Exploratory Study of Machine Learning Techniques for Supporting Failure Prediction

João R. Campos, Marco Vieira, Ernesto Costa
DEI/CISUC, University of Coimbra, Portugal
jrcampos@dei.uc.pt, mvieira@dei.uc.pt, ernesto@dei.uc.pt

***Abstract*—The growing complexity of software makes it difficult or even impossible to detect all faults before deployment, and such residual faults eventually lead to failures at runtime. Online Failure Prediction (OFP) is a technique that attempts to avoid or mitigate such failures by predicting their occurrence based on the analysis of past data and the current state of a system. Given recent technological developments, Machine Learning (ML) algorithms have shown their ability to adapt and extract knowledge in a variety of complex problems, and thus have been used for OFP. Still, they are highly dependent on the problem at hand, and their performance can be influenced by different factors. The problem with most works using ML for OFP is that they focus only on a small set of prediction algorithms and techniques, although there is no comprehensive study to support their choice. In this paper, we present an exploratory analysis of various ML algorithms and techniques on a dataset containing failure data. The results show that, for the same data, different algorithms and techniques directly influence the prediction performance and thus should be carefully selected.**

***Keywords*-Dependability; Failure Prediction; Machine Learning; Classification;**

I. Introduction

The growing complexity of software makes it difficult or even impossible to detect all faults before deployment. Residual faults eventually lead to failures that may impact the availability and reliability of systems and thus compromise the supported business processes. Software faults are in fact recognized as a major cause of system outages [1].

Various techniques have been developed with the purpose of avoiding and handling faults, including best practices for coding, approaches for testing and models for reliability characterization [2]. OFP is a complementary technique that intends to mitigate the potential effects of residual faults, by using past data and the current system state for predicting the potential occurrence of failures [3]. This allows taking preemptive measures to avoid such failures or mitigate their consequences, thus improving dependability attributes, such as availability and reliability [4]. However, despite the great potential of OFP, it is still not widely implemented nowadays.

Recent OFP techniques attempt to predict if a failure will occur in the near future, based on the observation of the target system. This is achieved by monitoring its internal state through different variables (such as memory consumption, CPU usage, etc). Some techniques use information about past failure events, such as the values of the system parameters before the failure. The output of a predictor can then be that a failure is bound to occur, or some continuous value that measures how failure-prone the system is [3].

Given recent technological developments, ML algorithms have shown their ability to adapt and extract knowledge in a variety of complex problems. ML algorithms are able to find (complex) patterns in the data and learn from them without relying on a predetermined model and make predictions on new unseen data based on what was learned. Various approaches relying on ML have already been proposed for OFP (find a survey in [3]). Nonetheless, most of those works are fairly limited. A key problem is that most of them focus on a small set of prediction algorithms and techniques, usually based on what has previously been used in a similar approach. Also, they are usually limited to a very specific context, focusing on predicting specific failure modes, and using only fractions of the available datasets. Such approaches do not take advantage of the recent developments on the ML field as well as the existing computational power, possibly providing sub-optimal solutions, without exploring different techniques.

This paper presents an exploratory study of different ML techniques for supporting failure prediction. For this, three datasets with failure data are used to demonstrate how different techniques can assist in the development of a data-driven study to create prediction models. These datasets were generated using fault injection and considering different software faults targeting Windows XP (SP3). They comprise both virtualized and non-virtualized environments, contain the data of 233 system variables monitored during execution, and focus on two failure modes: *System Hang* and *System Crash*. Multiple algorithms, as well as different data pre-processing techniques, were considered in the experiments. The results are analyzed, and a brief comparison with one of the most used algorithms for OFP, Support Vector Machine (SVM), is provided.

Results indicate that, although SVMs are able to perform well when considering a single failure mode, they are not able to generalize for multi-class prediction of failures. Other algorithms, such as Decision Tree (DT), Neural Network (NN), and Bagging, are able to perform well when predicting both single and multiple failure modes. Furthermore, pre-processing techniques, such as feature selection and data sampling, have shown to be able to improve the performance of many models.

This paper is organized as follows. Section II provides background concepts and discusses related Work. Section III describes the methodologies used in our study and Section IV presents the results obtained. Section V discusses those

results. Section VI addresses the threats to validity. Finally, Section VII concludes and puts forward ideas for future work.

II. Background and Related Work

In this section, we present a brief overview of OFP and ML concepts, as well as related work.

A. Failure Prediction and Fault Injection

OFP is a key technique for proactive fault management. It is concerned with identifying at runtime whether a failure will occur in the near future, by using past data and the current system state [3]. Using such predictions, it is possible to take preemptive measures to avoid, or at least mitigate, the consequences of a failure.

Various methods based on ML have been proposed for OFP [3]. Although there are multiple data inputs that can be used (e.g. log-files and system parameters), one of the most relevant are the system variables monitored during execution (*symptoms monitoring*). In practice, the OFP problem can be defined as a decision process for predicting at time t if a failure is going to occur within a precise time, called lead-time Δt_l. Such prediction can be valid for a given time window, called prediction-window Δt_p. In practice, at time t, the prediction model should predict if there will be a failure in the interval $[t + \Delta t_l, t + \Delta t_l + \Delta t_p]$ [3].

