Liu, S., Reviriego, P., Hernandez, J. A. & Lombardi, F. (2021). Voting Margin: A Scheme for Error-Tolerant k Nearest Neighbors Classifiers for Machine Learning. *IEEE Transactions on Emerging Topics in Computing*, 9(4), 2089-2098.

DOI: 10.1109/tetc.2019.2963268

Voting Margin: A Scheme for Error-Tolerant *k* Nearest Neighbors Classifiers for Machine Learning

Shanshan Liu, *Member, IEEE*, Pedro Reviriego, *Senior Member, IEEE*, José Alberto Hernández and Fabrizio Lombardi, *Fellow, IEEE*

Abstract—Machine learning (ML) techniques such as classifiers are used in many applications, some of which are related to safety or critical systems. In this case, correct processing is a strict requirement and thus ML algorithms (such as for classification) must be error tolerant. A naive approach to implement error tolerant classifiers is to resort to general protection techniques such as modular redundancy. However, modular redundancy incurs in large overheads in many metrics such as hardware utilization and power consumption that may not be acceptable in applications that run on embedded or battery powered systems. Another option is to exploit the algorithmic properties of the classifier to provide protection and error tolerance at a lower cost. This paper explores this approach for a widely used classifier, the *k* Nearest Neighbors (*k*NNs), and proposes an efficient scheme to protect it against errors. The proposed technique is based on a time-based modular redundancy (TBMR) scheme. The proposed scheme exploits the intrinsic redundancy of *k*NNs to drastically reduce the number of re-computations needed to detect errors. This is achieved by noting that when voting among the *k* nearest neighbors has a large majority, an error in one of the voters cannot change the result, hence voting margin (VM). This observation has been refined and extended in the proposed VM scheme to also avoid re-computations in some cases in which the majority vote is tight. The VM scheme has been implemented and evaluated with publicly available data sets that cover a wide range of applications and settings. The results show that by exploiting the intrinsic redundancy of the classifier, the proposed scheme is able to reduce the cost compared to modular redundancy by more than 60% in all configurations evaluated.

Index Terms— Machine learning, *k* nearest neighbors, error tolerance

1 Introduction

CLASSIFICATION is one of the most popular and widespread applications of *Machine Learning* (ML) [1]. *Supervised Machine Learning* methods start from an existing labeled dataset whereby a collection of data points (characterized by a set of input features *X)* are assigned to a target label *y*. From this information, the ML method seeks to build a function *f* that best relates the input features with the target label for a given training set, such as for instance by reducing some error function. When the target label *y* is continuously valued, the supervised learning strategy is referred to as *regression*; however if the label is discrete, the function *f* performs a *classification* task. The ML model characterized by such a function *f* is said to be well constructed if the function is capable of *generalizing*, in other words, if it is capable of predicting the correct label for a new unseen data point [1]. A number of algorithms have been proposed in the technical literature to accomplish *accurate classifiers*, such as *Logistic Regression, Support Vector Machines, Decision Trees, and Deep Neural Networks* [1],[2].

One of the simplest, yet powerful classification algorithms is the so-called *k* Nearest Neighbors (*k*NNs) [3]. This method, also called *instance-based* or *lazy learning*, uses the *k* closest elements (neighbors) in the dataset to predict the output class of a new data point by taking majority voting of the *k* closest neighbors [4].

ML classifiers are implemented by computing or embedded systems that are prone to suffer a number of errors and failures [5]. In some applications, failures are not acceptable, and the design must be able to mitigate them. For example, in automotive applications, systems have to comply with error tolerance requirements depending on criticality [6]. This means that the processors or algorithms used in those applications have to be designed to tolerate errors [7]. For example, radiation induced soft errors are one of the main reliability challenges for electronic systems. Errors can affect any circuit node and technology process or circuit level protection schemes incur in large area and power dissipation overheads [8]. An alternative approach to protect applications that have specific algorithmic properties is to exploit them to detect and correct errors at a lower cost than replication or low level techniques [9]. This has been done extensively for signal processing applications by using for example reduced precision redundancy [10] or by exploiting the adaptive nature of few signal processing algorithms [11].

As ML algorithms are incorporated into safety critical systems, there is a need for efficient error-tolerant

- *Manuscript received August 18 and revised November 15, 2019.*
- *S. Liu and F. Lombardi are with Northeastern University, Dept. of ECE, Boston, MA 02115, USA (email: ssliu@coe.neu.edu, lombardi@ece.neu.edu).*
- *P. Reviriego and J.A. Hernández are with Universidad Carlos III de Madrid, Av. Universidad 30, Leganés, Madrid, Spain (email: revirieg@it.uc3m.es, jahgutie@it.uc3m.es).*

implementations [12],[13],[14]. Therefore, there is a strong motivation to design algorithmic-based error-tolerant schemes for classifiers that can reduce the protection overhead of existing schemes.

