IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 21, 2019, accepted November 15, 2019, date of publication December 9, 2019,
date of current version December 23, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2958480

Improving Failure Prediction by Ensembling the Decisions of Machine Learning Models: A Case Study

JOÃO R. CAMPOS, ERNESTO COSTA, AND MARCO VIEIRA
DEI/CISUC, University of Coimbra, 3030-290 Coimbra, Portugal

Corresponding author: João R. Campos (jrcampos@dei.uc.pt)

This work was supported in part by the Portuguese Foundation for Science and Technology (FCT) under Grant SFRH/BD/140221/2018, in part by the MobiWise project: *From mobile sensing to mobility advising* under Grant P2020 SAICTPAC/0011/2015, in part by the COMPETE 2020, in part by the Portugal 2020-Operational Program for Competitiveness and Internationalization (POCI), in part by the European Union's ERDF (European Regional Development Fund), and in part by the Portuguese Foundation for Science and Technology (FCT).

ABSTRACT The complexity of software has grown considerably in recent years, making it nearly impossible to detect all faults before pushing to production. Such faults can ultimately lead to failures at runtime. Recent works have shown that using Machine Learning (ML) algorithms it is possible to create models that can accurately predict such failures. At the same time, methods that combine several independent learners (i.e., ensembles) have proved to outperform individual models in various problems. While some well-known ensemble algorithms (e.g Bagging) use the same base learners (i.e., homogeneous), using different algorithms (i.e., heterogeneous) may exploit the different biases of each algorithm. However, this is not a trivial task, as it requires finding and choosing the most adequate base learners and methods to combine their outputs. This paper presents a case study on using several ML techniques to create heterogeneous ensembles for Online Failure Prediction (OFP). More precisely, it attempts to assess the viability of combining different learners to improve performance and to understand how different combination techniques influence the results. The paper also explores whether the interactions between learners can be studied and leveraged. The results suggest that the combination of certain learners and techniques, not necessarily individually the best, can improve the overall ability to predict failures. Additionally, studying the synergies in the best ensembles provides interesting insights into why some are able to perform better.

INDEX TERMS Ensembles, failure prediction, machine learning, reliability.

I. INTRODUCTION

Given the growing complexity of software in recent years, it is nearly impossible to detect every fault before deployment. Such (residual) faults can eventually escalate to failures, compromising the system's availability, reliability, and supported business processes. In fact, software faults have been recurrently identified as a major cause of system outages [1].

Depending on the purpose of the system, failures may incur considerable risk or cost, either due to the recovery mechanisms or to the resulting system interruption/corruption. For example, a 63-minute Amazon outage in 2018 reportedly cost the company around $100 million [2]. In a more critical scenario, in 2016, a GPS satellite failure compromised worldwide GPS systems for several hours [3].

Given the potential consequences associated with failures, over the years several techniques have been developed to avoid or tolerate faults (e.g., coding practices, testing) [4]. Online Failure Prediction (OFP) is a complementary technique that intends to mitigate the potential effects of residual faults. It correlates past failure data with the current system state to predict the occurrence of failures in the near future [5]. Such predictions can then be used to take preemptive measures, such as saving data or restarting parts of a system to mitigate their consequences. Notwithstanding, despite its great potential, OFP is still not widely implemented.

Due to the technological developments in recent years, Machine Learning (ML) algorithms have been successfully used in a variety of complex problems. Such algorithms can

The associate editor coordinating the review of this manuscript and approving it for publication was Ah Hwee Tan.

find (complex) patterns in the data and learn from them without relying on a predetermined model. Afterward, they can be used to make predictions on new unseen data based on what was previously learned. Although there are several popular classification algorithms (e.g., Support Vector Machine (SVM)), *Ensemble Methods* have also recently gained relevance. They consist of a compilation of independent algorithms whose predictions are gathered to work as a whole. It is believed that a good ensemble is made of base learners that are as accurate and diverse as possible [6]. Although some of the most well-known ensemble methods use only a single base learner (also known as *homogeneous*, e.g., Random Forest (RF)) it is also possible to combine different learning algorithms (also known as *heterogeneous*) [7], often leading to better results [8]. Various approaches relying on ML for OFP have already been proposed [5], and the use of homogeneous ensembles has already shown promising results in our previous work [9]. However, for OFP, the combination of different learners has not yet been explored.

This paper presents an exploratory case study on using different ML algorithms and techniques to create heterogeneous ensembles for supporting OFP. To this end, failure data are used to assist in the development of a data-driven study on how combining different learners can improve the performance of individual models. These data were generated using fault injection and considering different software faults. They contain the data of 233 system variables monitored during execution at the Operating System (OS) level and consider two failure modes: *System Hang* and *Crash*. Multiple algorithms and preprocessing techniques were considered for both the base learners and ensemble combination methods. Finally, the best ensembles are explored in detail and a preliminary analysis is conducted to understand why and how their constituent models complement each other. For these experiments, we used the Propheticus framework [10], which allows us to easily explore the problem/data and effortlessly run and combine several ML and techniques.[1]

More precisely, this paper attempts to answer the following research questions concerning OFP:

- *RQ1*: does combining different learners improve the overall performance?
- *RQ2*: do different combination techniques influence the results and the (best) ensemble composition?
- *RQ3*: can the interactions between the learners in an ensemble help to understand the performance obtained?

Results suggest that for OFP it is indeed possible to improve the performance of individual learners (even if homogeneous ensembles themselves) by combining models created with different algorithms and techniques (thus, affirmatively answering *RQ1*). Additionally, it was also observed that the structure of the ensembles, their performance, and their interpretability are influenced by the combination techniques used (thus, affirmatively answering *RQ2* and *RQ3*).

[1]Due to space restrictions it will not possible to show every result, which can be found at `http://www.joaorcampos.com/Access-2019`.

This study intends to assist researchers and systems administrators who want to find the best model that fits their system. It provides insights into how to ensemble individual models to achieve better prediction performance, by exploring several ML algorithms, techniques, and ensembling methods.

This paper is organized as follows. *Section II* provides background concepts and discusses related work. *Section III* overviews the methodology used. *Section IV* describes the process of selecting the base learners, while *Section V* focuses on combining their predictions, and *Section VI* analyzes the best ensembles. *Section VII* discusses the results obtained and *Section VIII* addresses the threats to validity. Finally, *Section IX* concludes and puts forward ideas for future work.

II. BACKGROUND AND RELATED WORK

This section provides some concepts and related work on Online Failure Prediction (OFP) and Machine Learning (ML).

A. THE FAILURE PREDICTION CONCEPT

OFP allows foreseeing incoming failures at runtime, enabling the mitigation of their effects [11]. It is particularly useful in the context of residual faults (i.e., faults that escaped the testing process) that cannot be tolerated by other fault-tolerance mechanisms [4]. Several different types of OFP techniques were proposed so far, and an exhaustive survey can be found in [5]. Among them, a method based on clustering the system state and the use of Hidden Semi-Markov Models to predict failure-prone system states was proposed [12], and SVMs were used to predict failures of hard drives [13]. More recently, we conducted an exploratory study on using different ML algorithms and techniques for OFP [9]. However, while relevant, this work focused on studying the performance of various ML algorithms and techniques, without configuring and tuning the models thoroughly.

The prediction of failures is based on different kinds of information, including the past data from the system (used to train the predictor), the current information about the state of the system (obtained by monitoring some system variables), the time horizon of the prediction, among others. More precisely, a prediction performed at time t targets a window starting at time $t + \Delta t_l$, and lasting for Δt_p (Δt_l and Δt_p are normally referred to as *Lead Time* and *Prediction Window*, respectively) [5]. In practice, at time t, a model should predict if a failure is going to occur in the interval $[t + \Delta t_l, t + \Delta t_l + \Delta t_p]$.

B. MACHINE LEARNING FOR OFP

ML algorithms have shown their ability to adapt and extract knowledge in a variety of complex problems, including for failure prediction. Such 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. Concerning terminology, 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 or not occur for that sample is known as the *class*. As each instance in the dataset contains the information on whether a failure is going to be observed within a given time, it is considered a *Supervised Learning* problem. Thus, every supervised algorithm can theoretically be used.

One of the uses of OFP is to predict how failure-prone the system is, which is a *Regression* problem. Regression problems try to predict continuous values, and the performance of those models is often measured with metrics such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE). However, more commonly, OFP can also be used to predict whether a failure will or not occur [5] (**the focus of this work**). This is a *Classification* problem that is concerned with separating data into distinct classes that are discrete and categorical. The performance of these classification models is often measured through metrics such as recall and precision. As the complexity of any model depends on the number of inputs, reducing dimensionality (e.g., *Feature Selection/Extraction*) may be important. Additionally, to help algorithms to cope with the infeasibility of very large datasets, *Instance Selection* techniques (e.g., *sampling*, *boosting* [14]) can also be used. Moreover, imbalanced datasets may compromise the performance of the algorithms on the minority classes. For this, there are specific solutions such as *Undersampling* and *Oversampling* [15].

