On the Difficulties of Supervised Event Prediction based on Unbalanced Real-World Data in Multi-System Monitoring

Andreas Schörgenhumer[1], Mario Kahlhofer[1], Peter Chalupar[1],
Hanspeter Mössenböck[2], Paul Grünbacher[3]
{firstname.lastname}@jku.at
[1] Christian Doppler Laboratory MEVSS, Johannes Kepler University Linz, Austria
[2] Institute for System Software, Johannes Kepler University Linz, Austria
[3] Institute for Software Systems Engineering, Johannes Kepler University Linz, Austria

Abstract

Online failure prediction of performance-critical events is an important task in fault management of software systems. In this paper, we extend our previous multi-system event prediction by analyzing its performance on unbalanced, real-world data, which represents a realistic online scenario. We train a random forest classifier with different data preprocessing configurations, including data augmentation to cope with the extreme class imbalance. The results reveal that the prediction quality of the tested multi-system model drops significantly compared to the balanced scenario. Although our supervised event prediction approach as well as different data preprocessing configurations turned out to be ineffective, we consider the insights of our work valuable for the community.

1 Introduction

Preventing slowdowns, failures and other deviations from expected behavior is an important aspect in managing the operation of software systems [4]. Rare events indicating an abnormal state of the system are called anomalies. Numerous approaches have been proposed to automate their detection. If labels are available, supervised anomaly detection [2] can be pursued, which uses data indicating the possible cause of such events and then creates machine learning (ML) models to predict events based on new data.

In earlier work [7], we investigated such a problem by using time series monitoring data (CPU, memory, disk, network) to predict so-called service slowdown events with quite promising results (median F1-score of 85%). We performed our evaluation on *balanced*[1] data to explore whether such an approach could yield acceptable results. However, our experiment did not represent a real-world testing scenario with rare anomaly events and therefore *unbalanced*[2] data. Unfortunately, this is where major problems arise [2], most notably due to high class imbalance with only a small portion of the data labeled as anomalous. Many ML models, e.g., the random forest classifier [1], strive to achieve high accuracies, so appropriate steps must be taken before training to avoid that the models simply predict the majority class[3]. Common techniques for preprocessing the training data include under-sampling, over-sampling and combinations of both [3]. The prediction must then be performed on the unbalanced test data in order to yield meaningful insights for real-world scenarios.

We focus our research on multi-system environments that collect monitoring data from multiple systems. Such environments can bring benefits like building more general models to predict yet unseen systems or allowing to deal with limited data (e.g., use rare events of multiple systems). We extend our previous approach on multi-system environments [7] and perform an evaluation based on a real-world test scenario. We again use industrial monitoring data provided by an industry partner, which consists of time series of CPU, memory, disk and network metrics from selected, multiple software systems, with the goal to predict service slowdown events. Due to the sheer amount of configuration options during data selection, preprocessing, hyper-parameter tuning, etc., we first discuss the selected set of configurations and then present the performance of our ML approach on real-world unbalanced test data. We conclude this paper by discussing our approach and the issues that arose.

2 Data

This work is an extension of [7] and uses the same monitoring data of multiple software systems. Each system is a set of connected entities such as hosts, disks, network interfaces and services. Table 1 lists the 34 time series data for the first three entity types, which are provided in 1-minute resolution. Services represent the business logic of a system. Our events

[1] Balanced means that each class has equally many observations (here: normal data class, anomaly class).

[2] Unbalanced means that there is a majority class with more observations (here: normal data class) and a minority class with fewer observations (here: anomaly class).

[3] For example, a class imbalance of 99% to 1% and a model predicting the majority class would lead to an accuracy of 99%.

of interest (the *service slowdowns*) occur at these entities. An event is fired in case the response time of a service exceeds a threshold compared to its baseline. Service slowdowns are very rare in comparison to normal behavior: The data in our test set has a class imbalance of 99.74% to 0.26%.

Host	Disk	Network
CPU idle [%]	Space avail. [b]	Bytes recv. [b/s]
CPU system [%]	Space used [b]	Bytes sent [b/s]
CPU load [c]	Space free [%]	Recv. packets [/s]
CPU user [%]	Read bytes [b/s]	Sent packets [/s]
CPU IO wait [%]	Write bytes [b/s]	Recv. dropped [/s]
Page Faults [/s]	Read ops. [/s]	Sent dropped [/s]
MEM avail. [%]	Write ops. [/s]	Recv. errors [/s]
MEM avail. [b]	Read time [ms]	Sent errors [/s]
MEM used [b]	Write time [ms]	Recv. util. [%]
SWAP avail. [b]	Util. time [%]	Sent util. [%]
SWAP used [b]	Queue length [c]	
	Inodes avail. [%]	
	Inodes total [c]	

Table 1: Metrics for each entity type with units as subscripts (/s = per second, b = bytes, c = count). Underlined metrics indicate our selected subset.