In this paper, Voting Margin (VM), a scheme to implement error-tolerant *k*NNs classifiers is proposed and evaluated. The results show that by exploiting the algorithmic features of the classifier, an efficient protection can be obtained at a significantly lower cost than existing techniques.

The remaining part of the paper is organized as follows. Section 2 covers the background on the *k*NNs algorithm and its implementation. Then, the effects of errors and existing error-tolerant schemes are also discussed. The proposed VM scheme is presented in Section 3 and evaluated in Section 4. The paper ends in Section 5 with the conclusion and a discussion of future work that could extend the innovative scheme here presented.

2 Preliminaries

This section first provides a brief description of the *k*NNs algorithm and its common implementation; in the second part, the effects of errors on a *k*NNs classifier and existing protection schemes for different implementation options are discussed. Finally, the target implementation and error model for the proposed scheme are outlined.

2.1 *k*NNs Algorithm and Implementation

The *k*NNs algorithm is illustrated in Figure 1. The first step of the algorithm is to identify the *k* nearest neighbors to the element being classified (grey bullet in the figure) for some distance metric (typically Euclidean distance). To do so, the elements must be retrieved from memory and their distances computed keeping a list of the *k* ones with lowest distances. The second step is to vote among the *k* nearest neighbors to determine the class assigned to the grey element.

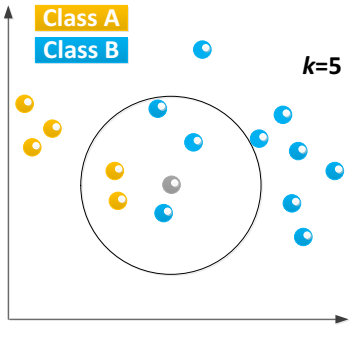

Figure 1 Illustration of the *k*NNs algorithm. The yellow and blue elements are stored with their class. The grey element is the one being classified. As $k = 5$, first the five elements closest to it are identified (shown by the solid circle). Then a vote is taken among them to determine the class of the grey element.

The first step accounts for most of the computational effort. This depends largely on the number of elements stored and on the number of features used for classification because for each element, all features are used to compute the distance with the element being classified (and the distance is computed for all elements). If the number of elements is not large, an exhaustive comparison against all elements can be performed resulting in a computational cost that is linear with the size of the stored set. If the number of elements is large, then more sophisticated schemes that use data structures to store and search the elements may be used [15]. Regardless of the data structure used, in this step, distances are calculated between the element being classified and all or some of the stored elements.

The second step is significantly simpler; we just take the *k* nearest neighbors and vote to obtain the class assigned to the element being classified.

Depending on the data structure used, the size of the data set stored and the application constraints, the algorithm may be implemented on different platforms. Those include large computing systems that use MapReduce to implement the algorithm [16], specialized processors such as GPUs [17], FPGAs [18], microcontrollers and ASICs [19].

2.2 Errors on *k*NNs Classifiers

As discussed previously, there are two main components in a *k*NN implementation: 1) the classified elements that are stored in memory and 2) the computation of distances and comparisons to find the nearest neighbors. Next, the protection of both these components against soft errors is discussed as dependent on the target implementation.

The classified elements are commonly stored in a memory in which errors may occur; however, memories can be efficiently protected against errors by error correction codes as widely used in computing systems [20]. For example, a 64-bit data word can be protected against single bit errors with only 7 additional bits per word [21] and single bit errors can be detected by adding a parity bit per word.

The computation of distances can be also prone to suffer from errors in the arithmetic circuitry. A number of techniques can be used to detect and correct these errors by adding spatial redundancy. For example, Reduced Precision Redundancy (RPR) [10], [22], residue checking [23] or parity prediction [24] can be used to detect arithmetic errors. Those techniques add a detection circuitry that operates in parallel with the main circuit and check the validity of its output. The detection techniques combined with dual modular redundancy can correct an error by identifying the erroneous module and taking the output from the other module [8]. This requires a significant overhead, because the arithmetic circuitry has to be duplicated and detection circuitry must be added. Another alternative is to triplicate the arithmetic circuitry and vote among the three copies. However, this further increases the overhead [8]. Therefore, it is evident that techniques to correct errors based on spatial redundancy imply a large overhead in terms of additional circuitry. Another option for protecting arithmetic operations against soft errors is to use time redundancy; so instead of computing an operation at the same time in replicated modules, such operation is computed several times using the same circuit [8]. Time redundancy is commonly used in systems implemented on embedded processors, for example by using redundant threads that compute the