To have a realistic estimate of the performance of a model, two main sets of data are normally used: *train* and *test*. The training set is used for training the model, while the testing set estimates the generalization error of the final model [16]. Additionally, a third set (*validation*) can be created from the training set and be used to avoid overfitting. This division is not trivial, as it may inadvertently influence the performance/representativeness of the model. Thus, several techniques have been proposed over the years (e.g., *Partition/Leave-one-out*, *Bootstrap Methods* [17]). Finally, despite all the factors that can be considered, one should not forget that whatever conclusions are drawn, these are always conditioned by the dataset with which the model was created.

C. ENSEMBLE METHODS

Ensemble methods are a compilation of several independent algorithms whose predictions are gathered to work as a whole, as depicted in *Figure 1* (x represents the input feature vector, and y the ensemble decision/output). They contain a number of learners called *base learners* (or *weak learners*), which are usually generated using a single algorithm such as *Decision Tree (DT)* or *Neural Network (NN)* [7]. Some of the most well-known ensemble methods are based on DT classifiers, such as *Boosting*, *Bagging*, and *RFs*.

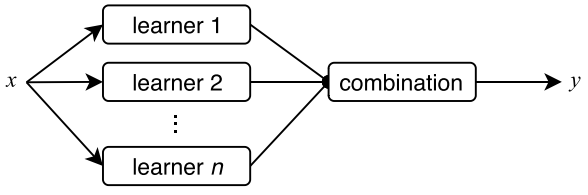

FIGURE 1. Ensemble methods..

Creating an ensemble can be divided into two steps: generating the base learners and combining them. It is believed that a good ensemble is made of base learners as accurate and diverse as possible. Additionally, their outputs should be combined in such a way that correct decisions are amplified and incorrect ones are canceled out [6]. Although demanding accurate classifiers is easily understood (i.e., the combination of wrong predictions cannot easily generate a correct decision) measuring diversity is complex, as there is no consensus or formal definition [18]. Nonetheless, several metrics have already been proposed (e.g., Q Statistic, disagreement measure), although the correlation between diversity and performance is not fully understood [19].

The construction of an ensemble can be separated into two approaches: *Classifier Selection* and *Fusion* [20]. In classifier selection, the classifiers are trained to become experts in some area of the feature space. Then the output of the classifiers identified as the best for a specific classification problem is selected. Usually, the input sample space is partitioned into smaller areas and each classifier learns the example in each area. Classifier fusion, on the other hand, combines the outputs of the different classifiers. Each one has knowledge of the entire feature space and tries to solve the same classification problem using different methods based on different training datasets, classifiers or parameters [20]. Considering the goal of this work, the **experiments will focus solely on classifier fusion**. Some of the most well-known ensembles use a single base learner and are called *homogeneous* ensembles. However, it is also possible to combine different learners into a *heterogeneous* ensemble [7], often leading to better results than using a single base learner [8].

After creating the set of base learners, ensemble methods rely on combination (instead of choosing the best individual model) to achieve a good generalization ability, where the combination method plays a crucial role. Thus, there are several approaches to combine the various results depending on the nature of the predictions (e.g., continuous or categorical) [7]. As the approach used in this paper focuses on classification (i.e., categories), the most applicable techniques are based on voting [7]. *Plurality Voting* is one of the most popular and simple approaches, where each classifier votes for one class label and the final prediction is the class with more votes. *Majority Voting* is very similar, with the difference being that the final prediction is the class which obtains more than half of the votes (if there is no majority a rejection option will be given). Alternatively, *Soft Voting* is a strong option for algorithms that provide class probability outputs. Its simplest approach averages the predictions for each class and selects the one with the highest probability. This allows taking into consideration both the prediction and the confidence of the model in the decision. Finally, *Stacked Generalization* (also known as *Stacking*) uses the concept of a *meta-learner*. Briefly, the base learners are trained using

a training dataset and then a different dataset is used for training the meta-learner through cross-validation (where the outputs of the base learners are the input features for the meta-learner). This meta-learner attempts to model the outputs of the base classifiers (either the class probabilities or the predictions) to generate the final output.

In recent years, ensemble methods have been recurrently used in various software related domains [21]–[27]. For example, in [21] an ensemble-based method was used to predict the effort of agile software development projects, outperforming existing approaches. Also, in [22] several ensemble methods were combined with Synthetic Minority Oversampling Technique (SMOTE) oversampling to predict defects in software, effectively enhancing the defect prediction accuracy. In [23] the authors combined SVMs and ensemble techniques to detect vulnerable software components, achieving promising results compared to the state-of-the-art. Finally, in some research areas (e.g., Intrusion Detection Systems) the use of ensemble methods has been so profuse that even lengthy surveys are made on the subject [24]. However, no research has been found on using ensemble techniques for OFP. Nonetheless, a conclusion that can be drawn from the related works above, is that their proposed ensemble approaches outperform the alternative state-of-the-art individual methods.

III. EXPERIMENTAL APPROACH

This section describes the experimental process and the most relevant choices/components concerning the experiments.

A. PROCESS

OFP is a key technique for proactive fault management. As briefly introduced in *Section II*, 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]. In practice, it can be defined as a decision process that at a time t tries to predict whether a failure is going to occur within a precise time, called lead-time Δt_l. These predictions are valid for a given time window, called prediction-window Δt_p. In short, at time t, the prediction model should predict if a failure is going to happen in the interval $[t+\Delta t_l, t+\Delta t_l+\Delta t_p]$ [5], as depicted in *Fig. 2*.

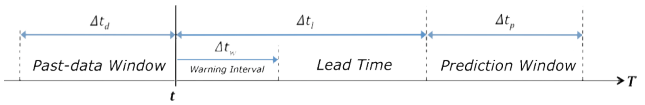

FIGURE 2. Time relations in Online Failure Prediction (OFP), adapted from [5].

A high-level overview of the experimental process is depicted in *Fig. 3*. Briefly, after training and assessing the performance of the various classifiers, the best set is chosen to a pool of candidate learners. Then, a selection process chooses the base learners, which are then trained and tested and their outputs combined using a combination method.

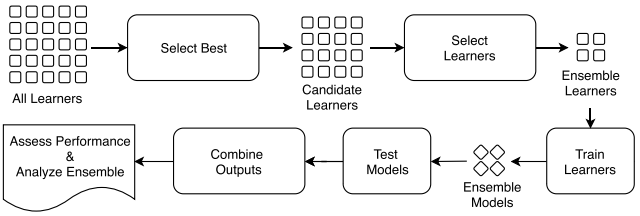

FIGURE 3. Experimental process.

Finally, the performance of the ensemble is assessed and its structure is analyzed.

B. DATASETS

The data used were generated and made available by a previous work to investigate the use of fault injection on Windows XP (SP3) to generate failure-related data [28]. We are aware that Windows XP is an old OS, but the dataset is nevertheless valid and is the most complete publicly available, thus providing an adequate context for this study.

The data has been generated considering two different workloads and the software faults were injected into the OS by a tool implementing the Generic Software Fault Injection Technique (G-SWFIT) technique [29]. According to the authors, different faults were injected in specific locations previously selected. To generate the data several executions (*runs*) of the workloads were performed (with and without fault injection). A run in which faults are injected is a *Fault Injection Run*. If a failure occurs during such a run, it is considered a *failing run*; if not, it is a *non-failing run*. Additionally, *Golden Runs* represent the behavior of the system in the absence of fault injection [29]. The failure modes considered are *System Crash* (OS becomes corrupted and crashes or reboots) and *System Hang* (OS becomes unresponsive and must be terminated by force). The dataset contains 233 numeric variables, chosen by the authors as the best representative set of the system behavior.

Although it would be possible to study ensembling techniques for each failure mode separately, due to the exploratory nature of this work it is focused solely on predicting both failure modes (i.e., a multiclass problem). From a practical perspective, having various models for each failure mode raises several issues (e.g., combining conflicting predictions), and thus the possibility of having a single model that is able to predict multiple failure modes is more compelling.

Concerning the pair Δt_l, Δt_p this work focuses only on the configuration [40, 20], as it was the one that systematically achieved the best results in our previous work (among those studied, [20, 20], [40, 20], and [60, 20], for a "short", "medium", and "long" term prediction) [9]. This configuration was used to label the data according to the approach proposed in [5]. 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.

