Automated Triage of Performance Change Points Using Time Series Analysis and Machine Learning

Data Challenge Paper

André Bauer, Martin Straesser, Lukas Beierlieb, Maximilian Meissner, and Samuel Kounev
University of Würzburg
Würzburg, Germany
andre.bauer@uni-wuerzburg.de

ABSTRACT

Performance regression testing is a foundation of modern DevOps processes and pipelines. Thus, the detection of change points, i.e., updates or commits that cause a significant change in the performance of the software, is of special importance. Typically, validating potential change points relies on humans, which is a considerable bottleneck and costs time and effort. This work proposes a solution to classify and detect change points automatically. On the performance test data set provided by MongoDB, our approach classifies potential change points with an AUC of 95.8% and accuracy of 94.3%, whereas the detection and classification of change points based on previous and the current commits exhibits an AUC of 92.0% and accuracy of 84.3%. In both cases, our approach can save time-consuming and costly human work.

CCS CONCEPTS

• General and reference → Measurement; **Metrics**; **• Computing methodologies → Machine learning**; **• Software and its engineering → Software performance**.

KEYWORDS

ICPE data challenge, change point classification, change point detection, automatic testing, performance regression testing

ACM Reference Format:
André Bauer, Martin Straesser, Lukas Beierlieb, Maximilian Meissner, and Samuel Kounev. 2022. Automated Triage of Performance Change Points Using Time Series Analysis and Machine Learning: Data Challenge Paper. In *Companion of the 2022 ACM/SPEC International Conference on Performance Engineering (ICPE '22 Companion), April 9–13, 2022, Bejing, China.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3491204.3527486

1 INTRODUCTION

With the emerging DevOps principle, performance regression testing has become an essential process in software development to keep track of the performance as one of the quality attributes of the software product. Automating performance regression testing is necessary to cope with the high frequency of commits and releases. The detection of change points, i.e., updates or commits that cause a significant change in the performance of the software, is of particular importance. If the detected performance change is negative, the rollout of the version might be stopped.

MongoDB is known as a pioneer in the area of performance regression testing and change point detection [4, 6]. However, their proposed approaches still rely on human power to validate (triage) the results of the change points algorithm. This creates a new bottleneck in the testing pipeline and further costs. Our solution process is divided into three steps to approach this problem: First, we have to develop a model that can classify detected points in false and true positives. This results in RQ1: *How to describe a change point for machine learning models?*. Second, we evaluate the performance of different models on pre-labeled data from the MongoDB data set [5] to answer RQ2: *Which model is suitable for the automated triage of pre-labeled performance change points?* As a final step, we need to apply our resulting model to unlabeled data to tackle RQ3: *How can the model be applied on unlabeled data?*

We propose a novel approach for the automated detection and triage of change points to address the research questions. The key idea is to describe the commits before as well as after a potential change point with time series characteristics and use them as features to classify this point. Moreover, we evaluate the performance of different machine learning models for the automated triage of change points. The results show that a random forest model can classify pre-labeled data from MongoDB with an AUC (area under the curve) of 95.8% and accuracy of 94.3%. Then, we derive a window size heuristic to apply the model to unlabeled data. On this unlabeled data, our approach finds 698,766 additional change points and exhibits an AUC of 92.0% and accuracy of 84.3%. Both the approach and the results are available at GitHub[1]. To sum up, our approach can be used to classify already found potential change points or detect and classify change points directly results of performance regression tests. In both cases, our approach can save time-consuming and costly human work.

The remainder of our paper is structured as follows: In Section 2, we introduce the applied time series characteristics and machine learning methods and summarize preliminary work of MongoDB. In Section 3, we describe and evaluate the methodology for classifying potential change points. Section 4 presents and evaluates the change point detection algorithm. In Section 5, we discuss threats to validity, while Section 6 concludes the paper.