same operation and compare the results [25]. The overheads associated with time redundancy are still significant, because if an operation is executed twice, the power consumption and use of processor resources are also doubled. The time to complete the operations is also double if the operations are executed sequentially. When run on different cores of a processor in parallel, the time could be reduced. When using time redundancy, it is common to run the operation twice and only when the results are different, then a third execution run is used to obtain the correct result [26]. This has a cost of approximately 2x in terms of computation and power consumption (assuming errors are rare events) which may be acceptable for some applications. Hereafter in this paper we require the execution of an operation twice to detect errors and a third time to correct them as a time-based modular redundancy (TBMR) is employed.

Therefore, regardless of whether spatial or temporal redundancy is used to correct errors in the distance computations, the cost could be significant. Hence, the reduction in the cost of protecting classifiers against errors on arithmetic operations is of interest.

2.3 Target Implementation and Error Model

Hereafter in this paper, we assume that *k*NNs is implemented in a system based on a microprocessor or microcontroller that stores the dataset on memory. The computations are performed in the arithmetic units of the processor. The main reason to focus on this implementation is that it is probably the most generic and used implementation.

It is also assumed that the processor is given and thus we cannot make modifications on its internal structure to provide error protection. Further, it is assumed that the processor has no protection against soft errors on its arithmetic units (as often encountered in most processors).

The error model considers soft errors on the arithmetic units. The reasoning is that memories can be efficiently protected with error correction codes as discussed previously and are thus less of a concern. Further, as soft errors are rare, it is assumed that during the classification of an element, at most a single soft error occurs. This assumption is common when protecting against soft errors as uphold also for traditional schemes, such as triplication with voting [5],[8].

The technique that is used as baseline for comparison is TBMR executing the operation twice to detect errors and a third time to correct them when needed. As soft errors are random and seldom events, most of the executions would be error free and thus the cost of TBMR is approximately 2x.

3 Voting Margin based Protection

The proposed scheme is based on the following two observations.

1. An error in a distance computation can only modify one of the k nearest neighbors selected by the algorithm.
2. When the vote among the k nearest neighbors has a majority margin, then a single error cannot affect the classification result.

Therefore, when there is a sufficient majority in voting, there is no need for recalculation to detect errors as the classification result is already correct. Therefore, an error in a voter does not affect the result; this can be used to drastically reduce the overhead (in both time and power) needed to detect errors as shown in the next section.

To better illustrate the proposed margin strategy, the voting results for one of the datasets that will be used in the evaluation are shown in Table 1. In this case the *k*NNs classifier uses k = 3 (three voters), thus the following options are possible: (i) all three neighbors agree either on class A or B; or (ii) two neighbors vote for one class, while the third one votes for the other one. In the former case, there exists a margin because when a voter produces an error, the other two already agree on a majority. However, the second case has no margin because an error in a neighbor causes an erroneous voting decision. In particular, as per Table 1 only two of the four voting results have margin. In general, for a vote of k, k-1 out of the k+1 possible voting results have margin.

TABLE 1
Distribution of Votes for Class A in the Sonar Dataset When Using k=3 Across All the Elements in the Test Set (62 elements).

Number of Class A	Frequency
0	29
1	9
2	5
3	19

Consider again Table 1, in the majority of the votes, there are either none or three elements of class A. Specifically, in 46.77% of the cases there are no voters of class A and for 30.65% three. This adds up to 77.42% of votes that have a margin. For all of them, there is no need to recompute to detect errors.

Before describing the proposed scheme in detail, consider the first observation. An error in the distance computation can affect the set of k elements selected as nearest neighbors in two ways. This is shown in Figure 1 (where the yellow (blue) elements belong to class A (B) and the grey element needs to be classified; elements in the range of the black solid (dot) circle are (were) the *k*NNs under a single error (in the error free case)):

- *Type 1:* An element that should be in the set of k nearest neighbors has a larger distance as result of the error and thus, it is no longer in the set, the k+1th nearest neighbor is in the set instead (as shown in Figure. 2 a)).
- *Type 2:* An element that should not be in the set of k nearest neighbors has a smaller distance as result of the error that puts it in the set. The k^{th} nearest neighbor is therefore no longer in the set (as shown in Figure. 2 b)).