Although the data includes several environments (e.g., different hypervisors), as the focus of this work is not

on virtualization techniques, we consider only the bare-metal instantiation (removing any interference from the virtualization environment, although it should not alter the behavior of the system variables [29]). Additionally, despite faults are injected in the system, not all of them lead to the observation of failures. Hence, due to the fact that the dataset contains a very large number of runs without failures (e.g., 5112 compared to 89 failing runs for the bare-metal setup), the data used were limited to a maximum of 50000 samples. This is large enough to comprise all the failure runs, several *golden runs*, and numerous runs where faults were injected but no failures were observed. Thus, the class distribution was 48937 *Control*, 832 *Hang*, and 145 *Crash* samples.

C. ML ALGORITHMS AND TECHNIQUES

As the purpose of this work is to explore the combination of different learners (heterogeneous ensembles), various ML algorithms and techniques were considered, which can be seen in *Table 1*.

TABLE 1. Techniques used in the experiments.

Process / Step	Techniques
Feature Selection	Variance, Correlation
Sampling	Random under/oversampling, SMOTE, Instance Hardness Threshold
Algorithms	DT, Bagging RF, GB
Ens. Combination	Plurality and Soft Voting, Stacking (DT, RF, Bagging, Log. Regr., NN)

Also, because the focus is the ensembles some of the techniques used were selected based on the preliminary results found in our previous work [9]. Additionally, GB, an algorithm that has recently shown promising results, was included in the experiments. Although most of the algorithms are already ensembles, in this study their output is used as if they were a single classifier. Their hyperparameters were tuned according to the values described in *Table 2*.

TABLE 2. Algorithms' hyperparameters.

Alg.	Hyperparameters
DT	**criter.**: [gini, entropy], **min_samp_split**: [.001, .01, .1, 2], **max_feat.**: [.1, .55, 1.0], **min_samp_leaf**: [.001, .01, .1, 1]
RF	**estimators**: [10, 100, 200], **max_feat.**: [.1, .55, 1.0], **criter.**: [gini, entropy], **min_samp_leaf**: [.001, .01, .1, 1], **min_samp_split**: [.001, .01, .1, 2], **bootstrap**: [1, 0]
Bag.	**max_features**: [.1, .55, 1.0], **bootstrap**: [1, 0], **estimators**: [10, 50, 100, 200]
GB	**estimators**: [50, 100, 200], **learning rate**: [1, .1, .01], **min_samp_leaf**: [.001, .01, .1, 1], **max_feat.**: [.1, .55, 1.0], **min_samp_split**: [.001, .01, .1, 2]

The imbalanced nature of the dataset requires considering different sampling techniques. To this end, both under/oversampling techniques were used (see *Table 1*). Concerning the sampling ratios (which were tuned through an ad-hoc approach), undersampling was done to achieve a ratio of 1:1 between the majority and minority classes (i.e., the number of samples for each class will be the same) and the oversampling was done for a ratio of 1:3 (within the minority classes, that is, their representation increased 200%, a ratio that showed promising results for this dataset). Concerning feature selection, only two simple methods were considered: *variance*, removing features with 0 variance (i.e., constants, which do not add any information but still increase the complexity of the model), and *correlation*, removing highly correlated features (>90%) (due to their correlation such features can normally be removed without a significant loss of information).

Although the dataset is large, the amount of failures is considerably small, so the recommended approach to evaluate the performance of an algorithm is cross-validation [14]. *Stratification* was also used to keep the representation of the classes from the original dataset in the resulting folds. The number of folds used is 5, based on previous studies where it has proved to produce good results across multiple experiments [30]. Each algorithm was run 30 times with different seeds for the random generators, to allow the reproducibility of the experiments and minimize any possible bias.

D. BASE LEARNERS SELECTION AND COMBINATION METHODS

Deciding which base learners to consider is not a trivial task. They should be both accurate and diverse. However, measuring diversity is not straightforward, as there is no formal definition and several measures can be found in the literature [19]. Moreover, although it is common to have a correlation between diversity and performance, this is not always the case [19]. Thus, due to the exploratory nature of this work, the best models were manually selected and then an exhaustive search of all the possible combinations was conducted (as further explained in sections *IV* and *V*).

Concerning the combination methods, three approaches were considered: *Plurality Voting* (i.e., the most voted class is chosen); *Soft Voting* (i.e., the class prediction probabilities are averaged and the class with the highest probability is chosen); and *Stacking* (i.e., a meta-learner is used to model the outputs of the individual learners) for both crisp predictions (i.e., labels) and class probabilities outputs. For the Stacking meta-learner, different algorithms were also considered (seen in *Table 1*). Concerning their configurations, due to time constraints, it was not possible to thoroughly explore and tune all the meta-learner hyperparameters, thus, an ad-hoc approach based on the defaults used by Scikit-learn (which in turn are based on literature) was used [31].

IV. BASE MODELS SELECTION

This section describes the process used to choose the best learners and combine them into an ensemble.

A. INDIVIDUAL LEARNERS

Based on the results and conclusions from our previous work [9], we focused only on using ML algorithms based on DT (as shown in *Table 1*), which systematically achieved better results. As the hyperparameters of the algorithms can considerably influence their performance, several values were analyzed to find the best configuration (which can be seen in *Table 2*). Additionally, various sampling techniques were considered to (partially) handle the imbalance in the data. Thus, the first stage of the experimental process was to create individual models, assess their performance, and identify the best solutions.

By analyzing the results it was possible to see that some algorithms were able to achieve very good performances. As can be deduced from the number of algorithms, hyperparameters, and techniques, this led to a significant number of experiments. Due to space restrictions, it is not possible to present all the results, thus, only the most relevant will be discussed (see footnote[1]).

To compare and rank the different models we need to choose a performance metric that allows characterizing their effectiveness. However, although there are several metrics available, they should be carefully used as they are not independent from the data [32]. In fact, while it is common to see a widespread use of metrics such as accuracy and precision, they should be carefully analyzed. Concerning accuracy, it is not an adequate metric for imbalanced data because it does not take into account the (representation) differences between classes. In fact, in a highly imbalanced dataset a model that correctly predicts all negative samples and no positive samples (i.e., just predict everything as negative, no failure will ever occur) would just have a slightly lower accuracy than a model that could also predict all failures (i.e., a ''perfect'' model). Concerning precision, while relevant, alone it is also not adequate, as, for example, a model that predicts everything as negative with the exception of a single correctly predicted failure (i.e., 1 True Positive (TP)) would have a precision of 100%, with a (near) 0% recall. Additionally, the metric should also take into consideration the requirements of the system where the predictor should operate. In fact, the definition of the ''best'' model is not straightforward. As an example, for a more critical context (e.g., home banking), one wants to select a predictor with a higher detection rate, even if it raises more false alarms than others (within some acceptable bounds), since unpredicted failures may have serious consequences. On the other hand, for a more medium-quality context (e.g., corporate site), one may want a predictor with a high detection rate, but that does not raise too many false alarms, since the cost of pro-actively dealing with those false alarms may be high compared with the mitigation of failures.

For OFP, a common/general goal is to predict as many failures as possible but also taking into account the number of False Positives (FPs) (i.e., a system that is constantly giving false alarms is not very useful for many scenarios). Thus, considering this context, for this case study the F_2-score metric was chosen, which gives double the importance of recall when compared to precision (i.e., it is more important to predict failures (recall), yet a compromise must be made with the number of FPs (precision)). To establish a baseline performance an initial study was conducted using the default hyperparameters, which achieved performances similar to those obtained for the same algorithms and techniques in our previous work [9].

Before combining the algorithms with different sampling techniques, we wanted to study and assess their performance using different hyperparameters. By analyzing the results it was possible to observe that tuning the hyperparameters allowed most algorithms to improve their baseline performance. As an example, tuning Bagging with feature selection by correlation (whose baseline performance, also removing highly correlated features, was 99.8%, 75.6%, and 92% of *Control, Hang,* and *Crash* samples, respectively), we were able to achieve better results, correctly classifying 99.9%, 89%, and 95.4% of *Control, Hang*, and *Crash* samples. After tuning the hyperparameters the remaining algorithms had similar performance (except the DT, which was considerably lower, correctly predicting 99.6%, 80.9%, and 92.8% of *Control, Hang*, and *Crash* samples), with GB correctly predicting 99.8%, 88.6%, 95.9% of *Control, Hang,* and *Crash* samples.