ICPE '22 Companion, April 9–13, 2022, Bejing, China
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9159-7/22/04…$15.00
https://doi.org/10.1145/3491204.3527486

[1] GitHub: https://github.com/DescartesResearch/ICPE_DATA_CHALLENGE22

2 BACKGROUND

2.1 Time Series Characteristics

In general, the observations forming a time series are assumed to be recorded in consecutive and equidistant time steps (e.g., days). In the case of MongoDB's performance data, this assumption does not hold because the performance data are time-discrete measurements: A performance test is only triggered after a commit, and the resulting measurement was therefore recorded at irregular intervals. Consequently, this collection of performance measurements forms an unevenly distributed time series. In the following, we refer to the univariate, unevenly distributed time series just as time series.

Since time series can vary in length and are therefore difficult to compare, we transform a time series from the time domain to a feature domain. This transformation has the advantage that each time series has the same number of features regardless of its length. An appropriate combination of time series features (also referred to as time series characteristics) is required to train machine learning models with high accuracy to detect change points. To this end, we applied a genetic search algorithm to find such a subset out of 20 considered time series characteristics [1, 9, 11]. The genetic search algorithm was configured with a population size of 200, stopped after 71 generations, and used the classification accuracy as the fitness function. In total, 10,752 time series characteristics combinations were tested. The resulting eight characteristics are described as follows:

Crossing points: The number a time series crosses the median line.

Spectral entropy: This characteristic is the Shannon entropy of the spectral density of the time series [8] and computed as

$$-\int_{-\pi}^{\pi} \hat{f}(\lambda) \log \hat{f}(\lambda) d\lambda, \tag{1}$$

where $\hat{f}(\lambda)$ is an estimate of the spectral density of the time series.

Max shift: The largest mean shift between two consecutive sliding windows of a time series.

Norm mean and norm sd: The mean and standard deviation of the min-max-normalized values of a time series.

Serial Correlation: This characteristic describes the correlation of the time series with itself to an earlier time and computed as

$$n \cdot \sum_{k=1}^{l} r_k(Y)^2, \tag{2}$$

where r_k is the auto-correlation coefficient with lag k, n the length of the time series, l the considered lag of the time series, and Y the time series.

Stability and *lumpiness*: For calculating these characteristics, the time series is broken down into non-overlapping sub-time series. For each of these sub-time series, the means and variances are calculated. The *stability* is the variance of the means of the sub-time series, and the *lumpiness* is the variance of the variances of the sub-time series.

2.2 Random Forest

A *random forest* [2, 7] consists of multiple decision trees, where the prediction is the average (regression) or majority (classification) of each tree's output. The method trains several decision trees on the bootstrapped training samples, but for each split, only a random sample of the features is considered in each tree. The decision trees within the ensemble focus not only on dominant features but also on features that would not be selected for the top split.

2.3 Support Vector Machines

SVMs [10] are typically used for classification and pattern recognition. For example, in binary classification, the basic idea of SVM is to find a linear separator that partitions the data into two classes. The separation line is fit so that the margin between the line and the borderline cases is maximized. In other words, the training samples are represented by their feature vectors in high-dimensional space, and the SVM is trained to find a line where all samples from one class are on one side and all other samples are on the other side.

2.4 Extreme Gradient Boosting

XGBoost [3] is an ensemble of decision trees based on gradient boosting. Here, the trees are grown sequentially in the boosting approach. Each tree is grown on a modified version of the original data set while using the previously grown trees' information. Each tree learns from its predecessors and updates the residual errors. That is, each subsequent tree is fit to the current residuals instead of the target. The newly grown tree is then added to the fitted function to update the residuals. This procedure is repeated until the accuracy is no longer improved. Each tree is kept to a minimum to approach the dependence on past trees. Also, the contribution of each tree to the final model is unequally weighted.

2.5 Related Work

Daly et al. [6] describe their experience of employing change point detection to performance measurements to identify commits that introduce performance changes. The initial approach comprised humans looking at graphs and manually picking change points. This expensive, error-prone, hardly scalable methodology was superseded by a threshold-based algorithm. However, choosing a threshold capable of detecting change points while ignoring performance variations due to noise proved to be infeasible. A large number of false positives resulted in only 1% of automatic notifications being relevant. Applying E-Divisive Means, a statistical analysis approach for change point classification, showed much better results, with a 67% percentage of useful automatic reports.