Despite the potential of OFP, it is still not widely implemented, partially because failures are rare events, and thus collecting data for training and testing new methods is a complex endeavor. To address this, several studies have been conducted on the injection of software faults to generate representative failure data [5] [6] [7]. The Generic Software Fault Injection Technique (G-SWFIT) is a fault injection technique that injects realistic software faults in machine-code level [5], and has been used in previous works on OFP to generate failure data [6] [7]. With it, it is possible to generate a considerable amount of realistic failure data in a short time, thus removing one of the main limitations for the use of OFP.

Another influencing factor is that most studies on OFP consider only on a limited set of techniques and algorithms, often based on previous similar works and very specific experimental contexts. In [8], SVMs were used to improve failure prediction of disk-drives, and a method based on Hidden Semi-Markov Models to predict failure-prone system states was used in [9]. More recently, SVMs were used to independently predict two failure modes on specific workloads, using a sliding window, thus considering the time dimension of the data [10]. This approach was also used later to demonstrate the need for evolving prediction models [11], and for the development of an adaptive framework [7].

This paper attempts to demonstrate that there is a wide range of techniques and algorithms that perform differently depending on the nature of the problem, using datasets containing different faultloads, workloads and environments.

B. Machine Learning

In the context of OFP, each instance in the training/testing dataset includes information that states if a failure is going to be observed within a given time. As such, it is considered a *Supervised Learning* problem, thus all the existing algorithms for supervised problems may be used.

Although OFP can also be used for predicting how failure-prone the system is [3], in this work we are concerned with predicting if a failure will occur, which renders it as a *Classification* problem. Briefly, classification problems are concerned with separating data into distinct classes, which are discrete and categorical. Some of the most common classification algorithms are *DT, SVM*, and *NN*. Other methods, called *Ensemble Methods*, have recently gained relevance and consist of a compilation of independent algorithms that are trained, and whose predictions are gathered to work as a whole.

As the complexity of any model usually depends on the number of inputs, which determines both the time and space complexity to train the model, reducing dimensionality may be very important. This can be achieved through *Feature Selection* and/or *Feature Extraction*, which chooses/creates a subset of features based on the original set [12]. Furthermore, it is well accepted that a "powerful computationally intensive procedure operating on a sub-sample of the data may, in fact, provide superior accuracy than a less sophisticated one using the entire database" [13]. As such, *Instance Selection* is used in order to help algorithms to cope with the infeasibility of very large datasets. It can be achieved through various techniques, such as *sampling* and *boosting* [14]. Moreover, imbalanced datasets may compromise the algorithms' performance on the minority classes. For this, there are specific solutions for instance selection, such as *Undersampling* and *Oversampling* [15].

In short, a learning model is good if it produces good predictions on unseen examples. However, saying that a classification algorithm is "good", is just a qualification of how well its inductive bias matches the properties of the data [12]. Despite all the factors that can be taken into account when evaluating an algorithm, one should not forget that whatever conclusions may arise, those are conditioned by the dataset with which the model was developed. To obtain a realistic estimate of the prediction error of a model, ML usually works with two main sets of data: train and test. The test set estimates the generalization error of the final model [16]. The division between these two sets is not trivial, thus various techniques have been developed, such as *Holdout Method*, *Partition/Leave-one-out Methods*, and *Bootstrap Methods* [17].

Evaluating an algorithm requires measuring performance. There are various means for this: *Confusion Matrix (CM)*, *Accuracy*, *Precision*, *Recall*, *F-Score*, and *Receiver Operating Characteristics (ROC)*. Briefly, the samples that are correctly predicted are known as *True Positives* and *True Negatives*. The positive samples that are predicted as negatives are *False Negatives* and the opposite are *False Positives*. Metrics should be carefully used, as they are not independent of the data, and thus can be influenced by their distribution [18].

Transposing the concepts of OFP to ML terms, the monitored system variables are referred to as *features*. The set of values of those variables at a given time t is known as a *sample*, and whether a failure will occur or not for that sample is known as the *class* or *target* of that sample.

III. Methodology

To analyze the behavior and performance of various algorithms and techniques, several configurations need to be tested. Due to the number of possible configurations and experiments, only those deemed relevant are detailed here.

A. Dataset

The data used in this study was generated in a previous work to investigate the possibility of using the injection of realistic software faults on Windows XP (SP3) to generate failure-related data [19]. It has been generated considering two different workloads, one based on the Winrar application, and the other on the COSBI OpenSourceMark benchmark suite. Software faults were injected into the Operating System (OS) by a tool implementing the G-SWFIT technique [20]. With this tool, different faults, characterized by the fault type, fault location, and fault injection time, are injected in specific locations previously selected (the dynamic libraries kernel32.dll and ntdll.dll, used by the system process svchost.exe). To generate data using fault injection, several executions (*runs*) of the workloads are performed (with and without fault injection). A run in which faults are injected is a *Fault Injection Run (FIR)*. If a failure occurs during an FIR, it is considered a *failing run*; if not, it is considered a *non-failing run*. To represent the behavior of the system in the absence of fault injection and failures, *Golden Runs (GRs)* were also considered [20].