A similar study was conducted exploring the different sampling techniques. By analyzing the results, it was possible to observe that the use of different techniques created classifiers that were better at modeling different parts of the problem. For example, using oversampling techniques lead to an overall small improvement of the models' ability to predict failures, while maintaining a similar performance on predicting *Control* samples. More precisely, using SMOTE oversampling, GB was able to correctly predict 99.8%, 92.2%, and 96.2% of *Control, Hang,* and *Crash* samples, respectively, as depicted in *Fig. 4*. On the other hand, the use of undersampling techniques consistently created models that were considerably better at predicting failures, as can be seen

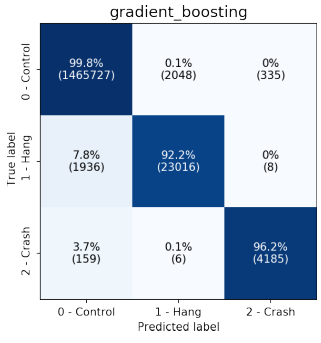

FIGURE 4. GB w/ SMOTE.

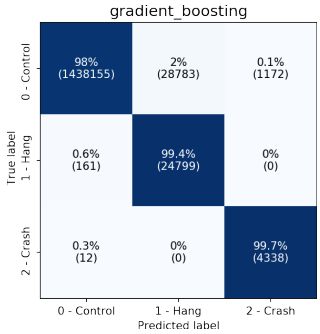

FIGURE 5. GB Rand. Under./Over.

in *Fig. 5*. However, this came at the expense of significantly more FPs. Note that, the numbers presented in the confusion matrices correspond to the total number of samples per class (∼50k total) multiplied by the number of executions/seeds (30). This is due to the fact that each random seed will generate different cross-validation folds (because of the inherent randomness in splitting the data) and thus the trained models (and their predictions) will vary between seeds. Still, this does not affect our analyses and conclusions as we will focus on the percentages of correct/incorrect predictions (and not on the absolute numbers).

Although the DT algorithm was able to achieve an overall good performance, it was considerably lower than the remaining algorithms. Thus, due to space constraints, it will not be considered for the remainder of the article.

B. SELECTING LEARNERS

The main purpose of ensembling models is to improve their individual performances, which often are not very good. A good ensemble should be composed of diverse models and their combination should reduce incorrect decisions and amplify the correct ones [6]. To select a diverse set of base learners there are several metrics available (e.g., Q Statistics). However, an issue that arises in our experiment is that the set of algorithms and techniques considered already yield models with very good (and similar) performances (*Fig. 4*), thus, there is not much room for improvement. This is more blatant when considering the imbalance in the data and the actual number of samples that are correctly predicted.

An obvious approach for choosing the base learners is to combine the best solutions found for each algorithm (i.e., those with the highest performance, i.e the highest F_2-score in our case study). The three models selected this way were: *i)* Gradient Boosting (GB) with oversampling through SMOTE; *ii)* Bagging also with SMOTE oversampling; and *iii)* Random Forest (RF) with Random oversampling.

The problem is that when we analyzed the results (further detailed in *Section V*) we concluded that, although they were individually the best, they did not complement each other, resulting in a lower combined performance. Hence, a different approach was necessary, which is depicted in *Fig. 6*.

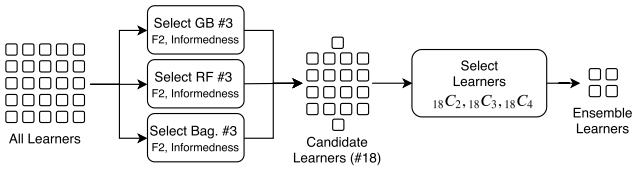

FIGURE 6. Base learners selection.

As we observed before (when finding the best individual learners, *Section IV-A*) that using different techniques created models that were better at predicting different parts of the problem, we found of interest to combine them. However, using only the F_2-score to rank the models could make all those selected to be very similar, thus predicting identical parts of the problem. Hence, they were also chosen taking into consideration Informedness (which measures how consistently a predictor predicts the outcome of both a TP and True Negative (TN)) [33]. More precisely, the top three models of each algorithm for each metric were selected to a pool of potentially interesting (candidate) base learners. However, it is worth mentioning that every model selected through this process also had very good performance (e.g., for F_2-score the best was GB with SMOTE, seen in *Fig. 4*, and for Informedness the best was GB with both Random under- and oversampling, as shown in *Fig. 5*).

This approach lead to a pool of 18 learners. Although there are several metrics that could be used to select the base learners (e.g., Q-statistics, entropy), we opted for an exhaustive search to consider every possible combination. Additionally, as all the selected models were already very accurate, there was no guarantee that the diversity measures would lead to a better ensemble. Exploring all the possible combinations of the 18 models for every possible length is infeasible, thus the ensembles were limited to a maximum of **4** learners. This still leads to a significant number of combinations (i.e., 4029 ensembles for each type of output, crisp labels, and class probabilities).[1]

V. ENSEMBLE COMBINATION

A crucial decision when building an ensemble is how to combine the predictions of the individual models to work as a whole. As stated in *Section III*, three approaches were considered: *i)* Plurality Voting (i.e., the class with most votes is selected); *ii)* Soft Voting (i.e., the class probabilities for each prediction are averaged and the one with the highest probability is returned); and *iii)* Stacking, where a meta-learner is used to combine the results of the various learners. A summary of the best results can be seen in *Table 3*, where the results obtained by the overall best ensemble are highlighted. A brief note, due to the considerably higher time complexity of Stacking the set of ensembles considered for this approach (which will be described further in *Section V-C*) we did not include the combination of best individual models. Thus, for Stacking, only the best meta-learner is shown.

TABLE 3. Ensemble combination results summary.

Comb.		Class: 0 - Control	1 - Hang	2 - Crash
Plur.	*Indiv.*	99.8%	91.4%	96%
	Best	99.8%	97.1%	98.3%
Soft	*Indiv.*	99.9%	93.6%	97.5%
	Best	**99.8%**	**97.5%**	**98.6%**
Stk.	*Best*	99.8%	97.6%	96.8%

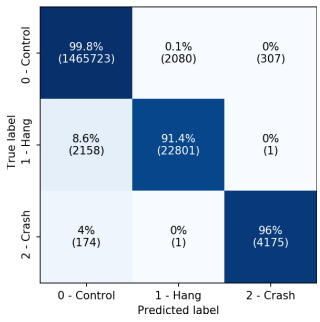

FIGURE 7. Plurality: ensemble of best individual models.

To explore if using more diverse learners would lead to better performance, the combinations of the 18 candidate base learners (explained in *Section IV*) were studied. This led to some promising results, as it was possible to observe that there were, in fact, ensembles that outperformed the best individual models. The best correctly predicted 99.8%, 97.1%, and 98.3% of *Control, Hang,* and *Crash* samples (as shown in *Fig. 8*). Although the improvements are not overwhelming (when compared with *Fig. 4*, but there was not much room for improvement to begin with) this ensemble was able to predict almost 5% more *Hang* and 2.1% more *Crash* samples. Still, this also came at the expense of slightly more FPs.

Concerning the execution overhead, nowadays most algorithms allow making predictions on a trained model almost instantly. Hence, besides the time needed to initially train the individual learners (which is entirely dependent on the algorithms and techniques selected for the base learners) the only overhead of plurality voting is to identify the most voted class at prediction time. As a result, it is mostly negligible.

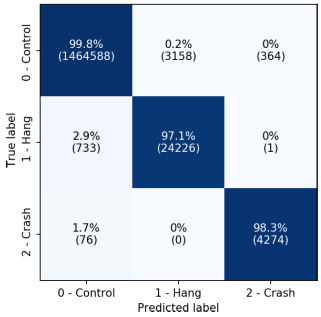

FIGURE 8. Best plurality voting ensemble.

A. PLURALITY VOTING

As stated in *Section IV*, the initial approach to selecting the learners was to choose the best model for each algorithm (i.e., Bagging, RF, and GB). Concerning the combination strategy, Plurality Voting was studied first. However, combining the best models did not lead to any improvement. Compared to the individual models, the ensemble had a slightly lower performance than the one obtained by GB, correctly predicting 99.8%, 91.4%, and 96% of *Control, Hang,* and *Crash* samples (depicted in *Fig. 7*). This was somewhat expected, as the performance of the individual models was already very good and they were created using similar ML techniques.

B. SOFT VOTING

Although using crisp predictions allowed improving the overall performance (when compared with individual models), it ignores the confidence that the algorithms have on their own predictions. Using the probability outputs allows combining the prediction and the confidence of the base learners [34].