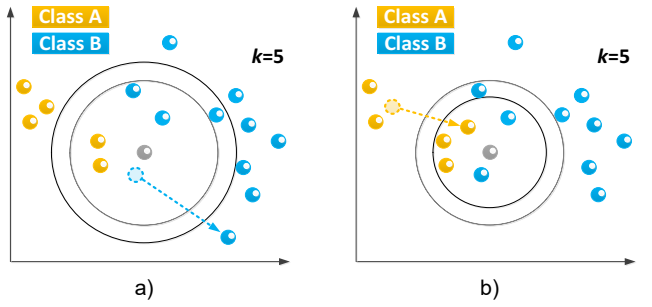

Figure 2 An error in a distance computation: a) an error of type 1; b) an error of type 2.

Intuitively, the second case seems to have a larger impact on the classification result, because the element that is brought into the decision has nothing to do with the original set of neighbors and thus, it can take any class value. Instead for the first case, the voting is done among the $k+1$ nearest neighbors, except the one that has been pulled away. The elements in this set are more likely to have the class value that would be selected by an error free kNNs.

Consider the second observation and an example for k = 5. Then, a vote can take a majority of 3, 4 or 5 because the threshold is equal to $\lceil (k+1)/2 \rceil$ (assuming that k is always odd). In the last two cases, an error that changes one of the elements in the voting set cannot change the result. For example, an error on a vote with a majority of 4 can only in the worst case make it a vote with a majority of 3, but the outcome would be the same. Those cases have a VM, which means that a change of one of the elements does not change the outcome of the vote.

It is interesting to note that the vote would be correct in some cases even if there is no VM, i.e., the vote is equal to the threshold (taking a majority of 3 in the case of k=5). For example, if applicable and the k^{th} nearest neighbor belongs to the same class as the minority, an error of type 1 cannot affect the voting result (that element was brought into the set by an error). The reason is that no matter if the element that has been replaced by the current k^{th} nearest neighbor belongs to the majority or minority class, then there are at least $\lceil (k+1)/2 \rceil$ majority elements, leading to the same classification result. This can be seen from Figure 3, which illustrates the example of k=5 with no VM (three blue elements and two yellow elements so B is the majority class). In Figure 3 a), the current k^{th} nearest neighbor (a yellow element that belongs to the minority class A) has replaced a blue element due to an error of type 1. Then in the error free case, there are four blue elements and one yellow element, so B was the majority class, which is the same as the result in the presence of an error of type 1. In Figure 3 b), a yellow element has been replaced by another yellow element (the current k^{th} nearest neighbor), so the classification result is not changed by the error.

Conversely, if the $k+1^{th}$ nearest neighbor belongs to the same class as the majority, an error of type 2 could not affect the voting result because there are also at least $\lceil (k+1)/2 \rceil$ elements belonging to the majority class in the error free case. Figure 3 c) and d) show the cases in which an element (that is the current $k+1^{th}$ nearest neighbor) has been taken out of the set due to an error of type 2; so if the element moved by the error is a yellow element, there are four blue elements in the error free case, so the classification result is B. Therefore, moving the yellow element reduces the majority to three, but the result is also B. Instead, if a moved element is blue, then there are three blue elements in the error free case and we still have three after the error. The result in both cases is B and the error has no effect in the classification result. Therefore, when both conditions are met (the k^{th} nearest neighbor belongs to the minority class and the $k+1^{th}$ nearest neighbor to the majority class), even if there is no VM, the result is correct if an error has occurred in a distance computation.