The experiments were run in three different environments: a "real" machine (without virtualization), and two virtualized environments, one running VMWare vSphere and the other Citrix XEN [20]. The failure modes considered include System Crash (OS becomes corrupted and the machine crashes or reboots) and System Hang (application or OS becomes unresponsive and must be terminated by force). Each dataset contains 233 numeric variables, chosen by the authors as the best representative set of the system behavior. Further details of the experimental setup and the dataset can be found in [20].

Although faults were injected, not all of them led to the observation of failures. Thus, due to the fact that all the datasets contained a very large number of runs without failures, the data used were limited to a maximum of 100000 samples per dataset (a sample is the set of values for all the monitored variables at a given time, as described in *Section II*). This is large enough to comprise all the failure runs, several runs without fault injection (*golden runs*), and numerous runs where faults were injected but no failures were observed. These were included as, albeit no failure was observed during the run, they may contain relevant information describing a perturbed system, caused by the fault injection. This information can be pertinent to distinguish between an abnormal state (which nonetheless does not lead to failure) and a failure-prone state.

TABLE I
Datasets Class Distribution

Dataset	Control	Hang	Crash
Real	99006	832	145
XEN	98651	1057	186
vSphere	97360	1552	892

Three configurations were tested in this work for the pairs $\Delta t_l, \Delta t_p$: [20,20], [40,20], and [60,20], considering a "short", "medium", and "long" term prediction. The samples were labeled according to the approach proposed in [3], which generates different datasets for each pair of $\Delta t_l, \Delta t_p$. Samples for which no failures were observed within the interval $[t_{sample} + \Delta t_l, t_{sample} + \Delta t_l + \Delta t_p]$ will be from now on referred to as *Control* samples. As an example, the classes distribution for $\Delta t_l = 40, \Delta t_p = 20$ for each dataset can be seen in *Table I*. As we can see, it is a severely imbalanced set of data, where failures are rare events, although hang failures were more often than crash failures.

B. Data Pre-Processing

Although we performed detection of outliers, they were not removed, as they may be symptoms of failures and thus provide relevant information. Furthermore, as most features are based in different scales, a Z-score normalization was applied, which rescales all features so that they have the properties of a standard normal distribution, with zero mean ($\mu = 0$) and a standard deviation of one ($\sigma = 1$) [16].

Some of the faults injected led to failures in a very short timeframe. This behavior is not relevant as, most likely, the fault injected is not representative of a residual fault, nor such information can be used in our context, as there is not enough time to make predictions. This way, runs where the time between the fault injection and the failure observation was smaller than the pair $\Delta t_l, \Delta t_p$ were removed, albeit such cases were rare (4, 6, and 7 runs for pairs [20, 20], [40, 20], and [60, 20], respectively).

To have a better understanding of the datasets, both *descriptive analysis* and *exploratory data analysis* techniques were used. Although some features presented low variance, most had significant dispersion, for both hang and crash failures. However, by analyzing the mean values of the features per class (failure mode) we can observe that *Control* samples and both *Hang* and *Crash* are considerably different. On the other hand, the differences between the *Hang* and *Crash* samples are not so considerable, although for some features they are still differentiable. This is depicted in *Figure 1*, where the x-axis represents the features and the y-axis their normalized mean.

In this work, we considered both *Filter* (by variance, and correlation) and *Wrapper* (through Recursive Feature Elimination (RFE)) feature selection methods. Features with zero variance were removed, as they provide no relevant information. When using feature selection by correlation, features with correlation with others above 90% were removed. The correlation was measured using the Pearson's Correlation

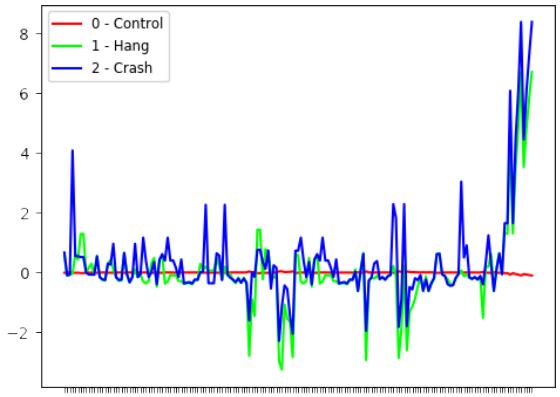

Fig. 1. Features' Mean by Class

Coefficient, which is one of the most common measures [17]. Another method that was used was RFE, using 10 DTs, which were chosen due to the effectiveness and the way the DT algorithm weights features to develop its trees. To choose the number of features to keep, a cross-validation approach was used with each combination of features resulting from the RFE, and in the end, the subset of features that produced better results was kept. The scoring function used for this method was *recall*, to try to benefit those models that could predict the most elements of each class. As an example, for the *Real* dataset considering both failure modes, using $\Delta t_l = 40, \Delta t_p = 20$, it was observed that several features had very high correlations, superior to 90%, thus 85 were removed when using correlation feature selection. For the same dataset, and using RFE, 154 features were discarded, which suggests that for this dataset a small number of features contains most of the required information. Although the set of features reduced per dataset varied, the number of features removed for each feature selection technique was similar.