Once again the performance of the best individual models (i.e., GB and Bagging with SMOTE oversampling, and RF with Random oversampling) combined was analyzed. Using the probability outputs it was possible to achieve slightly better performance, correctly classifying 99.9%, 93.6%, and 97.5% of *Control, Hang*, and *Crash* samples, respectively. However, although these results are better than any of the individual models, they are worse than what was achieved using Plurality Voting. This way, we decided to study the 4029 ensembles of 18 base learners. Results showed that, although some ensembles were able to perform better than the individual models, their performances were not significantly different from the ones observed with Plurality Voting (the best ensemble correctly classified 99.8%, 97.5%, and 98.6% of *Control, Hang*, and *Crash* samples, respectively).

Similarly to plural voting, the execution overhead of soft voting is negligible, as it only needs to identify the class with the highest averaged probability at prediction time.

C. STACKED GENERALIZATION

To model the predictions of the individual learners, Stacking was also considered. The probability outputs are likely the best option to use as the features for the meta-learner, as they allow using both the prediction and its confidence [34]. Still, the use of crisp labels was also studied for completeness.

Besides the choices already made in the previous sections (e.g., base learners), Stacking requires the definition and configuration of a meta-learner (e.g., algorithm and hyperparameters). Additionally, this approach also involves training the meta-learner before making predictions. To this end, an inner 2-fold cross-validation (using the training data from the outer cross-validation fold) was done to train the individual learners and make predictions, which were then used to train the

meta-learner. All this adds considerable complexity to the approach, thus for Stacking only the learners that constitute the top three ensembles (for both Plurality and Soft Voting) were considered and combined (instead of the original 18).

As described in *Section III*, various algorithms were considered to use as a meta-learner, and, although some were able to adequately model the outputs, none was able to achieve performances similar to those using Plurality or Soft Voting. More precisely, the best ensemble obtained through Stacking was using the NN algorithm, which was able to correctly predict 99.8%, 97.6%, and 96.8% of *Control, Hang,* and *Crash* samples, respectively. Although the remaining algorithms used (i.e., DT, Bagging, RF, and Logistic Regression) have been successfully used for several problems, a two-layer NN has been proved to be very effective and is considered a universal function approximator [16]. Still, no thorough analysis or tuning of these algorithms was done.

Regarding the execution overhead of Stacking, at prediction time it is also minimal. Although there is a meta-learner that has to make a prediction based on the outputs of the base learners, nowadays most algorithms do that almost instantaneously, and therefore it does significantly increase the execution time. However, this approach has a considerable higher complexity at training, as it requires a second level of cross-validation/optimization for the meta-learner. Nonetheless, in a real scenario training phases happen only periodically and thus such complexity may not be an issue.

VI. ENSEMBLES ANALYSIS

The previous analyses allowed us to conclude that by combining certain learners it was possible to increase their overall performance. Thus, the next step was focused on analyzing how the ensembles were composed to get some insights into why some models work better as a whole. This section focuses on Plurality Voting (which is the simplest approach) and Soft Voting (which achieved the best results).

A. GAINS DIRECTED GRAPH

To analyze how the different learners in an ensemble interact, a directed graph between every learner was plotted (similar to *Fig. 9*), showing how many different samples were correctly predicted between any two learners. When analyzing the graph for the ensemble composed of the best individual models it was possible to observe that they all had similar interactions among them, that is, they all had similar prediction gains to each other. However, although it appears that each one explores different parts of the problem, if none of the other models in the ensemble shares those predictions they will (incorrectly) win the vote at prediction time.

Concerning the best ensemble for Plurality Voting, it was composed of four models. By analyzing its models it was possible to observe that each one was created using different techniques: a model without any data sampling, two models with different oversampling techniques, and a fourth model combining both under- and oversampling. In fact, this observation was also present in the other two best ensembles for

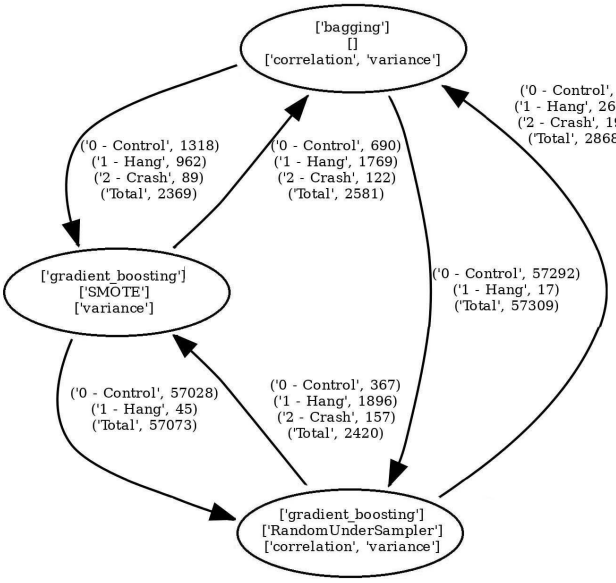

FIGURE 9. Best soft voting ensemble graph.

Plurality Voting. The fact that this combination has better results suggests that they are able to predict different parts of the problem, and when combined, complement each other.

When analyzing the best ensemble obtained by Soft Voting, we observed that it was composed of just three models, which are depicted in *Fig. 9*. Additionally, and similar to what was seen for the previous ensemble, the three models were built using different techniques: one without sampling, another using oversampling, and the third using undersampling. The fact that the ensemble is composed of only three learners also makes it easier to interpret. *Fig. 9* shows how they complement each other. More precisely, when comparing the model without sampling with the one that uses oversampling, the latter can predict more failures while correctly predicting almost the same number of *Control* samples. On the other hand, when analyzing the relationships with the model that used undersampling it was possible to observe that it could predict considerably more failures when compared to both the other models. Moreover, it was also possible to conclude that this model not only predicts more failures, it also correctly predicted almost all of the failures that the other two predicted (i.e., only 45 and 17 *Hang* different correct predictions were made by the GB and Bagging model respectively). Besides the indication that each of the sampling techniques allows the models to model different parts of the problem, it also suggests that most predictions in the ensemble are supported by at least two algorithms. More precisely, GB with SMOTE and Bagging support and complement the non-failure predictions, while GB with Random undersampling supports and complements the failure predictions of the remaining two learners.

B. VENN DIAGRAMS

Directed graphs do not allow us to visualize how the different predictions actually overlap between learners.

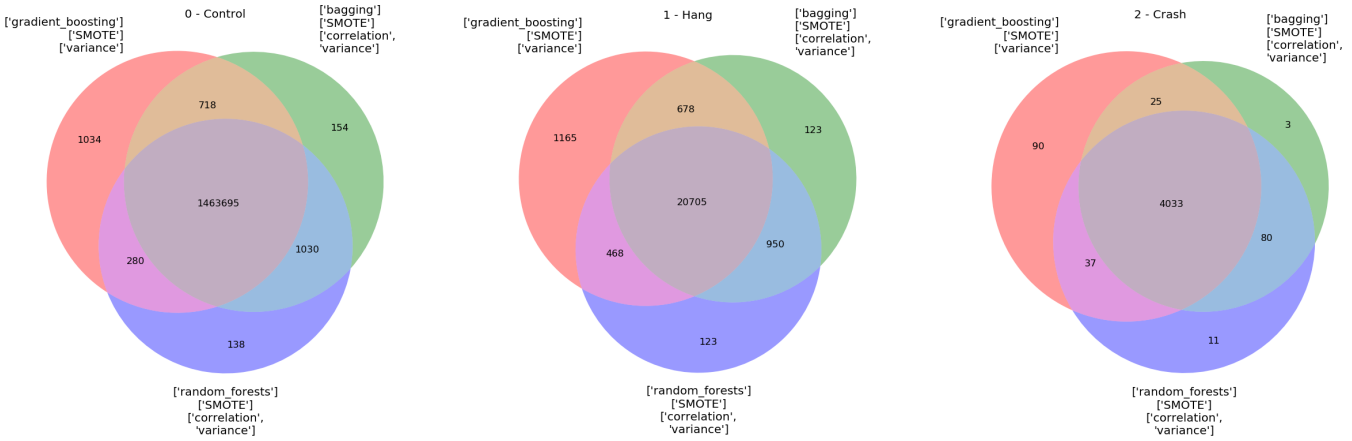

FIGURE 10. Ensemble of best individual models.

