisfied:
  - Temporarily remove one feature from the set  $A$ . Calculate the classification accuracy using the resulting feature set  $A_1$ .
  - Repeat the previous step for each feature in the set  $A$ . As a result, we have  $N$  accuracies for all possible sets  $A_n$  (where  $N$  is the size of the feature set  $A$ ) without one feature.
  - Find the set  $A_i$  that shows the highest accuracy.
  - Use this set  $A_i$  and go to the next iteration.

There are two points that need to be clarified. The first one is how the accuracy is calculated. We use cross-validation: the training data is split into five equal parts; one part serves as the test set, the other four are chosen as the training set. There are five possible ways to do this, and the cross-validation accuracy is calculated based on this 5-fold splitting. Then we repeat cross-validation five times and take the average accuracy value. The second point to clarify is the break criterion. There are several standard ways to do this: do not stop until a particular number of features is reached or do not stop until the accuracy loss is less (or the overall accuracy is higher) than a specified threshold.

Figure 1 shows average accuracy results for the backward stepwise method. Along the X axis, we have the numbers of features left in the feature set; the Y axis corresponds to the accuracy value for the chosen feature set at each iteration of the backward stepwise algorithm.





**Fig. 1.** Interval classification accuracy variation during the backward stepwise method. The horizontal line corresponds to the basic classifier accuracy

It can be seen that the same accuracy as for the basic classification (marked with a red horizontal line) is achieved with 50 to 70 features, whereas the basic feature set consists of 33 features. Using 70+ features is not a suitable option for us because it does not provide a substantial accuracy gain, while decreasing the classification speed and increasing the probability of the overfitting problem. This means that the use of the backward stepwise method leads to worse results than those of the basic classification.

So it was decided to switch to the second method: the forward stepwise.

**Forward Stepwise Method.** The forward stepwise algorithm works the other way round compared with backward stepwise: we start with a small feature set and add one feature that maximizes the accuracy at each iteration. But there are also several points that should be determined in this case: what features to start with and when to stop.

One of the most important steps in the forward stepwise algorithm is the choice of a suitable starting feature set. Usually it includes 5 to 10 features. But they can be selected in a different way, for example, randomly or based on expert knowledge. At first, we tried to rely on our experience-based assumption that the median and the oscillation rate are the most important integral values that are useful for the classification process. So we tried to use only them as the starting feature set. The break criterion was the maximum size of the feature set: not more than 35 to 40 features. However, this led to poor accuracy results, meaning that this assumption is not good enough. Using randomly chosen starting sets is not a good option either, since we know that some combinations of features must be included, otherwise important behavior peculiarities could be omitted by the classifier.

As a result of trying different variants, the following method for choosing a starting feature set and finishing the forward stepwise process was formed. We combined data types into the following groups (several groups contain only one data type):

- CPU load;
- load average;
- memory reference intensity (number of load/store operations per second);
- number of L1 cache misses per second;
- number of L2 cache misses per second;
- number of L3 cache misses per second;
- MPI network usage intensity (number of bytes/packets sent/received per second);
- file system network usage intensity (number of bytes/packets sent/received per second);
- GPU utilization (GPU user load, GPU memory utilization).

Each group represents a part of information that must be included in the resulting feature set one way or another. Each group can be represented using any integral value specified earlier; there are no restrictions on that. It should be noted that three data types, namely CPU system load, CPU iowait load and GPU memory load, were considered not important enough, so they are not included in any group and may not therefore be included in the resulting feature set.

Initially, five random features were selected from the list above. We chose the following break criteria: (1) at least one feature from each group must be included into the feature set; (2) the size of the feature set must be not less than 30. The first criterion is needed to be sure that the resulting feature set includes all the information we think is necessary. The second one guarantees that the feature set will not be too small, which, in most cases, leads to poor accuracy results.

Owing to the random nature of the classification algorithm used, the feature set obtained can be quite different each time we run the forward stepwise method. So, in order to choose the most appropriate one, we need to collect enough statistics.

We ran the forward stepwise method 50 times and obtained 50 different feature sets. Next, we needed to determine which feature set shows the best performance results. So we performed cross-validation 2000 times for each feature set and calculated the average accuracy. After that, we took the top five feature sets having highest accuracy (their results were very similar) and then selected the best one among them, using our own knowledge on what features are more important. The chosen set turned out to be also the most uniform: the number of features in each group was almost equal.

*Final Results.* After we determine the final feature set, it is necessary to evaluate the results achieved using the final feature set (referred further as the “final classifier”) compared to the performance of the basic classifier.