To study the effect of the imbalance in the data, a simple data undersampling method, *Random Undersampling*, was used. With this technique, samples of the majority class are randomly eliminated until the ratio between the majority and minority class is at the intended level. One of the problems with this approach is that it is not possible to control what information about the majority class is lost. Nonetheless, despite its simplicity, it has shown to be one of the most effective undersampling methods [21].

C. Algorithms Studied

The list of classification algorithms used is as follows: *Naive Bayes*, *Gaussian Process*, *Logistic Regression*, *Decision Tree (DT)*, *k Nearest Neighbors (k-NN)*, *Support Vector Machine (SVM)*, *Neural Network (NN)*. The ensemble methods *Bagging*, *Extra Trees* and *Random Forest (RF)* were also tested. Some of the parameters of the various algorithms were chosen based on existing literature. Due to space restrictions, only the most relevant configurations are described next.

1) Support Vector Machine (SVM): The SVM method used was configured with a Radial Basis Function (RBF) kernel, similar to previous studies [26]. The gamma value for the kernel was set with a relative number: $\frac{1}{n\_features}$. For multi-class, it uses One-Versus-One (OVO) approach to develop its classifiers. That involves creating a model for each pair of classes, which results in $\frac{N(N-1)}{2}$ classifiers. When testing, a majority voting of the votes of each classifier defines the predicted class [22]. As the computational complexity of SVMs grows at least quadratically with the size of the training data and their dimensionality [23], the data for this algorithm were reduced to a maximum of 40000 samples.

2) Neural Network (NN): The NN was created with 1 hidden layer, thus making it a two-layer NN which is proved to be very effective and is considered a universal function approximator [16]. The activation function used is the rectified linear unit function $f(x) = max(0, x)$. The solver used to optimize the weights is the Limited-memory Broyden–Fletcher–Goldfarb–Shanno (LBFGS) with a regularization parameter of 0.0001, and a constant learning rate of 0.001.

3) Decision Tree (DT), Bagging: To create the DT models, the Gini Impurity was used to measure the quality of the split at each node, and the best one was always chosen. The algorithm used was the Classification and Regression Trees (CART) which is similar to C4.5, however, instead of the entropy-based approach it uses the *Gini index*, a generalization of the binomial variance [24]. As for Bagging, 10 DTs were used as base learners. The DT algorithm was chosen due to their general ability to create good classifiers across datasets.

D. Training, Testing, and Evaluation

Although the datasets are large, the amount of failure data is considerably small, so the recommended approach to evaluating the performance of an algorithm is *Cross-Validation* [12]. *Stratification* was also used to keep the representation of the classes from the original dataset in the resulting folds. The number of folds used was 10, which was based on previous studies where it has been proved that it produces good results across multiple models [25]. Each algorithm was run 30 times with different seeds for the random generators, to allow the reproducibility of the experiments and to minimize the possible bias from random values.

Although there are several performance metrics for classification problems, the main tool used to assess the performance was the Confusion Matrix (CM). It allows the clear identification of the number of samples of each class that is being confused with others, something that is not as easily understandable from other metrics (e.g. accuracy and Area Under the Curve (AUC)).

IV. Results

To analyze the performance of the different algorithms, they were run with all the combinations of the three datasets, different pre-processing techniques, and various $\Delta t_l, \Delta t_p$ pairs, as shown in *Table II*. Furthermore, different approaches were considered for each dataset, from binary to multi-class classification. This obviously leads to a large number of combinations that cannot all be presented here. This way, we focus the

TABLE II
EXPERIMENTS CONFIGURATIONS

Parameter	Values
Datasets	Real, vSphere, XEN, All (Merge of all 3)
Targets	Binary, Multi-failure Binary, Multi-Class
$\Delta t_l, \Delta t_p$	[20, 20], [40, 20], [60, 20]
Feature Selection	Variance, Correlation, RFE
Algorithms	SVM, DT, Random Forest (RF), Bagging, Adaboost, Extra Trees, NN, Naive Bayes, Gaussian Process, Logistic Regression, k Nearest Neighbors (k-NN)

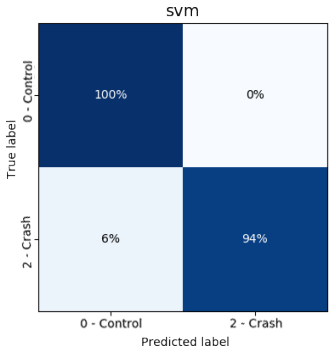

Fig. 2. Crash: SVM, Correlation

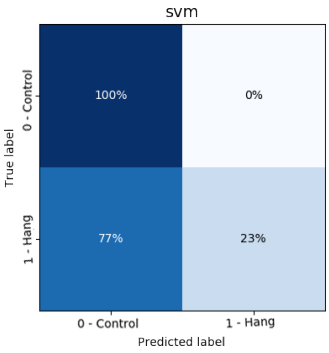

Fig. 3. Hang: SVM

discussion on the deemed relevant for the purpose of this paper. Except where stated otherwise, the results refer to the *Real* dataset, that excludes any interference from the virtualization environment, although it has been shown that virtualization should not alter the behavior of the system variables [20]. Also, due to the space constraints, the values of $\Delta t_l, \Delta t_p$ considered are [40, 20], as this was the configuration that systematically achieved best results. Despite being able to provide similar results in some configurations, the remaining two combinations, [20, 20] and [60, 20], were not able to provide equivalent results in others. In fact, the pair [60, 20] was the one that produced the worst results, most likely due to the fact that its prediction is too distant from the failure. Nonetheless, no thorough analysis has been made yet regarding the effects and possible causes for such differences. Due to the recurrent use of SVMs in previous OFP studies, their performance will be used as an initial reference for comparison with the results of the other algorithms.