In a follow-up, Daly [4] elaborates on the changes of the testing infrastructure, where the change point detection is embedded, and their impact on workflows and development efficiency. A key takeaway here is that increases in testing frequency and extensiveness require intelligent automation; otherwise, humans cannot comprehend the results on time. In return, more significant amounts of timely feedback positively affect developer productivity and ultimately result in better software.

3 CLASSIFICATION OF CHANGE POINTS

3.1 Preparing the Data Set for the Classification

The main idea of our approach is to use machine learning to predict whether a given point is a change point or not. Each point represents a commit and its reported performance measurements - in the case of the MongoDB data set aggregated measures like the

90^{th} percentile. In a first step, we retrieve from the data set 7,795 time series comprising 15,331 triaged (labeled as true positive or false positive) points by MongoDB. Then, for each triaged point, we extract two sub-time series from the time series containing that point: *Pre* and *After*. The idea of determining *Pre* and *After* is depicted in Figure 1. *Pre* starts at the previous confirmed change point p^{-} (labeled as true positive) and ends before the potential change point p. *After* starts with p and ends before the next confirmed change point p^{+}. Mathematically, let I_j be the index of point j, then Pre $= [I_{p^-} + 1; I_p - 1]$ and After $= [I_p; I_{p^+} - 1]$. If there is no confirmed change point before or after q, the start is set to the first point or the last point of the time series, respectively.

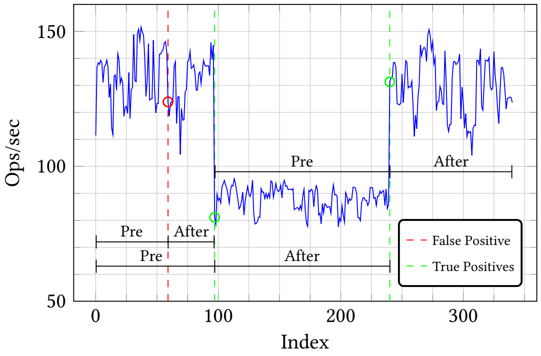

Figure 1: Example extraction of sub-time series.

3.2 Training the Classification Algorithms

As described in Section 2, we transform both *Pre* and *After* from the time domain to the feature domain, since machine learning methods require a fixed input size. That is, each sub-time series is now described by a set of eight features (see Section 2.1). Each time series characteristic is calculated separately on *Pre* and *After*, except for *norm mean* and *norm sd*. For these two characteristics, the min-max-normalization takes place on the merged *Pre* and *After* sub-time series of a potential change point.

The characteristics of *Pre* and *After* are the features[2] and the target are the labels of the triaged points (true positive or false positive). To train and test the methods described in Section 2, we split this data into 80% train and 20% test set. Furthermore, the train set is further divided to use 10% for validation. To avoid arbitrary partitions, we split the data 100 times randomly. All methods were optimized and trained with the *R* caret package.

3.3 Machine Learning Approaches Comparison

To determine which machine learning method performs best for the change point classification (RQ2), we compare three state-of-the-art methods against each other. Each method was trained and tested on the 100 splits as described in Section 3.2. The results are shown in Table 1. Each cell shows the mean value and the 95% confidence interval. The best values are highlighted in bold. Random forest exhibits the best values for AUC (area under the curve), accuracy, sensitivity, and specificity, all over 94%. Therefore, we chose this method for our approach and further experiments.

Measure	Random Forest	SVM	XGBoost
AUC [%]	**98.50** [98.46;98.54]	94.54 [94.46;94.62]	94.60 [94.52;94.68]
Accuracy [%]	**94.34** [94.25;94.43]	89.03 [88.92;89.14]	88.38 [88.25;88.51]
Sensitivity [%]	**94.40** [94.27;94.53]	87.36 [87.17;87.55]	86.77 [86.57;86.96]
Specificity [%]	**94.29** [94.17;94.42]	90.44 [90.30;90.58]	89.74 [89.58;89.90]

Table 1: Comparison of the machine learning approaches.