3 Data Preparation

We used an enhanced version of the framework presented in [6] to process the data with various configuration options. Evaluating all possible configuration options is practically infeasible, so we selected a subset of these configurations. Some configurations are fixed while others change in different test runs.

3.1 Fixed Configurations

Training set sampling. For each system, we created a sample right before every event timestamp, optionally augmenting the sample (see below), yielding p positive samples in total. Afterwards, we created $n = p$ negative samples at random timestamps where no event occurred. This step yielded a balanced training set.

Test set sampling. For each system, we stepped through the entire observation period and created a sample for every 60 minute interval. If an event occurred within the following 60 minutes, we labeled the sample as positive, otherwise as negative. This resulted in an unbalanced, realistic input test set which can be expected in real-world scenarios.

Aggregation. For every sample, we aggregated the data by storing the minimum, maximum, average, standard deviation, median and slope of these observation windows as feature vectors in a CSV file.

3.2 Grid-searched Configurations

Data subset [subset]. We investigated two options: First, we used all 34 metrics. Secondly, we selected a subset of 10 metrics (underlined in Table 1). We chose this subset based on aggregate metrics that still cover the most relevant information (e.g., CPU idle = 100% − system − user − IO wait). We also discarded metrics with too much missing data (e.g., disk read and write times) as well as metrics which are nearly constant throughout the observation period and are not significantly Pearson-correlated with events (e.g., 0 for sent and received network errors).

Observation window [win]. When creating a sample at a given timestamp, we took the preceding 10, 30 and 60 minutes of raw data.[4]

Standardization [standardize]. To show whether our results improve, we standardized every data point before training our models, i.e., for every metric m, we calculated $x_m = \frac{x_m - \mu_m}{\sigma_m}$, where μ_m and σ_m are the mean and the standard deviation.

Augmentation [augment]. Since our ML models are trimmed to accuracy, our training set must be balanced. However, due to the enormous class imbalance in our data, simply creating a negative sample for every positive sample (i.e., randomly choosing only as many negative samples as there are positive ones = random under-sampling) is problematic. The main issue is that the few negative samples most likely do not sufficiently represent the negative data distribution. Therefore, we decided to also over-sample our positive data by including the time-warp augmentation suggested in [5]. Time-warping randomly shifts the timestamps of an observation window by a slight amount, e.g., from $\begin{smallmatrix} t \\ y \end{smallmatrix} = \left[\begin{smallmatrix} 0.00 & 1.00 & 2.00 & 3.00 & 4.00 \\ 5 & 7 & 5 & 8 & 7 \end{smallmatrix}\right]$ to $\begin{smallmatrix} t \\ y \end{smallmatrix} = \left[\begin{smallmatrix} 0.00 & 1.01 & 1.97 & 3.02 & 4.00 \\ 5 & 7 & 5 & 8 & 7 \end{smallmatrix}\right]$, and then reconstructs the values of the original timestamps via interpolation, e.g., $\begin{smallmatrix} t \\ y \end{smallmatrix} = \left[\begin{smallmatrix} 0.00 & 1.00 & 2.00 & 3.00 & 4.00 \\ 5 & 6.98 & 5.09 & 7.94 & 7 \end{smallmatrix}\right]$. We still used the same random under-sampling by creating equally many random negative samples, but with α augmentations, we requested α times more negative samples, thereby significantly increasing the probability to get a representative portion of the negative data distribution. We set α to 250 for a drastically lowered class imbalance of about 55% to 45%.

4 Evaluation

Since we augmented our training data up to 250 times and our test set is based on a sliding window (unbalanced, real-world scenario), we created substantially more samples. Due to limited hardware resources, we thus only selected 50 independent systems with data over an observation period of 18 days. We split this period into 18 day-sized parts for each system and exported both the training set and the test set. We then applied day-based cross-validation, where we took 17 days from the training set for training and the (non-overlapping) remaining day from the test set for testing, yielding a total of 18 runs.