A. Prediction of Individual Failure Modes

Most of the existing literature on failure prediction is focused on binary classification, that is, predicting if a specific failure mode will or will not occur. Some, although considering more than one mode, still try to predict them individually [26]. As such, the first experiments attempted to classify *Hang* and *Crash* samples separately.

When predicting crash failures, Support Vector Machine (SVM) models are able to achieve very good results, correctly classifying almost all the *Control* samples (i.e. those for which there is no failure in the interval $[t_{sample} + \Delta t_l, t_{sample} + \Delta t_l + \Delta t_p]$), and 89% of the *Crash* samples (i.e. those for which there is a crash failure in the interval $[t_{sample} + \Delta t_l, t_{sample} + \Delta t_l + \Delta t_p]$). By removing features using correlation, the correct *Crash* predictions improved to 94%, as illustrated in *Figure 2*.

When used to predict hang failures, SVM models are barely able to distinguish between the samples of the two classes, correctly classifying only 21% of *Hang* samples, as depicted in *Figure 3*. In this case, feature selection techniques were not able to significantly improve the results either. As it is well known that imbalanced data can influence the performance of the algorithms [15], we used Random Undersampling to study what was preventing the algorithms from performing better. Although the results are not nearly as good as they were when predicting crash failures, there was a significant improvement, correctly classifying 87% of *Control* samples and 83% of *Hang*. Here we can consider the number of correctly predicted failures as acceptable, although the number of false positives (controls predicted as failures) is significant, reaching 13% of the total number of *Control* samples, as depicted in *Figure 4*.

Using Random Undersampling, most algorithms were able to increase their ability to classify failure samples. However, most of the times this came at the cost of losing the ability to classify *Control* samples, significantly increasing the number of false positives. When using imbalanced data some algorithms, such as SVMs, may have suboptimal performance, producing biased results towards the majority class [27]. Undersampling the data addresses this issue by altering the classes distribution, which will result in the algorithms giving equal weights to both classes.

Other algorithms also performed well for predicting both failure modes, even without pre-processing techniques. The Decision Tree (DT) algorithm achieved very good results in predicting both hang and crash failures. When attempting to predict crash failures, it was able to correctly predict 93% of the failure samples. However, the major difference was when attempting to predict hang failures, where it managed to correctly predict 81% of the *Hang* samples, while correctly classifying almost all *Control* samples. By using undersampling and feature selection, the DT algorithm was able to predict almost all of the *Hang* samples, at the expense of a larger number of false positives, approximately 5%.

Although the performance values vary slightly across the datasets, the conclusions are similar amongst them. The main exception was when predicting crash failures in the vSphere environment, where SVM had very poor performance. This can be due to several reasons, such as the variables not being as indicative with the chosen pair $\Delta t_l, \Delta t_p$ on that environment, or the symptoms may not be as easily distinguishable, thus the SVM algorithm is not able to create an adequate decision function that can separate the samples from both classes.

B. Prediction Undistinguished Failure Modes

Although predicting each failure mode individually is relevant, it is an oversimplification of the problem, as in pro-

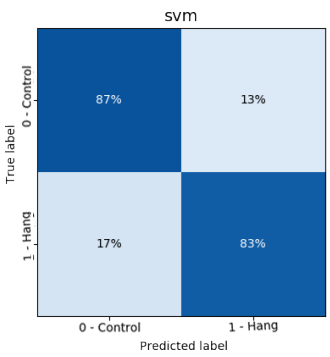

Fig. 4. Hang: SVM, Correlation, Undersampling

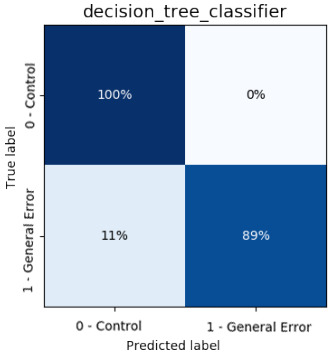

Fig. 5. Combined: DT

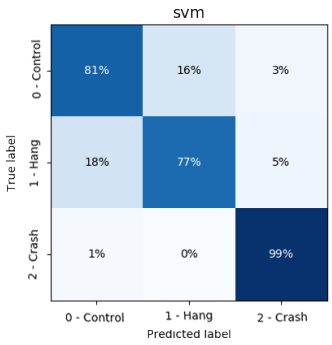

Fig. 6. Multi-Class: SVM, Random Undersampling

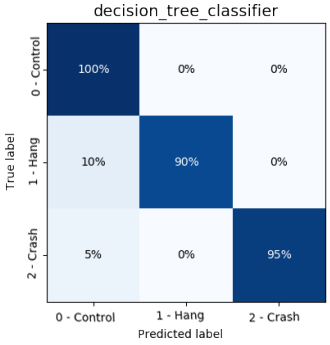

Fig. 7. Multi-Class: DT

duction systems multiple failure modes will exist. As such, in order to analyze if it would be possible to create a model that is able to predict failures regardless of their type, the samples of both classes were grouped into a single one. The problem is still binary but attempts to predict failures of both types.