The overall cross-validation accuracy of the interval classification for the Lomonosov-2 real-life applications improved from  $\sim 0.93$  with the basic classifier to 0.95 with the final classifier. This means that the final accuracy is above the  $t$ -criterion value, which is 0.94 (calculated in the beginning of Sect. 3), so that the achieved accuracy improvement is statistically significant.

The confusion matrix (Table 1) summarizes the classification accuracy for particular classes (average values for 1000 cross-validation runs). It can be seen that our final classifier works best with the normal class, but the results for other classes are also quite high.

**Table 1.** Confusion matrix for interval classification

		Predicted class		
		Normal	Abnormal	Suspicious
Actual class	Normal	268	0	6
	Abnormal	2	62	5
	Suspicious	10	1	168

False-negative error is also an important measurement in our case. It is not a big issue to misclassify a few normal jobs as suspicious or abnormal since we want just to notify users about anomalies found. But it is not acceptable to miss abnormal or suspicious behavior since it leads to loss of computing resources. The results show that this error is very small for the final set: 0.027 in average.

We have also compared other classification performance metrics, such as the  $F$ -score and the false-negative error.  $F$ -score results for both the basic and the final versions are summarized in Table 2. The  $F$ -score helps to evaluate the classification results from another point of view. It is calculated using the following formula (2):

$$F = 2 * (precision * recall) / (precision + recall). \quad (2)$$

According to Table 2, the  $F$ -score also improved for each class. This is especially true for suspicious jobs, where it increased from 0.892 to 0.935.

**Table 2.** Comparison of  $F$ -score values for the basic and the final classifiers.  $F$ -score calculated independently for each class

	$F$ -score (normal)	$F$ -score (abnormal)	$F$ -score (suspicious)
Basic classifier	0.944	0.918	0.892
Final classifier	0.966	0.93	0.935

All the described results for interval classification show that the accuracy improved compared to the basic version. But our final goal is to detect jobs with abnormal behavior, so we need to evaluate job classification results as well. The performance for the job classification was tested on previously unclassified real-life jobs from the Lomonosov-2 supercomputer. We detected 190 suspicious and 64 abnormal jobs with our final classifier based on the analysis of  $\sim 10$  days

of the Lomonosov-2 functioning. These results were manually validated, leading to accuracies of 0.98 and 0.95 for abnormal and suspicious jobs, respectively.

Thus, it can be seen that the final feature set obtained using discriminant analysis enabled us to improve the overall classification accuracy. Moreover, the data preparation process described in this section made the overall classification process much more portable. The next section is devoted to this topic.

## 4 Methodology for Applying Anomaly Detection Method to Other Supercomputers

The methodology in this section describes in detail the process of applying the solution proposed for anomaly detection to other supercomputing systems. At the top level, this methodology is quite universal and, in fact, is suitable for most machine learning based classification methods; however, more specific details of this process relate to this method in particular. A description of the sequential steps of this methodology is provided below.

**1. Prepare core software tools based on the proposed methods.** This can be implemented manually using the description given in this paper, or the source code that we plan to upload to GitHub in the near future. This core software should include methods that are independent of the supercomputer it is used on: method for partitioning the job timeline into intervals, interval classifier, and job classifier based on the interval classification results.

**2. Determine the range of possible input data.** In this step, all data that can potentially be useful for the interval classification process should be selected. As in our case, which was described in Sect. 3, this includes selection of data types and integral values used for aggregation. It is worth recalling that integral values are used for triple aggregation: by time and twice by space (between cores in a node and between nodes). We believe that both data types and integral values chosen in our case can serve as a good starting point by default, but a user may make his own changes in these lists if needed, according to his knowledge of the usual behavior of jobs on the target supercomputer.

**3. Implement data collection using a monitoring system.** For detection of anomalies, we need the input data that were described in the previous step. This is provided by a monitoring system, so one should be installed and configured at the target supercomputer. This step is done outside of our anomaly detection process, so we do not specify how this step should be done.

Nevertheless, several remarks should be made. Usually, it is impossible to store all the data that a monitoring system can provide for the whole supercomputer, so data aggregation is used (as it was done in our case, see Sect. 3). The frequency of data aggregation can influence the classification performance, and one should keep that in mind: if classification results are too low, a possible reason could be that data are too frequently aggregated.

The second remark is that this step can reduce the range of possible data from step 2. This is due to the limitations of the hardware being monitored.

For example, modern processors allow to simultaneously collect data from just a few hardware counters, so we need to select the most needed ones. Moreover, different processor families provide different sets of counters. So it is likely that the range of data specified in step 2 will have to be adjusted after the installation of a monitoring system.

**4. Choose the initial feature set.** At this point we have fixed the data that can possibly be used for classification. So now the initial feature set that will be used in the basic (reference) version of the classifier can be determined. This set is chosen on the basis of our understanding of a feature importance since, at this point, we have no analytical insights on which features are more important for classification.