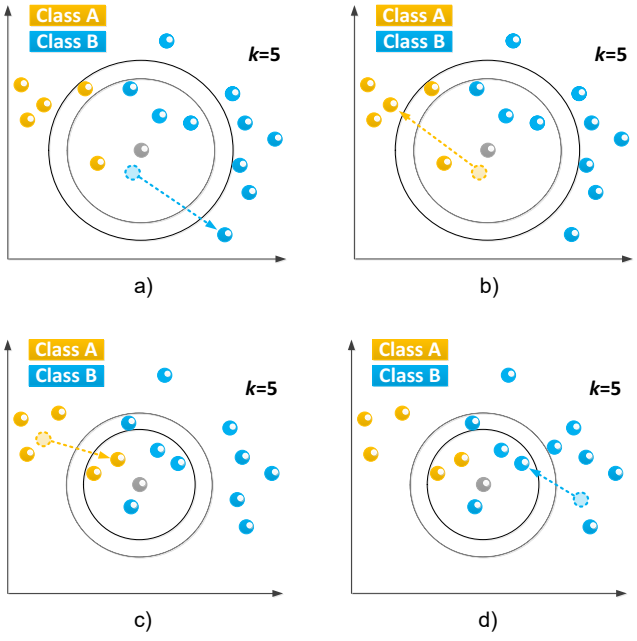

Figure 3 Classification result for the new element is B with no VM: a) the k^{th} nearest neighbor belongs to the minority class A under an error of type 1 affecting a blue element; b) the k^{th} nearest neighbor belongs to the minority class A under an error of type 1 affecting a yellow element; c) the $k+1^{th}$ nearest neighbor belongs to the majority class B under an error of type 2 affecting a yellow element; d) the $k+1^{th}$ nearest neighbor belongs to the majority class B under an error of type 2 affecting a blue element.

After this initial discussion, the proposed VM algorithm can be presented. Figure 4 illustrates the different steps in the algorithm. The first step computes the kNNs as in an unprotected implementation. Once that is completed, the second step checks if the vote is reliable due to VM, if so, the classification result is output. If not, the condition that ensures a reliable vote even if there is no VM, is checked; if so, we can again safely output the classification result. Finally, if the vote cannot be determined to be reliable, it is required to recompute the kNNs and compare it with the first computation to detect errors. If no error is detected, the classification result is provided as output. If an error is detected, the kNNs is computed for a third time and a vote is performed to correct the error.

The main benefit of the proposed scheme is that when

a vote can be determined to be reliable, there is no need to recompute the *k*NNs. Therefore, the benefit of the proposed scheme depends on the classification dataset, the accuracy of the classifier and the value of *k*. In the next section the proposed algorithm is evaluated for a wide range of datasets and configurations to determine the settings for which it provides significant benefits.

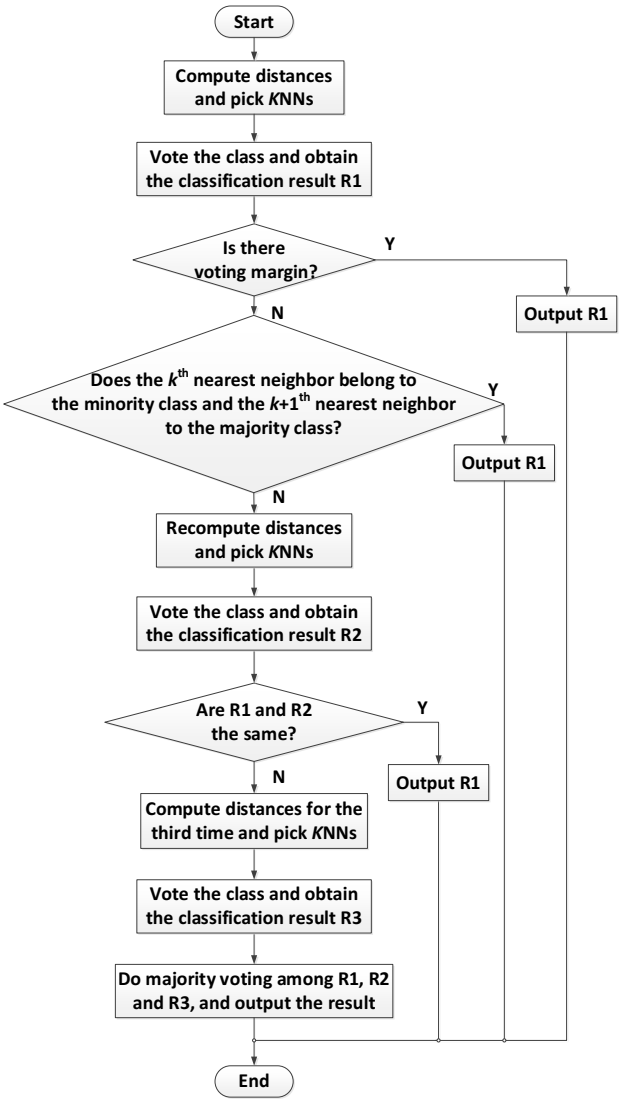

Figure 4 Diagram of the proposed Voting Margin (VM) algorithm

The proposed VM scheme protects against errors in the distance computations as the main part of the *k*NNs execution. There are however other parts of the algorithm on which errors could occur. For example, once the *k*NNs have been selected, voting counts the number of elements in each class. An error in the counting can modify the classification result. Similarly, to select the *k*NNs, the distances must be compared and again an error in the comparisons may lead to an incorrect selection of the *k*NNs. To prevent errors on these operations from modifying the classification result, TBMR can be used, i.e. counting of the votes and the comparison of the distances can be executed twice to detect errors. If an error is detected, a third execution is used to correct the error. Since the comparison and counting of the votes are significantly simpler than the distance computations, the use of TBMR on them only introduces a small overhead in the classification process.