In this context, SVMs are no longer able to perform satisfactorily. Considering the same conditions previously used, SVM models have poor performance, with 78% of the failure samples being misclassified as *Control*. Using feature selection techniques slightly improved the results (reducing by 3% the misclassification of failures), yet when combining them with undersampling, they improved considerably, correctly classifying 86% of *Control* and 85% of *Hang* samples.

These results fall far behind what is achieved with DTs. DTs are able to accurately classify the failure samples, with performances similar to those observed in the previous section, correctly classifying almost all the *Control* samples, and 83% of the failure samples, as can be seen in *Figure 5*. Feature selection techniques were not able to improve the results.

C. Multi-Class Prediction

After concluding that it is possible to create models that are able to predict a failure regardless of its nature, it became relevant to study if it would be possible to predict differentiated failure modes with a single model. Such a model would provide valuable information, allowing different precaution measures, as well as thorough analyses of which factors and features influence each failure mode.

In this context, SVMs were still unable to perform acceptably, with most of the *Crash* and *Hang* samples, 71%, and 79% respectively, being misclassified as *Control*. Using undersampling, the results improved, although not enough, as can be seen in *Figure 6*. Albeit these results vary slightly across the datasets, they are very similar amongst them.

DTs were able to separate between the various classes with acceptable performance, correctly classifying almost all the *Control* samples, 81% of *Hang*, and 93% of the *Crash* samples, as depicted in *Figure 7*. Once again the results vary slightly across the datasets, yet they are similar amongst them.

By analyzing the performance of the remaining algorithms, it was possible to observe that some also had interesting results, although not as good as DTs. NNs managed to correctly predict almost all *Control* samples, as well as acceptably distinguish between the two failure modes, correctly predicting 60% and 90% of the *Hang* and *Crash* samples respectively.

Another algorithm with very good results was Bagging that, when used together with feature selection by correlation, was able to achieve results similar to those of DTs, correctly classifying 79% of the *Hang* and 92% of the *Crash* samples. Other algorithms, such as Extra Trees and RF, also performed well, albeit not as good as the previously described (e.g. Extra Trees with feature selection by correlation were able to correctly predict almost all *Control* samples, 71% of *Hang* samples, and 95% of *Crash* samples).

To have an insight into how much it was possible to generalize the results, the algorithms that previously performed better were applied to a dataset combining the data from the three different environments (Real, vSphere, and XEN). Although some performance was lost, DTs were still able to very accurately classify the *Hang* (80%) and *Crash* (86%) samples. Ensemble algorithms, such as Bagging, achieved similar results, correctly classifying 77% of the *Hang* samples, as well as 86% of the *Crash* samples.

V. Discussion

Different algorithms were able to achieve good performance, although some performed better across the different contexts studied. Different ML techniques, such as feature selection and undersampling, were also able to improve the performance of the models in different scenarios. In many configurations feature selection techniques, both correlation and RFE alike, were able to improve the performance of the algorithms. By undersampling the data it was also possible to improve the performance of some of the algorithms. As the dataset was highly unbalanced, it was not surprising that evening the distributions in the training phase allowed a better performance when predicting the failure samples. Another advantage of undersampling is that it considerably reduces the number of samples used for training, thus leading to a faster training process. However, in most of the experiments, it also led to a lower performance regarding the *Control* samples. Furthermore, strongly undersampling the data can

be unsafe, as relevant data may be lost, eventually leading the model to overfit. Nevertheless, undersampling is a well-known technique, with considerable research, although in this work only one of the most basic implementations was used.

From the results of the different algorithms in the different datasets and contexts, predicting hang failures seems more complicated than predicting crash failures. This may be due to the fact that the symptoms in the defined $\Delta t_l, \Delta t_p$ are not as significant as they are for crash failures, thus limiting the distinction between the *Hang* and *Control*. Still, when considering the multi-class case, very few samples of *Hang* and *Crash* were misclassified, suggesting that their symptoms are indeed distinguishable.

The performance of the different algorithms varied across the different datasets, implying that the behavior of the failures is somewhat different amongst them. Still, when gathering all the data in a single dataset, the results suggest that it is still possible to create good distinctions between the different classes. This indicates that the behavior of the different failure modes in the various environments is different from each other, yet it is similar across the datasets.

Although predicting each failure mode individually produced good results, combining them, and predicting multiple classes in a single model also achieved similar performance. Furthermore, the latter approach provides much more information regarding the state of the system, without the need to have a different model for each failure mode.

SVMs were able to perform well when considering only one failure mode at a time, as reported in previous works (e.g. [26]). Yet, when faced with data from multiple failure modes, their performance fell considerably and were only able to recover using different pre-processing techniques. SVMs are also limited by the number of samples that can be used, as the process can easily become too computationally expensive. Other algorithms were able to outperform the SVMs for this problem: the ones that performed the best were all based on the DT algorithm (e.g Bagging, RF), with DTs actually being the best for most configurations. An advantage of the DT algorithm is that it is fairly easy to understand the decision function of the models, which allows a thorough study of the features of a particular failure mode.