Creating this basic version is done for several reasons. Based on the results of the reference version, we can decide whether the proposed approach is applicable in general in this case. Working with the basic version, we get first assumptions on the preferable size of the feature set, the time needed for classification, etc. Next, we can use reference classification results for comparison with all other versions that are going to be created in future. And also, this allows us to verify whether this implementation works correctly on this supercomputer.

**5. Create the training set.** This is both one of the most challenging and one of the least automated steps. We need to scan through real data (real-life supercomputer job flow) and pick out intervals we want to include in the training set. Each interval should be classified as normal, suspicious or abnormal, based on the chosen monitoring data set. Also, since our final goal is to classify applications, we need to create a second training set of classified jobs, so we must assign a class to each job according to its classified intervals. It is not necessary to use every interval from a job in the training set; only the most useful ones may be included. But if not all intervals in a job are classified, this job normally should not be included in the second training set for the job classifier (the one based on the simple criteria, not on ML techniques) since the result in this case can be incorrect.

In our experience, the training sets do not need to be very big: 500+ intervals from 100+ jobs were enough for our classifier.

There are several general rules that should be followed during this process:

- The manual interval classification for the training set should be based on exactly the same data that will be used in the main classification process.
- The training set should include as many different types of dynamic behavior as one wants the classifier to identify. If one type of behavior is not present in the training set, then the classifier is likely to misclassify it.
- The numbers of intervals in the classes of the training set should not differ significantly. Otherwise, the classification results can be incorrectly biased to the more popular class.
- A suspicious class can be divided into subclasses if desired. This can make classification results more informative but only if the division into subclasses in the training set was made accurately enough. There is usually no point

in detecting subclasses in a normal class (since there are no performance issues in such jobs) as well as in an abnormal class (since such jobs behavior normally is not so diverse). At the same time, there are many possible types of dynamic behavior for suspicious jobs. For example, on the Lomonosov-2 supercomputer, we have manually found such types of specific behavior as “1 active process on each node”, “1 active process on all nodes”, “stalled because of Lustre issues”, etc. It should be noted that the classifier developed has not been tested in practice with subclass division, but this should work out of the box.

**6. Configure proposed core software.** The methods developed in the core software have a number of input parameters that can be configured. All input parameters can be used with default values but some of them depend on the target supercomputer peculiarities, so it is recommended to analyze if more suitable values can be used.

The method for partitioning the job timeline into intervals has only one parameter: the minimal number of time points in an interval. In our case, it was decided that each interval should be not less than 30 min, otherwise the number of intervals in the job could be too large. Since 2-min aggregation is used on the Lomonosov-2 supercomputer, it results in a minimum of 15 time points per interval.

The main interval classifier has two internal parameters that are used for tuning the Random Forest algorithm: (1) the number of trees in the ensemble, and (2) a measure for choosing the optimal database split (impurity). On the Lomonosov-2 supercomputer, 256 decision trees are used in production mode and 32 in test mode (using less trees speeds up the classification process significantly and leads to only a slight decrease in accuracy); increasing this value does not lead to any significant changes in classification accuracy but causes a decrease in speed. Also, Gini impurity measure is used since it tends to provide more accurate results. We believe that it is not necessary to change these parameters in most cases, but they may be adjusted if needed.

The last method developed is the job classifier based on interval classification results. It uses a set of criteria (see [8]) based on constant thresholds that generally determine how much CPU hours (and what part of the overall job) are consumed by abnormal/suspicious intervals. It is recommended to configure these parameters based on the job flow structure of the target supercomputer, since the default thresholds were specifically adjusted for the Lomonosov-2 supercomputer.

**7. Run the classification using the initial feature set.** All the preliminary steps are done, so now it is time to run the classification for the first time, using data from the initial feature set. If the accuracy results are unsatisfying, it may be necessary to rethink steps 2 (in the part related to the aggregation implementation), 5 or 6.

**8. Search for a suitable feature set using the forward stepwise method.** This step is devoted to the feature selection, which was described in detail in

Sect. 3. This is another one of the most challenging steps, along with step 5, since there is a lot of possible ways to implement it. According to the results from Sect. 3, the following steps should be carried out:

1. *Choose forward stepwise parameters.* This includes the initial set of features to start from, as well as the break criterion (at what point the forward stepwise algorithm must stop). By default, values specified in this paper may be used but these can be altered if needed. This step also includes changing the forward stepwise method to other possible methods, but we hope this will be necessary on rare occasions.
2. *Run the forward stepwise method.* As a result, a new feature set will be defined.
3. *Tune the feature set.* It is always useful to add more semantic knowledge to the classification process. If some features are known to be meaningful for classification, it is likely that they should be added to the resulting feature set. But this should be done with caution, since it is very hard, in our experience, to understand all the dependencies in the performance data collected for supercomputer jobs.
3. *Check the accuracy.* After the feature set is formed, it is necessary to check the classification accuracy. The result can be now compared to the reference results achieved with the initial feature set.
4. *Repeat if necessary.* If the classifier works poorly, return to the feature set tuning in this step. If this does not help, return to step 5 or 6.