We used a random forest classifier with 100 estimators as our ML prediction model for service slowdown events. To avoid overfitting on larger systems, we bal-

[4] We chose the values for comparability with previous studies and in accordance with domain experts.

	win	subset		¬ subset	
		augment	¬ augment	augment	¬ augment
standardize	10	.098[1] ± .049 .085[2] ± .037	.068 ± .045 .061 ± .030	.101 ± .041 .086 ± .035	.087 ± .037 .068 ± .026
	30	.093 ± .051 .083 ± .035	.081 ± .038 .066 ± .028	<u>.103</u> ± .044 <u>.089</u> ± .034	.098 ± .044 .074 ± .030
	60	.088 ± .044 .081 ± .034	.080 ± .041 .064 ± .027	.082 ± .049 .078 ± .036	.080 ± .053 .065 ± .033
¬ standardize	10	.090 ± .046 .082 ± .036	.074 ± .038 .062 ± .026	.099 ± .047 .085 ± .037	.081 ± .052 .066 ± .033
	30	.097 ± .048 .086 ± .036	.086 ± .052 .069 ± .033	.092 ± .047 .082 ± .036	.085 ± .046 .069 ± .032
	60	.075 ± .047 .074 ± .035	.072 ± .052 .061 ± .031	.084 ± .049 .077 ± .036	.083 ± .046 .066 ± .030

Table 2: [1] = MCC and [2] = F1-score of all configuration combinations showing the mean ± standard deviation of the 18 runs (the observation window size is abbreviated with "win", the negation ¬ means that a configuration is not active). We report the results on the test set. <u>Underlined</u> = best scores.

anced the samples so that every system in the training set is equally represented. Table 2 lists all configuration combination results. Before calculating our evaluation metrics, we normalized the confusion matrix entries based on the sample count of each system, to not distort the results in favor of larger systems. We chose the F1-score and the Matthews correlation coefficient (MCC) as evaluation metrics due to our unbalanced data setting. Both are higher-is-better metrics, with the F1-score $\in [0, 1]$ and the MCC $\in [-1, 1]$.

Clearly, the performance of the ML model on real-world data is rather poor and unusable in practice. With a maximum MCC of 0.103 and F1-score of 0.089, we are only slightly better than a random classifier, which is disappointing. Selecting only a subset of data does not bring any benefits, nor do standardization or different window sizes. Augmenting the data seems to improve the results, but unfortunately, due to the the high standard deviations, it is not statistically significant (t-test, $\alpha = 0.05$) except for one pair of F1-scores.

5 Conclusion and Discussion

The experiences reported in this paper show that testing a multi-system event prediction approach based on infrastructure monitoring data with an unbalanced, real-world setting yields unacceptably poor results compared to a balanced setting. Overall, our data preprocessing steps did not improve the prediction quality. In a single case, we could obtain a significant improvement with data augmentation, however, not to an extent that it would be usable. We hope that other researchers who are working in a similar direction can benefit from the findings in this paper, such as to review initial results with care as they might not appropriately represent a real-world scenario.

There are many aspects to discuss that would require a comprehensive analysis on their own. First, we cannot be sure that every model and every configuration would lead to the same results. There are simply far too many options to be tested in reasonable time. We used a random forest classifier for comparability reasons with our previous studies and also because it does not have many hyper-parameters. Naturally, there exist a plethora of other models and entirely different approaches altogether, which could again be evaluated with various settings and configurations. This also includes endless possibilities to preprocess the time series data, such as detrending and deseasonalizing, which would definitely make sense to investigate. Furthermore, there is ultimately the issue that we cannot be sure that the data really contains information from which the events can be predicted. If they do not, there is no model capable of predicting those events. Finally, the supervised approach may not be the best for such a complex problem. In future work, we will investigate normal time series behavior and check for any discrepancies indicating such events.

Acknowledgements

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged.

References

[1] L. Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001).

[2] V. Chandola, A. Banerjee, and V. Kumar. "Anomaly Detection: A Survey". In: *ACM Comput. Surv.* 41.3 (2009).

[3] E. A. Garcia and H. He. "Learning from Imbalanced Data". In: *IEEE Transactions on Knowledge & Data Engineering* 21.09 (2009).

[4] F. Salfner, M. Lenk, and M. Malek. "A Survey of Online Failure Prediction Methods". In: *ACM Comput. Surv.* 42.3 (2010).

[5] T. T. Um et al. "Data Augmentation of Wearable Sensor Data for Parkinson's Disease Monitoring Using Convolutional Neural Networks". In: *Proceedings of the 19th ACM Int'l. Conf. on Multimodal Interaction*. 2017.

[6] A. Schörgenhumer et al. "A Framework for Preprocessing Multivariate, Topology-Aware Time Series and Event Data in a Multi-System Environment". In: *2019 IEEE 19th Int'l. Symp. on High Assurance Systems Engineering*. 2019.

[7] A. Schörgenhumer et al. "Can We Predict Performance Events with Time Series Data from Monitoring Multiple Systems?" In: *Companion of the 2019 ACM/SPEC Int'l. Conf. on Performance Engineering*. 2019.