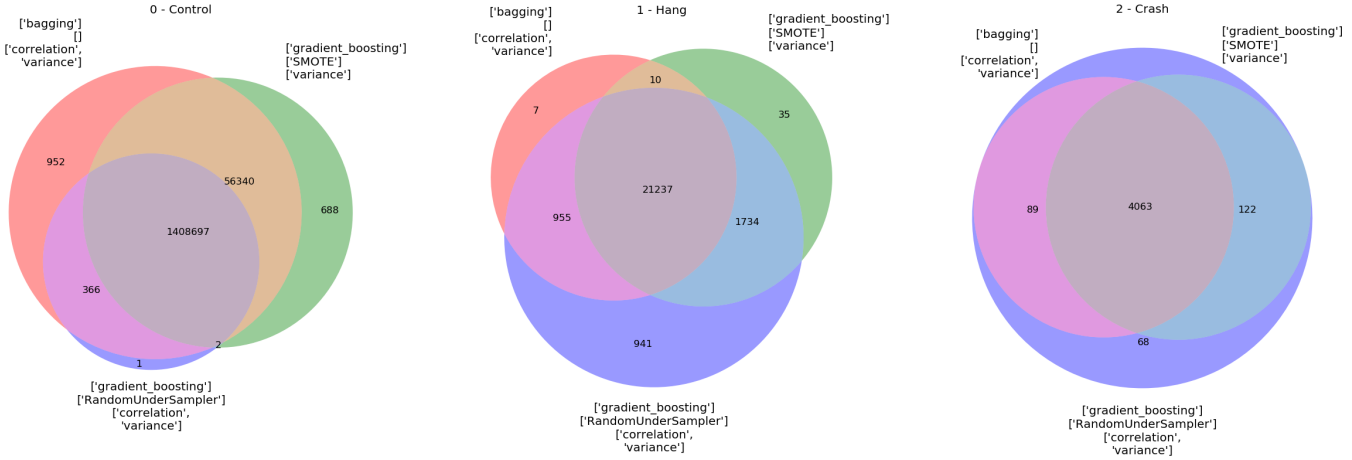

FIGURE 11. Best soft voting ensemble.

Thus, as some of the best ensembles were composed of only three models, Venn diagrams were plotted for each of the classes to illustrate how the different predictions relate. The diagrams for the ensemble composed of the best individual models and for the best Soft Voting ensemble can be seen in *Fig. 10* and *Fig. 11*, respectively. Note that a logarithmic base was used to calculate the areas of the diagrams, due to the fact that the differences between the number of individual and shared predictions were too steep (thus, the diagram would be mostly composed of just the overlapping areas). Hence, the size of the areas is not linearly proportional to the number of samples, but still provides a relative notion of proportion.

Concerning the ensemble composed of the best individual models (and confirming what was previously assumed), although there is a good cover of the problem space by the different learners, there is a considerable number of samples that are correctly predicted only by a single model. On the other hand, when analyzing the diagram for the best Soft Voting ensemble, it is possible to observe that the learners also complement each other in reaching different samples, but most samples are common to at least two models. In hindsight, this is one of the essential components for a good ensemble, to combine the outputs in a way that correct decisions are amplified [6]. Hence, although not all models will agree, for most cases at least two will, thus ''winning'' the vote.

The diagrams for both failure modes show that there is indeed a single model that is able to predict more failures. For the hang failure mode, there are still some samples that are only predicted by the other learners, whilst for the crash failures, such model can predict all the samples the others can and more. However, when analyzing its ability to predict *Control* samples it is possible to observe that the other learners can predict considerably more samples.

Finally, although these diagrams allow us to observe how the models in the ensemble interact, it is also relevant to study how the individual predictions relate to the ensemble predictions. Thus, another set of Venn diagrams was plotted, this time relating the individual (correct) predictions to those of the ensemble, as shown in *Fig. 12*. This validated the hypothesis that decisions made by a single algorithm will less likely be correctly predicted by the ensemble. In fact, most of the predictions that were made only by a single algorithm were not correctly predicted by the ensemble (e.g., concerning the non-failure samples, out of the 952 correct predictions

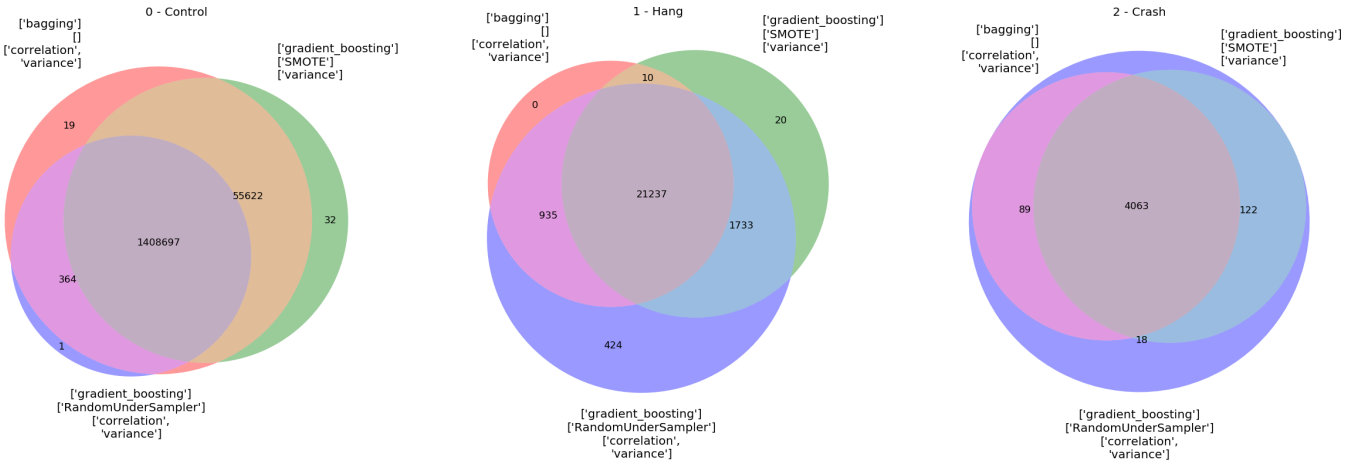

FIGURE 12. Best soft voting ensemble prediction contributions.

made by Bagging only 19 were also correctly predicted by the ensemble). On the other hand, some of the samples that were correctly predicted would not be possible using crisp outputs (otherwise the two wrong votes would prevail). This suggests that either the correct model was highly confident (enough to trump wrong classifications) or that these are fringe samples, where the learners are not confident of the decision, but their average probabilities lead to a correct prediction.

C. DIVERSITY METRICS

As stated in several related works, one of the key components for successful ensembles is that the base learners should be as accurate and diverse as possible. Although the approach used in this article followed a more driven and exhaustive direction, the next logical step was to validate if the best ensembles were, in fact, more diverse (according to the metrics identified in the literature) than those with a lower performance. However, although diversity is needed, the way to properly measure the necessary diversity in ensembles is still yet to be found [19]. Notwithstanding, the following diversity metrics were computed [19]: *Q Statistics, Correlation Coefficient, Disagreement, Double-fault, Entropy*, and *Interrater Agreement (using Fleiss Kappa [35])*. Briefly, the higher the Q Statistics, Correlation Coefficient, Double Fault, and Interrater Agreement, the less diversity there is; and the bigger the Disagreement and Entropy the more diversity exists. All metrics were calculated according to [19] except Entropy, which used a joint entropy formula [36].

Three ensembles were analyzed, all obtained through Soft Voting: *i)* an ensemble with lower performance (referred to as "worst" from now on) than that of combining all the best individual models (predicted 99.9%, 90.9%, and 96.5% of *Control, Hang*, and *Crash* samples); *ii)* the ensemble composed of the best individual models (referred to as "average" from now on, which predicted 99.9%, 93.5%, and 97% of *Control, Hang*, and *Crash* samples); and *iii)* the best ensemble (99.8%, 97.5%, and 98.6% of *Control, Hang*, and *Crash* samples). The computed metrics can be seen in *Table 4*. It is worth noting that although some ensembles are better than the others, they all are overall very good models with similar performances (i.e., all have similar performance pertaining to *Control* samples, which represent the vast majority of the samples; the main difference is mostly on the predicted failures, which represent only a small fraction of the predictions).

TABLE 4. Diversity metrics per ensemble (soft).

Metr. \ Ensem.	Worst Ensem.	Average Ensem.	Best Ensem.
Q Statistics ↓	0.991	0.996	**0.930**
Correlation ↓	0.366	0.538	**0.227**
Disagreement ↑	0.005	0.003	**0.027**
Double Fault ↓	0.001	0.002	**0.001**
Entropy ↑	0.187	0.193	**0.419**
Inter. Agreem. ↓	0.875	0.918	**0.560**

↑ The higher the better (i.e., more diversity)
↓ The lower the better (i.e., more diversity)

As expected, the values of the metrics for both the worst and average ensembles are quite similar, yet, there are some differences from the best. By comparing the worst and average ensemble, for all metrics besides entropy, the worst ensemble has values that suggest that it is more diverse than the average. Notwithstanding, these metrics may be correct, and the worst ensemble may have more diversity; however, as highlighted in [19], there is often no relationship between diversity and performance. That is, while almost all good ensembles are diverse, diversity per se does not mean that the ensemble will be good. Yet, when comparing both worst and average with the best, all metrics suggest that the latter is more diverse, which in this case is also translates into higher performance. These results suggest that while it appears to be possible to measure diversity, it is not always associated

with performance. Still, as the best ensemble has the highest diversity it suggests that diversity matters and that the metrics can, in fact, assist in the search for the best base learners.