**9. Verify the results.** At this step, we have developed a working classifier that shows, hopefully, high-performance results. However, this was achieved on training data, and the classification accuracy on real-life data can be slightly different owing to possible shortcomings of the training set (see step 5). So we need to verify the accuracy of the resulting classifier on unclassified real-life data. This can be done in a similar way as shown in Subsect. 3.1.

## 5 Conclusions

This paper describes the data preparation process for a machine learning method used for anomaly detection in a supercomputer job flow. The main goal is to determine what data types should be included in the feature set and how this data should be aggregated. Discriminant analysis methods were used for this purpose. The best results were obtained with the forward stepwise method, resulting in a new feature set that helped to increase the accuracy from 0.93 to 0.95.

The research conducted in this paper have shown that the basic classifier with the initial feature set shows a very good accuracy which can only be slightly improved. But the developed method for choosing an appropriate feature set has allowed us to make the overall anomaly detection solution much more portable. Thus, a methodology for applying this solution to other supercomputer systems has been proposed, which is also described within this paper.

In the future, we plan to further apply the proposed anomaly detection solution in practice. It is planned to implement an online job classification which would allow us to promptly notify supercomputer users about their running jobs that exhibit a suspicious behavior. Also, we are looking forward to trying our solution on other supercomputers, so we could analyze its performance and portability and make further improvements if required.

## References

1. Voevodin, V., Voevodin, V.: Efficiency of exascale supercomputer centers and supercomputing education. In: Gitler, I., Klapp, J. (eds.) ISUM 2015. CCIS, vol. 595, pp. 14–23. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-32243-8\\_2](https://doi.org/10.1007/978-3-319-32243-8_2)
2. Shvets, P., Voevodin, V., Zhumatiy, S.: Statistics of software package usage in supercomputer complexes. In: Proceedings of the 3rd Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists. CEUR Workshop Proceedings, vol. 1990, pp. 20–29 (2017)
3. Gallo, S.M., et al.: Analysis of XDMoD/SUPReMM data using machine learning techniques. In: 2015 IEEE International Conference on Cluster Computing, pp. 642–649. IEEE, September 2015. <https://doi.org/10.1109/CLUSTER.2015.114>
4. Tuncer, O., et al.: Diagnosing performance variations in HPC applications using machine learning. In: Kunkel, J.M., Yokota, R., Balaji, P., Keyes, D. (eds.) ISC 2017. LNCS, vol. 10266, pp. 355–373. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-58667-0\\_19](https://doi.org/10.1007/978-3-319-58667-0_19)
5. Zhang, H., You, H., Hadri, B., Fahey, M.: HPC usage behavior analysis and performance estimation with machine learning techniques. In: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), p. 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp) (2012)
6. Sidnev, A., Gergel, V.: Automatic selection of the fastest algorithm implementations. Numer. Methods Program.: Adv. Comput. **15**(4), 579–592 (2014). (in Russian)
7. Stefanov, K., Voevodin, V., Zhumatiy, S., Voevodin, V.: Dynamically reconfigurable distributed modular monitoring system for supercomputers (DiMMon). Procedia Comput. Sci. **66**, 625–634 (2015). <https://doi.org/10.1016/j.procs.2015.11.071>
8. Shaykhislamov, D., Voevodin, V.: An approach for detecting abnormal parallel applications based on time series analysis methods. In: Wyrzykowski, R., Dongarra, J., Deelman, E., Karczewski, K. (eds.) PPAM 2017. LNCS, vol. 10777, pp. 359–369. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-78024-5\\_32](https://doi.org/10.1007/978-3-319-78024-5_32)
9. Pedregosa, F.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**(Oct), 2825–2830 (2011)
10. How to Prepare Data for Machine Learning. <https://machinelearningmastery.com/how-to-prepare-data-for-machine-learning/>
11. Measures of Skewness and Kurtosis. <http://www.itl.nist.gov/div898/handbook/eda/section3/eda35b.htm>
12. Feature Elimination Implementation in Scikit-Learn. [http://scikit-learn.org/stable/modules/generated/sklearn.feature\\_selection.RFECV.html](http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFECV.html)