In the next section, the use of the proposed VM scheme for the classification of elements in *k*NNs is evaluated, because it is in classification when the algorithm is more likely to run on a resource constrained system. However, VM could also be used to avoid errors during the hyperparameter tuning with cross-validation that determines the optimal *k* to use for classification. The evaluation of VM during the cross-validation phase is left for future work. Intuitively, it seems that this phase is more robust as a single error in the classification of an element does not affect the selection of *k* in most cases.

4 Evaluation

The proposed VM scheme has been evaluated using several widely used datasets taken from a public repository [27]. The selected datasets cover a wide range of applications and have different number of features, classification performance and optimal *k*. The goal of using a large number of datasets is to show that the proposed VM scheme consistently provides significant computation savings regardless of the dataset used. In the experiments, a wide range of values of *k* in addition to the optimal one are also evaluated. The main reason for this evaluation is to assess the dependency of the savings of the VM scheme with *k*. Finally, in all experiments the Euclidean distance on the feature values is used in the algorithm, the evaluation of other distance metrics is left for future work.

The relevant parameters of the datasets are summarized in Table 2 and the performance of *k*NNs for different values of *k* are shown in Figure 5. This large number of datasets allows to analyze the performance of the proposed scheme under different conditions and obtain insights on the configurations for which it is more effective. In all cases, 70% of the data set elements have been used for training, while the remaining 30% of the data is left for testing. The 70% training set is split into 10 blocks of equal size to run the well-known 10-fold cross-validation methodology [35] for the accurate selection of the algorithm's hyperparameters, in this case the optimal number of neighbors *k*. The average accuracy is plotted in Figure 5.

As discussed in section 2.3, the baseline for comparison is TBMR running each distance computation twice to detect errors. Therefore, in TBMR the percentage of re-computations is always 100% and all errors are corrected. In the proposed VM scheme, also all errors are corrected, but the percentage of re-computations is lower. Therefore, the main metric for comparison is the percentage of recomputations. For example, if for a given dataset, VM requires only 50% of recomputations, its cost is approximately 1.5x compared to 2x of TBMR.

TABLE 2
CONSIDERED DATASETS WITH TWO CLASSES

Dataset	Application	# Elements	# Features	Optimal *k*	Top accuracy	Class A	Class B
Pima Indians diabetes	Medicine	768	8	19	74.07%	34.90%	65.10%
EEG eye state	Medicine	14980	14	3	82.25%	44.88%	55.12%
LSVT voice rehabilitation [28]	Medicine	126	308	9	77.98%	33.33%	66.67%
Sonar [29]	Physics	208	60	3	80.18%	46.63%	53.37%
Ionosphere [30]	Physics	351	34	11	60.00%	35.90%	64.10%
Electrical grid stability simulated data [31]	Electricity	10000	13	19	94.49%	36.20%	63.80%
Banknote authentication	Business	1372	4	7	99.86%	44.46%	55.54%
Qualitative bankruptcy [32]	Business	250	6	3	96.88%	57.60%	42.40%
Phishing websites [33]	Computer	2456	30	5	93.25%	44.54%	55.46%
Climate model simulation crashes [34]	Physics	540	17	5	91.55%	64.81%	35.19%

TABLE 3
PERCENTAGE OF ERRORS THAT MODIFY THE CLASSIFICATION RESULT IN THE UNPROTECTED *k*NNS SCHEME (RESULTS FOR THE OPTIMAL K ARE HIGHLIGHTED IN BOLD AND UNDERLINED)

k	Pima Indian diabetes	EEG eye state	LSVT voice rehabilitation	Sonar	Ionosphere	Electrical grid stability simulated data	Banknote authentication	Qualitative bankruptcy	Phishing websites	Climate model simulation crashes
3	2.88%	**0.10%**	2.68%	**4.13%**	1.52%	2.20%	0	**0.84%**	1.12%	2.37%
5	2.71%	0.06%	1.71%	2.24%	1.19%	1.38%	0	0	**0.77%**	**1.11%**
7	2.01%	0.06%	1.82%	1.19%	0.56%	0.98%	**0**	0.95%	0.70%	0.41%
9	1.85%	0.04%	**1.05%**	2.85%	0.65%	0.87%	0	1.39%	0.52%	0.54%
11	1.40%	0.04%	0.95%	1.74%	**1.83%**	0.67%	0	0.88%	0.49%	0.46%
13	1.36%	0.03%	0.97%	2.18%	1.02%	0.46%	0	1.47%	0.44%	0.04%
15	1.02%	0.03%	1.66%	3.19%	0.34%	0.57%	0	0.97%	0.43%	0.02%
17	0.79%	0.03%	1.63%	1.74%	0.38%	0.51%	0	1.61%	0.30%	0.07%
19	**0.89%**	0.02%	1.84%	1.76%	0.58%	**0.40%**	0	3.20%	0.36%	0