As there appears to be different algorithms/techniques that excel in some parts of the problem, it would be interesting to develop an ensemble of such algorithms and combine their decision, to overcome their individual limitations.

A key aspect to stress out is related to the metrics used for the analysis. We based our work on the Confusion Matrix (CM) as it provides a visual interpretation of the actual predictions, regardless of class imbalance. Other performance metrics can easily lead to biased conclusions if taken individually. As an example, consider the results presented in *Table III* and *Table IV*, both models created with the SVM algorithm, the latter using Random Undersampling and feature selection by correlation. It is clear that a classifier such as the one in *Table III* is practically useless, as it classifies almost all samples as negatives, while the other reaches an acceptable

TABLE III
HANG: SVM CM

True \ Pred.	Control	Hang
Control	100%	0%
Hang	79%	21%

TABLE IV
HANG: SVM, CORRELATION, UNDERSAMPLING CM

True \ Pred.	Control	Hang
Control	87%	13%
Hang	17%	83%

TABLE V
PERFORMANCE METRICS

Configuration	Precision	Recall	F1-Score	ACC	AUC
SVM	58%	21%	31%	98%	0,95
SVM, Corr., Under.	10%	83%	18%	87%	0,93

rate of correct classifications (at the expense of having more *False Positives*). Yet, by analyzing the metrics in *Table V* we can see that, although both have poor results, the model of *Table III* appears to be better, with higher precision, F1-Score, accuracy, and AUC. This behavior, albeit predictable due to the strong imbalance of the dataset, is often neglected in many works. Furthermore, although AUC is a measure commonly used in many works on OFP, it is not sensitive to differences in the number of *False Positives*. Thus, a considerable change in the number of false positives can lead to small changes in the *False Positive Rate* ($FP/(TN+FP)$) used in ROC-AUC analysis [28] (as can also be observed in *Table V*). Hence, the choice of which metrics to use should be carefully made.

VI. THREATS TO VALIDITY

The work presented intends to be an exploratory study of ML techniques for support of OFP. As such, although it fulfills the purpose, there are threats that should be highlighted.

First and foremost, the OS from where the data used in the experiments was collected, Windows XP (SP3), has long been deprecated. As such, the conclusions on failure prediction of such systems are no longer relevant, as they will most likely differ from recent ones. Nonetheless, the work presented in this article does not intend to provide advances in the study of failures in a specific OS. Its purpose is to demonstrate that the field of ML is wide and that there are multiple algorithms and techniques that can improve the performance of OFP.

This work does not intend to be an extensive overview of all the ML techniques and algorithms. Instead, it attempts to be a demonstration of how an ML approach to the data, from descriptive to exploratory analysis, including the use of pre-processing techniques and different algorithms, can lead to better models. The same applies to the details of each algorithm, as some of the parameters used were based on existing literature and *ad-hoc* optimizations. Still, the parameters chosen give a fair representation of the behavior and performance of each algorithm. A similar argument can be made for the choice of the $\Delta t_l, \Delta t_p$ values, which is not intended to be a thorough study of their influence, but a representation how they affect the failure prediction abilities.

Most of these compromises were done due to the fact that tasks such as choosing the optimal algorithm's parameters, the

$\Delta t_l, \Delta t_p$ values, and the best data pre-processing techniques are in fact complex optimization problems.

VII. Conclusion

The purpose of this article was to demonstrate that there are multiple ML techniques and algorithms besides the ones commonly used for OFP. Although ML has already been used multiple times for similar problems, to the best of our knowledge a broader analysis such as the one provided had not yet been done, and most of the times the algorithms are chosen based on what was used in previous similar works, with no clear justification. As it was demonstrated, different pre-processing techniques can influence the performance of the model, as well as distinct algorithms, can perform differently.

Although the article does not intend to draw conclusions on the specific nature of failures in the considered environments, it was shown that ML techniques and algorithms have the potential to create accurate prediction models. Based on this premise, considering new datasets in a recent OS in combination with different ML techniques will most likely allow creating accurate failure prediction models. It was also observed that, despite different workloads, different injection sites, and even different environments, some algorithms are able to generalize and still provide accurate models. Also, models that can predict multiple failure modes are possible, replacing the typical binary classifications, thus providing further information regarding the state of the system.

As future work, we intend to conduct a thorough study of these and other ML methods with a dataset collected from a more recent OS. An extensive research of the inner-works of the different techniques will then be conducted, attempting to better understand the nature of the OFP problem, as well as the reasons behind the different performances. Ultimately, the goal is to create models that are able to accurately predict system failures and that can be used in real systems.

Acknowledgment

This work has been partially supported by CISUC grant no DPA 18-034 and project ATMOSPHERE, funded by the European Commission under the Cooperation Programme, Horizon 2020 grant agreement no 777154. We would also like to thank Ivano Irrera for the support and the datasets provided.