D. APPROACH EFFECTIVENESS

All the analyses conducted in the previous sections suggest that the ensembles perform better. Notwithstanding, statistical comparisons are required to assess if they are, in fact, better than the individual models. To this end, we used F_2-score as the performance metric to compare three models: *i)* the best individual model (seen in *Fig. 5*); *ii)* the ensemble obtained by soft voting of the top 3 individual models (seen in *Fig. 7*); and *iii)* the best overall ensemble (obtained through soft voting, which correctly classified 99.8%, 97.5%, and 98.6% of *Control, Hang*, and *Crash* samples, respectively).

Because the dataset was the same for all experiments, a paired statistical test must be used. To decide between parametric and non-parametric tests, the normality of the data (using the *Lilliefors* and *Shapiro-Wilk* test) and the homogeneity of variance (using the *Levene* test) was analyzed. As the results do not satisfy both conditions, it must be a non-parametric test. Thus, *Friedman's ANOVA* was used [37], which gave a $p\_value = 2.46e{-}13$. Hence, it is safe to state that there are significant differences between at least two models for a significance level of 5% ($p\_value < 0.05$). To identify the differences, a *post-hoc* analysis was done using the *Bonferroni* correction for multiple comparisons. The results suggest that there are differences between all three models. That is, the best ensemble obtained through soft voting is better than the ensemble made of the best individual models, and that the latter is also better than the best individual model, for a significance level of 5% (all $p\_value < 0.05$).

VII. DISCUSSION

By exploring different algorithms and techniques, we were able to create very accurate solutions. Still, combining the best individual models was not able to achieve significantly higher performance. Our analyses suggest that this may be due to the fact that, although each model explores slightly different parts of the problem, those predictions are only made by one of the models in the ensemble, which is not enough to prevail over the (wrong) predictions of the remaining learners. However, the combination of models created using different techniques appears to produce ensembles where they complement each other in a constructive way, thus considerably improving the overall performance (with statistical significance), therefore answering *RQ1*.

Concerning the different combination techniques, both Plurality and Soft Voting achieved very good/similar results. By analyzing the models in the best Plurality Voting ensemble in more detail it was possible to observe that it was composed of four models that were all built using different techniques (with emphasis on the sampling methods). This suggests that each of these techniques allows the learner to model different aspects of the problem which, when combined, create an overall better classifier. The analysis of the best Soft Voting ensemble showed that it required only three learners to achieve similar performance. Due to the basic voting strategy of Plurality Voting, it is possible that it requires more learners (exploring with larger ensembles could possibly prove this) to handle conflicts and generate majorities, while Soft Voting can take advantage of the classifiers' confidence in the predictions. In fact, each of the models in the best Soft Voting ensemble was built using one (and different) sampling technique (i.e., no sampling, oversampling, and undersampling) further validating the previous assumption that these methods allow the models to specialize in certain parts of the problem. Stacking was not able to improve the individual results with any of the configurations. This can be due to the fact that the classifiers are already very good and it is not possible/easy to model their combined individual predictions to improve the results. Additionally, Stacking also adds a layer of complexity as it requires the choice and optimization of the meta-learner. Moreover, as this model must be trained using the predictions of the individual models on a "test" dataset (that is, not on the data used to train them) it requires a further level of cross-validation (requires training each base learner as many times as the number of folds). These conclusions answer *RQ2*, as besides achieving different performances, the combination techniques influence the structure and complexity of the ensembles.

The Venn diagrams for the best ensembles "proved" that those composed of models that explore different regions but that also overlap with other learners in the ensemble were able to achieve better results. On the other hand, learners that complemented each other but did not overlap were not able to achieve such good performance (possibly because predictions would be canceled out by the remaining learners). In fact, by analyzing the contribution of each learner to the final prediction of the ensemble it was possible to observe that most of the samples predicted by a single learner were misclassified by the ensemble. Finally, although diversity metrics can identify some sort of diversity in the ensembles, it is not always correlated with performance (thus, the one with the highest diversity may not be the best). Additionally, the "best" model depends on the requirements of the predictor, and thus it is not trivial to match diversity, a "non-subjective" value, with performance metrics that depend on a variable context. Still, when automating the process of selecting the base learners such metrics will likely be good indicators. These analyses allowed us to interpret and assess how the different models in an ensemble interact, thus answering the final research question, *RQ3*.

VIII. THREATS TO VALIDITY

This work intends to be an exploratory study on heterogeneous ensembles for supporting OFP. Although in our opinion it fulfills its purpose, some threats should be highlighted.

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. Instead, its purpose is to demonstrate that combining various ML algorithms and techniques can achieve higher performance when compared to individual classifiers. Moreover, due to the non-existence of related work on using ensembles for OFP it is not possible to compare the conclusions of our study. We can, however, briefly compare with other works using ensembles for software related domains to the extent that we have also found improvements in using ensemble methods. Seemingly, due to the lack of available failure prediction datasets, it was not possible to further validate the use of heterogeneous ensembles for OFP. Still, the combination of several different learners achieved similar promising results in various other areas, and the observations in our case study also validate this premise. Notwithstanding, we are currently trying to generate new datasets through fault injection using other and more recent OSs to further validate our results.

Due to the exploratory nature of this work, the models considered to form the ensemble were selected from those already known to perform well, and in an exhaustive way. While this approach may not be easy to generalize, the focus of this work was on trying to validate if it was possible to improve the performance of the individual classifiers. Additionally, while a relevant problem, this paper does not focus on exploring why some algorithms and techniques are individually better than others. Instead, we focus on how and why a set of learners complements/works better as a whole for failure prediction. A similar argument can be made for the parameters specific to OFP problem (e.g., Δt_l, Δt_p) which were chosen based on existing literature. Additionally, due to space and time constraints, it was also not possible to thoroughly explore all the ensemble combination methods, nor to exhaustively search for the best Stacking meta-learner. Still, the techniques considered represent the most commonly used approaches, and the algorithms used for the meta-learner also provide a decent insight into the performance that can be expected. Additionally, due to time constraints, the maximum number of learners in an ensemble was limited to a maximum of 4. While it is possible that some combination using more learners could lead to slightly higher performance, the more learners an ensemble has the more difficult/impossible it becomes to interpret and assess how they cooperate. In any case, while some good ensembles were composed of 4 models (e.g., the best plurality voting) most of the overall best ensembles were composed of only 3 base learners.

Finally, these experiments used the F_2-score metric to rank the solutions. Although this is valid for the context defined for this case study, the target metric should be chosen taking into consideration the intended usage scenario, as this may influence the combination of learners that maximizes it.

IX. CONCLUSION

OFP makes use of ML to create models that are able to accurately predict incoming failures to mitigate the effects of residual faults. Still, as different algorithms and techniques are able to explore different parts of the problem, combining multiple learners can often improve the overall performance.

This article intended to conduct an exploratory study on combining different individual learners (i.e., heterogeneous ensembles) to achieve higher performance. Although this study focused on certain performance metrics and parameters (e.g F_2-score) it can easily be adapted to take into consideration the requirements of the predictor.

By exploring several techniques and approaches to build ensembles it was possible to conclude that combining various learners can, in fact, lead to better models. This also revealed that the use of different combination techniques influences the structure and performance of the resulting ensembles. Moreover, analyzing the structure of the best ensembles suggested that the combination of learners built using different techniques (which can, therefore, ''specialize'' in different parts of the problem) creates the best ensembles. Finally, by exploring the interactions between the learners in the best ensembles it was possible to observe that, ideally, they should complement each other in such a way that correct decisions are amplified. These analyses also demonstrate that the ensembles can be interpreted, to a degree where both the contributions and relevance of each model for the ensemble can be assessed. Furthermore, these results were compared to the standard process of selecting base-learners (i.e., diversity metrics) and it was possible to observe that such methods do not always provide the expected results.

Future work includes exploring different techniques and approaches to find the most promising learners using a dataset from a recent OS. Additionally, different combination techniques should be explored, as well as further interpret why and how learners from a given ensemble can model different parts of the problem, and how it relates to the various failure modes.

REFERENCES

[1] B. Dhanalaxmi, G. A. Naidu, and K. Anuradha, ''A review on software fault detection and prevention mechanism in software development activities,'' *IOSR J. Comput. Eng.*, vol. 17, no. 6, pp. 661–2278, 2015.