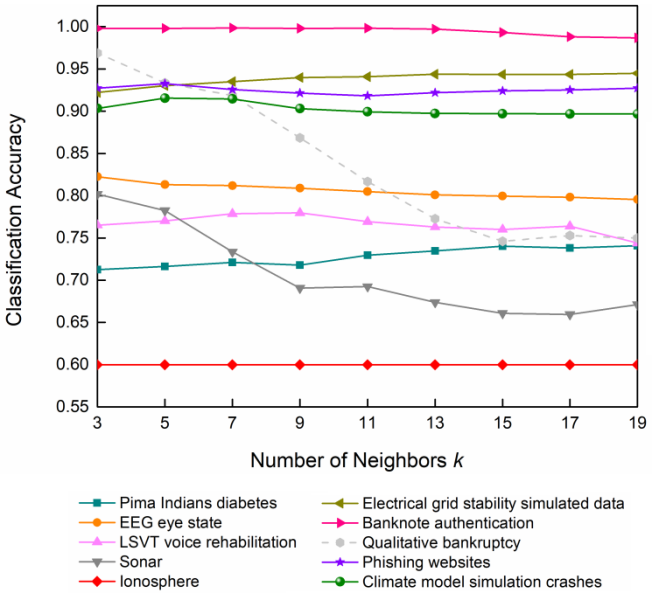

Figure 5 Classification accuracy of different datasets with different number of neighbors *k*

In a first experiment, errors were injected in the unprotected classifiers. The position of the errors was selected randomly with a uniform distribution and the magnitude of the errors was set as a random value within the range of all computed distances. A single error on a distance was inserted when classifying an element and the process was repeated 10,000 times for each element.

The results are presented in Table 3 in which the values that correspond to the optimal *k* are highlighted in bold and underlined. The *k*NNs implementation is capable of masking many errors; for all configurations considered, the percentage of errors that modify the classification result, is below 5%. This is an interesting observation that shows the intrinsic robustness of the *k*NNs algorithm and means that for some applications that can tolerate a few errors, an unprotected implementation may be acceptable. However, for critical applications, additional protection needs to be added to ensure that single errors do not lead to a failure.

In the second experiment, we simulated the proposed scheme to validate its effectiveness and evaluate its overhead in terms of execution time. The results are shown in Table 4; it shows the percentage of elements for which we need to recompute the *k*NN*s* to detect errors. It can be observed that for the optimal *k*, the percentage is always below 40%. Hence, the time needed to compute the *k*NN*s* and detect errors is less than 1.4x of the unprotected implementation (so significantly less than the 2x of TBMR). Moreover, for the majority of the datasets, the percentage is below 10%, so corresponding to a factor of only 1.1x. These results show that the proposed VM scheme can

TABLE 4
PERCENTAGE OF ELEMENTS FOR WHICH RE-COMPUTATION IS NEEDED IN THE PROPOSED SCHEME (RESULTS FOR THE OPTIMAL K ARE HIGHLIGHTED IN BOLD AND UNDERLINED)

k	Pima Indian diabetes	EEG eye state	LSVT voice rehabilitation	Sonar	Iono-sphere	Electrical grid stability simulated data	Banknote authentication	Qualitative bankruptcy	Phishing websites	Climate model simulation crashes
3	44.24%	**35.11%**	42.58%	**19.90%**	13.52%	19.90%	0	**2.69%**	13.49%	17.80%
5	32.29%	24.48%	28.97%	12.84%	7.80%	12.47%	0	2.23%	**9.68%**	**8.37%**
7	26.29%	19.30%	26.24%	14.08%	5.69%	8.95%	**0**	4.33%	7.43%	4.36%
9	17.34%	16.25%	**23.18%**	14.79%	5.99%	7.38%	0.10%	4.80%	5.21%	2.41%
11	13.45%	14.16%	15.63%	14.45%	**8.31%**	5.81%	0	5.07%	5.28%	2.46%
13	13.18%	12.35%	13.55%	14.68%	5.78%	5.14%	0	7.24%	4.76%	1.28%
15	11.86%	11.49%	16.34%	16.02%	3.13%	4.94%	0	6.84%	3.63%	0.22%
17	9.14%	11.31%	20.39%	12.02%	4.00%	4.39%	0	9.88%	3.53%	0.69%
19	**9.25%**	10.30%	17.79%	10.02%	3.55%	**3.50%**	0	11.48%	3.74%	0.04%