We applied the trained random forest model to the commits in the data set labeled as not triaged. That is, we calculated *Pre* and *After* for each of these points and fed this information into the model. Our model classified 40% of these commits as true change points. The list of these classified change points is available online[3].

3.4 Investigation of Window Size

So far, our approach can only classify potential change points if information about all change points within a time series is available. To find a change point as soon as possible, this approach is not feasible. Therefore, we investigate in this section how many commits need to be considered to classify a potential change point. Similar to Section 3.1, we generate *Pre* and *After* for a potential change point, but this time *After* has a limited window size w. In order not to corrupt the results in the following experiment, i.e., having multiple true change points in a window, *After* ends after the minimum of $(I_p + w)$ or $(I_{p^+} - 1)$, where I_p is the index of the potential change point and I_{p^+} the index of the next confirmed change point.

The value w ranges from 0 to 10, where 0 means that we consider only the potential change point as *After*. The results are depicted in Figure 2, where each point represents the average over the 100 splits and shows the 95% confidence interval. The dashed line illustrates the AUC value for the adaptive splitting as described in Section 3.1. The AUC improves with increasing window size up to 9, after which it begins to deteriorate. Remarkably, window size 0 has an AUC of 96.5%. Consequently, we can determine with high accuracy whether a commit is a change point regardless of subsequent commits.

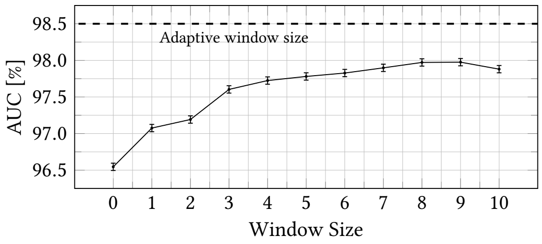

Figure 2: Comparison of different window sizes

[2] We also tried the difference and the change of these characteristics but considering the characteristics of *Pre* and *After* yielded the best results.

[3] Classified commits: https://github.com/DescartesResearch/ICPE_DATA_CHALLENGE22/blob/main/Output/classified_points.csv

4 DETECTION OF CHANGE POINTS

4.1 Detection Algorithm

Based on the insights gained in the previous section, we introduce a change point detection algorithm that decides whether a commit is a change point based on previous commits and the current commit. The basic idea is that our approach is applied every time a performance measurement of a commit is available and classifies this commit either as a change point or not. Therefore, the algorithm maintains a list of change points that it has previously found. The change point detection workflow is shown in Algorithm 1. The algorithm extracts from the previous commits the *Pre* sub-time series (see Section 3.1). If the algorithm had previously detected a change point, *Pre* starts from this point; otherwise, *Pre* contains all previous commits. For *After*, only the current commit is considered. Then, the time series features (see Section 2.1) for *Pre* and *After* are calculated and passed to the classification model trained on windows size 0 (see Section 3.4). Finally, the algorithm returns whether the current commit is a change point or not.

Algorithm 1: Change point detection workflow.

```
Input: Previous commits $ts$, current commit $c$, found change points
       $cp$, classification model $model$
Result: Classification of current commit
$s = 0$;
if $cp \neq \emptyset$ then
    $s = cp$.top(); // get index of last change point
$Pre = ts[s$ to end]; // only commits since index s
$After = c$; // After is only current commit
$features$ = calculateCharacteristics($Pre$,$After$); // see Sec. 3.2
$b$ = classify($features$, $model$);
if $b$ then // classified as change point
    $cp$.push(length($ts$)+1); // add index of current measurement
return $b$
```

4.2 Evaluation

To evaluate our detection approach, we apply the algorithm for each commit in each time series in the data set to detect whether this point is a change point or not. In total, we investigated 21,913,750 points in 795,171 time series contained in the MongoDB dataset. Here, we found 775,541 change points. Comparing these found points with the labeled points by MongoDB, we found 5,304 change points that are labeled by MongoDB as change points (true positive), 729 as false positives, and 270,189 as not triaged or under investigation. That means that our detection approach has an AUC of 92.0% and accuracy of 84.3% on the labeled points by MongoDB. The decrease in these two measures can be explained by the fact that, unlike the classification task in Section 3, the algorithm does not have information about which points are change points, leading to different *Pre* sub-time series when a change point is not found or is found incorrectly. The remaining 698,766 change points were not found by the detection algorithm of MongoDB. The list of detected change points can be found online[4].