[2] C. B. R. Ed Targett. *Amazon Outage: Estimated $99 Million Lost*. Accessed: Oct. 10, 2019. [Online]. Available: https://www.cbronline.com/news/amazon-outage-lost-sales

[3] B. C. Baraniuk. *Gps Error Caused '12 Hours of Problems' for Companies*. Accessed: Oct. 10, 2019. [Online]. Available: https://www.bbc.com/news/technology-35491962

[4] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, ''Basic concepts and taxonomy of dependable and secure computing,'' *IEEE Trans. Depend. Sec. Comput.*, vol. 1, no. 1, pp. 11–33, Jan./Mar. 2004.

[5] F. Salfner, M. Lenk, and M. Malek, ''A survey of online failure prediction methods,'' *ACM Comput. Surv.*, vol. 42, no. 3, pp. 1–42, 2010.

[6] R. Polikar, ''Ensemble based systems in decision making,'' *IEEE Circuits Syst. Mag.*, vol. 6, no. 3, pp. 21–45, Sep. 2006.

[7] Z.-H. Zhou, *Ensemble Methods: Foundations and Algorithms*. Boca Raton, FL, USA: CRC Press, 2012.

[8] V. S. Costa, A. D. S. Farias, B. Bedregal, R. H. Santiago, and A. M. D. P. Canuto, ''Combining multiple algorithms in classifier ensembles using generalized mixture functions,'' *Neurocomputing*, vol. 313, pp. 402–414, Nov. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231218307574, doi: 10.1016/j.neucom.2018.06.021.

[9] J. R. Campos, M. Vieira, and E. Costa, "Exploratory study of machine learning techniques for supporting failure prediction," in *Proc. 14th Eur. Dependable Comput. Conf. (EDCC)*, Sep. 2018, pp. 9–16.
[10] J. R. Campos, M. Vieira, and E. Costa, "Propheticus: Machine learning framework for the development of predictive models for reliable and secure software," in *Proc. IEEE 30th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Oct. 2019, pp. 173–182.
[11] F. Salfner and M. Malek, "Proactive fault handling for system availability enhancement," in *Proc. 19th IEEE Int. Parallel Distrib. Process. Symp.*, Apr. 2005, pp. 1–7.
[12] F. Salfner and M. Malek, "Using hidden semi-Markov models for effective online failure prediction," in *Proc. 26th IEEE Int. Symp. Reliable Distrib. Syst.*, Beijing, China, Oct. 2007, pp. 161–174.
[13] G. F. Hughes, J. F. Murray, K. Kreutz-Delgado, and C. Elkan, "Improved disk-drive failure warnings," *IEEE Trans. Rel.*, vol. 51, no. 3, pp. 350–357, Sep. 2002.
[14] E. Alpaydin, *Introduction to Machine Learning* (Adaptive Computation and Machine Learning), 3rd ed. Cambridge, MA, USA: MIT Press, 2014.
[15] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263–1284, Sep. 2009.
[16] H. Trevor, T. Robert, and F. Jerome, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (Springer Series in Statistics), 2nd ed. New York, NY, USA: Springer, 2009.
[17] J. P. Marques de Sá, *Pattern Recognition*. Berlin, Germany: Springer, 2001.
[18] M. P. Sesmero, A. I. Ledezma, and A. Sanchis, "Generating ensembles of heterogeneous classifiers using Stacked Generalization," *Wiley Interdiscipl. Rev., Data Mining Knowl. Discovery*, vol. 5, no. 1, pp. 21–34, 2015.
[19] L. I. Kuncheva and C. J. Whitaker, "Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy," *Mach. Learn.*, vol. 51, no. 2, pp. 181–207, 2003.
[20] S. Nagi and D. K. Bhattacharyya, "Classification of microarray cancer data using ensemble approach," *Netw. Model. Anal. Health Inform. Bioinform.*, vol. 2, no. 3, pp. 159–173, 2013. [Online]. Available: http://link.springer.com/10.1007/s13721-013-0034-x
[21] O. Malgonde and K. Chari, "An ensemble-based model for predicting agile software development effort," *Empirical Softw. Eng.* vol. 24, no. 2, pp. 1017–1055, 2019.
[22] H. Alsawalqah, H. Faris, I. Aljarah, L. Alnemer, and N. Alhindawi, "Hybrid SMOTE-ensemble approach for software defect prediction," in *Proc. Comput. Sci. On-line Conf.* Cham, Switzerland: Springer, 2017, pp. 355–366.
[23] Y. Pang, X. Xue, and A. S. Namin, "Early identification of vulnerable software components via ensemble learning," in *Proc. 15th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2016, pp. 476–481.
[24] A. A. Aburomman and M. B. I. Reaz, "A survey of intrusion detection systems based on ensemble and hybrid classifiers," *Comput. Secur.*, vol. 65, pp. 135–152, Mar. 2017, doi: 10.1016/j.cose.2016.11.004.
[25] T. T. Khuat and M. H. Le, "Ensemble learning for software fault prediction problem with imbalanced data," *Int. J. Electr. Comput. Eng.*, vol. 9, no. 4, pp. 3241–3246, 2019.
[26] S. Tofighy, A. A. Rahmanian, and M. Ghobaei-Arani, "An ensemble CPU load prediction algorithm using a Bayesian information criterion and smooth filters in a cloud computing environment," *Softw.-Pract. Exper.*, vol. 48, no. 12, pp. 2257–2277, 2018.
[27] A. A. Rahmanian, M. Ghobaei-Arani, and S. Tofighya, "A learning automata-based ensemble resource usage prediction algorithm for cloud computing environment," *Future Gener. Comput. Syst.*, vol. 79, pp. 54–71, Feb. 2018.
[28] I. Irrera, J. Durães, M. Vieira, and H. Madeira, "Towards identifying the best variables for failure prediction using injection of realistic software faults," in *Proc. IEEE 16th Pacific Rim Int. Symp. Dependable Comput.*, Dec. 2010, pp. 3–10.
[29] I. Irrera, J. Durães, H. Madeira, and M. Vieira, "Assessing the impact of virtualization on the generation of failure prediction data," in *Proc. 6th Latin-Amer. Symp. Dependable Comput.*, Apr. 2013, pp. 92–97.
[30] L. Breiman and P. Spector, "Submodel selection and evaluation in regression. The X-random case," in *Proc. Int. Stat. Rev./Revue Int. de Statistique*, 1992, pp. 291–319.
[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.
[32] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Inf. Process. Manag.*, vol. 45, no. 4, pp. 427–437, 2009.
[33] D. Powers, "Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation," *J. Mach. Learn. Technol.*, vol. 2, no. 1, pp. 1–24, 2011.
[34] K. M. Ting and I. H. Witten, "Issues in stacked generalization," *J. Artif. Intell. Res.*, vol. 10, no. 1, pp. 271–289, 1999.
[35] J. L. Fleiss, *Statistical Methods for Rates and Proportions*. Hoboken, NJ, USA: Wiley, 1981.
[36] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Hoboken, NJ, USA: Wiley, 2012.
[37] A. Field, *Discovering Statistics Using IBM SPSS Statistics*, ed. Newbury Park, CA, USA: SAGE, 2017.

JOÃO R. CAMPOS is currently pursuing the Ph.D. degree with the University of Coimbra, Coimbra, Portugal, under the supervision of professors Marco Vieira and Ernesto Costa. His Ph.D. focuses on using Online Failure Prediction (OFP) to improve and characterize the trustworthiness of software systems. His main research interests include dependability and artificial intelligence, mainly on the use of machine learning and evolutionary computation to improve failure prediction and trustworthiness characterization.

ERNESTO COSTA is currently a Full Professor with the Department of Informatics Engineering, University of Coimbra. Over the years, he published over 200 articles in books, journals, and proceedings of conferences. His research interests include bio-inspired artificial intelligence, developing novel algorithms and applying them to design, optimization and learning problems, and promoting the cross-fertilization of evolutionary computation and machine learning. He participated in several projects and received several best paper awards. He was a recipient of the 2009 EvoStar Award for Outstanding Contributions to the Field of Evolutionary Computation.

MARCO VIEIRA received the Ph.D. degree from UC, Coimbra, Portugal, in 2005. He is currently a Full Professor with the University of Coimbra. His research interests include dependability and security assessment and benchmarking, fault injection, software processes, and software quality assurance, subjects in which he has authored or coauthored more than 200 articles in refereed conferences and journals. He has participated and coordinated several research projects, both at the national and European level. He has served on program committees of major conferences of the dependability area and acted as a referee for many international conferences and journals in the dependability and security areas.