References

[1] B. Dhanalaxmi, G. Apparao Naidu, and K. Anuradha, "A Review on Software Fault Detection and Prevention Mechanism in Software Development Activities," *IOSR Journal of Computer Engineering Ver. V*, vol. 17, no. 6, pp. 2278–661, 2015.

[2] L. C. Algirdas Avižienis, Laprie Jean-Claude, Randell Brian, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[3] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," *ACM Computing Surveys*, vol. 42, no. 3, pp. 1–42, 2010.

[4] F. Salfner and M. Malek, "Proactive fault handling for system availability enhancement," *Proceedings - 19th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2005*, vol. 2005, 2005.

[5] J. A. Durães and H. S. Madeira, "Emulation of software faults: A field data study and a practical approach," *IEEE Transactions on Software Engineering*, vol. 32, no. 11, pp. 849–867, 2006.

[6] I. Irrera and M. Vieira, "Towards Assessing Representativeness of Fault Injection-Generated Failure Data for Online Failure Prediction," *Proceedings - 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN-W 2015*, pp. 75–80, 2015.

[7] I. Irrera, M. Vieira, and J. Durães, "Adaptive Failure Prediction for Computer Systems: A Framework and a Case Study," *2015 IEEE 16th International Symposium on High Assurance Systems Engineering*, pp. 142–149, 2015.

[8] J. Murray, "Hard drive failure prediction using non-parametric statistical methods," *Proceedings of ICANN*, no. 1, 2003.

[9] F. Salfner and M. Malek, "Using hidden semi-Markov models for effective online failure prediction," *Proceedings of the IEEE Symposium on Reliable Distributed Systems*, pp. 161–174, 2007.

[10] I. Irrera, C. Pereira, and M. Vieira, "The time dimension in predicting failures: A case study," *Proceedings - 6th Latin-American Symposium on Dependable Computing, LADC 2013*, pp. 86–91, 2013.

[11] I. Irrera, J. Duraes, and M. Vieira, "On the Need for Training Failure Prediction Algorithms in Evolving Software Systems," *2014 IEEE 15th International Symposium on High-Assurance Systems Engineering*, pp. 216–223, 2014.

[12] E. Alpaydin, *Introduction to Machine Learning*, 3rd ed., ser. Adaptive Computation and Machine Learning. Cambridge, MA: MIT Press, 2014.

[13] J. H. Friedman, "Data mining and statistics: What's the connection," in *Proceedings of the 29th Symposium on the Interface Between Computer Science and Statistics*, 1997.

[14] A. Ghosh, *Evolutionary Computation in Data Mining*, ser. Studies in Fuzziness and Soft Computing, A. Ghosh and L. C. Jain, Eds. Berlin/Heidelberg: Springer-Verlag, 2005, vol. 163, no. November.

[15] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[16] T. Hastie, R. Tibshirani, and J. Friedman, "The Elements of Statistical Learning," *Bayesian Forecasting and Dynamic Models*, vol. 1, pp. 1–694, 2009.

[17] J. P. Marques de Sá, *Pattern Recognition*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001.

[18] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing and Management*, vol. 45, no. 4, pp. 427–437, 2009.

[19] I. Irrera, J. Durães, M. Vieira, and H. Madeira, "Towards identifying the best variables for failure prediction using injection of realistic software faults," *Proceedings - 16th IEEE Pacific Rim International Symposium on Dependable Computing, PRDC 2010*, pp. 3–10, 2010.

[20] I. Irrera, J. Durães, H. Madeira, and M. Vieira, "Assessing the impact of virtualization on the generation of failure prediction data," *Proceedings - 6th Latin-American Symposium on Dependable Computing, LADC 2013*, pp. 92–97, 2013.

[21] B. Alexander Yun-chung Liu, "The Effect of Oversampling and Undersampling on Classifying Imbalanced Text Datasets," *The University of Texas at Austin*, no. August, pp. 1–52, 2004.

[22] G. Anthony, H. Gregg, and M. Tshilidzi, "Image Classification Using SVMs: One-against-One Vs One-against-All," *The 28th Asian Conference on Remote Sensing*, pp. 801–806, 2007.

[23] A. Bordes, e. Ertekin, J. Weston, and L. Bottou, "Fast Kernel Classifiers with Online and Active Learning," *Journal of Machine Learning Research*, vol. 6, pp. 1579–1619, 2005.

[24] W.-Y. Loh, "Classification and regression trees," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 14–23, 2011.

[25] S. Borra and A. Di Ciaccio, "Measuring the prediction error. A comparison of cross-validation, bootstrap and covariance penalty methods," *Computational Statistics and Data Analysis*, vol. 54, no. 12, pp. 2976–2989, 2010.

[26] I. Irrera and M. Vieira, "A Practical Approach for Generating Failure Data for Assessing and Comparing Failure Prediction Algorithms," *2014 IEEE 20th Pacific Rim International Symposium on Dependable Computing*, pp. 86–95, 2014.

[27] H. He and Y. Ma, *Imbalanced Learning: Foundations, Algorithms, and Applications*, 1st ed. Wiley-IEEE Press, 2013.

[28] J. Davis and M. Goadrich, "The relationship between Precision-Recall and ROC curves," *Proceedings of the 23rd international conference on Machine learning - ICML '06*, pp. 233–240, 2006.