5 DISCUSSION AND THREATS TO VALIDITY

As our approach was trained and tested on the data set provided by MongoDB, the approach is tailored to the data provided by MongoDB. Thus the results may not be generalizable. However, the approach can easily be adapted to other data, i.e., training the classification algorithm on these data. To classify and detect whether a commit is a change point, we rely on the labeling previously done by MongoDB. As the labeling provided by MongoDB was done manually by a human expert, it may be prone to error. Consequently, our approach may propagate this error. However, unlike MongoDB's current approach, our approach detects change points immediately after reporting the result of the commit, without requiring a human expert to intervene. Moreover, the found 769,508 change points that are labeled as not triaged or were not found by the detection algorithm of MongoDB have to be triaged manually by an expert of MongoDB. Still, due to the high accuracy of our approach, we are confident that these are real change points.

6 CONCLUSION

In this work, we present a novel machine learning-based approach to detect change points in performance regression tests automatically. To train the model, we describe the commits before and after a potential change point with time series characteristics (RQ1). Then, we compare three different machine learning methods, where random forest yields the best results (RQ2). To apply the classification on unlabeled data, we investigate different window sizes and develop an algorithm that iterates over the commits to detect change points (RQ3). The classification on the pre-labeled data exhibits an AUC of 98.5%, while the detection algorithm exhibits an AUC of 92.0%. Our approach can classify potential change points already found or detect and classify change points directly from performance regression tests. In both cases, our approach can save or reduce time-consuming and costly human work.

REFERENCES

[1] André Bauer, Marwin Züfle, Johannes Grohmann, Norbert Schmitt, Nikolas Herbst, and Samuel Kounev. 2020. An Automated Forecasting Framework based on Method Recommendation for Seasonal Time Series. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*. Association for Computing Machinery (ACM), 48–55.

[2] Leo Breiman. 2001. Random Forests. *Machine learning* 45, 1 (2001), 5–32.

[3] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A Scalable Tree Boosting System. In *ACM SIGKDD 2016*. ACM, 785–794.

[4] David Daly. 2021. Creating a Virtuous Cycle in Performance Testing at MongoDB. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*. 33–41.

[5] David Daly. 2021. MongoDB Performance Test Result Dataset. https://doi.org/10.5281/zenodo.5138516

[6] David Daly, William Brown, Henrik Ingo, Jim O'Leary, and David Bradford. 2020. The use of change point detection to identify software performance regressions in a continuous integration system. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*. 67–75.

[7] Tin Kam Ho. 1995. Random Decision Forests. In *Proceedings of 3rd international conference on document analysis and recognition*, Vol. 1. IEEE, 278–282.

[8] Jaewoo Jung and Jerry D Gibson. 2006. The interpretation of spectral entropy based upon rate distortion functions. In *2006 IEEE International Symposium on Information Theory*. IEEE, 277–281.

[9] Christiane Lemke and Bogdan Gabrys. 2010. Meta-learning for Time Series Forecasting and Forecast Combination. *Neurocomputing* 73, 10-12, 2006–2016.

[10] Vladimir Vapnik. 1995. The Nature of Statistical Learning.

[11] Xiaozhe Wang, Kate Smith-Miles, and Rob Hyndman. 2009. Rule Induction for Forecasting Method Selection: Meta-Learning the Characteristics of Univariate Time Series. *Neurocomputing* 72, 10 - 12 (2009), 2581–2594.

[4] Detected change points: https://github.com/DescartesResearch/ICPE_DATA_CHALLENGE22/blob/main/Output/detected_change_points.csv