TABLE 5
PERCENTAGE OF ERRORS (ON DIFFERENT BIT POSITION OF THE COMPUTED DISTANCES) THAT MODIFY THE CLASSIFICATION RESULT IN THE UNPROTECTED *k*NNS SCHEME IN THE CASE OF OPTIMAL *K* (LARGEST VALUES ARE HIGHLIGHTED IN BOLD AND UNDERLINED)

#bit	Pima Indian diabetes	EEG eye state	LSVT voice rehabilitation	Sonar	Ionosphere	Electrical grid stability simulated data	Banknote authentication	Qualitative bankruptcy	Phishing websites	Climate model simulation crashes
15	0.08%	0	0.50%	0.15%	0.14%	0	0	0	0	0.05%
14	0.09%	0	0.61%	0.19%	0.11%	0	0	0	0	0.02%
13	0.07%	0	0.76%	0.08%	0.07%	0	0	0	0	0.05%
12	0.09%	0	0.71%	0.10%	0.05%	0	0	0	0	0.01%
11	0.11%	0	2.79%	0.23%	0.18%	0	0	0	0	0.02%
10	0.25%	0	**3.76%**	**8.21%**	**1.90%**	0	0	0	**1.49%**	0.02%
9	**1.81%**	0.76%	2.18%	1.48%	1.27%	**0.97%**	0	**0.77%**	**1.49%**	**2.20%**
8	1.14%	5.59%	1.18%	1.29%	0.86%	0.14%	0	0.51%	0.27%	0.38%
7	1.06%	**6.79%**	1.03%	0.63%	0.22%	0.04%	0	0.05%	0	0.19%
6	0.09%	2.29%	0.24%	0.13%	0.08%	0.01%	0	0.16%	0	0.06%
5	0.04%	0.65%	0.05%	0.08%	0.19%	0	0	0	0	0.01%
4	0.01%	0.13%	0	0.03%	0.03%	0	0	0.03%	0	0.01%
3	0.02%	0.07%	0	0.03%	0.01%	0	0	0	0	0.01%
2	0	0.04%	0	0	0	0	0	0	0	0.01%
1	0	0.04%	0	0	0	0	0	0	0	0.01%
0	0	0	0	0	0	0	0	0	0	0

significantly reduce the execution time (and thus power consumption) of *k*NNs. The percentages in Table 4 are larger than those in Table 3 because even when there is no voting margin, many errors do not change the classification result. For example, an error on an element that belongs to the majority class and is not a *k*NN, does not affect the classification result.

The savings of the proposed scheme increase with *k* for most datasets. This is intuitive, because in general the larger the number of votes, the larger the probability to have VM. An exception is the Banknote bankruptcy data set for which the percentage of re-computations increases with *k*.

To further study the effects of errors on the *k*NNs algorithm a third experiment was conducted. In this case, the inserted errors model a single bit error and thus its magnitude depends on the bit position. Therefore, it was assumed that the distances were quantized with 16 bits and errors that correspond to the magnitude of an error on each of those bits, were inserted.

The results are shown in Table 5 for the optimal *k* value of each dataset. Errors on the least significant bits (LSBs) tend to have limited impact on the classification result. This is expected because they can only slightly move the elements and are thus less likely to change the *k*NNs. Differently from pure arithmetic computations [22], the most significant bits (MSBs) are not the ones that have a larger impact on the result. This can be explained going back to the two types of errors discussed in the previous section. Errors of type 1 are those on which an element that should have been in the set of *k*NNs, is moved away. Errors on the MSBs are more likely to cause this type of error (because in the datasets considered in this paper, most computed distances have values below 32 and thus the MSBs are always

0, so only causing errors of type 1). The second type error occurs when an element that should not be in the set of *k*NNs is moved close to the element being classified due to the error. In this case, this can occur when the original distance is similar to the change introduced by the bit error such that the element is moved close to the element being classified. The probability of this occurrence depends on the distribution of the distances as explained next.

When comparing the two error types, the first one can only occur during the computation of the *k* distances for the elements that are the *k*NNs for the element being classified. The second type of error can occur during the computation of the distances of the remaining elements. As in most cases the training set is larger than *k*, so the second error type is more likely. Moreover, as discussed in the previous section, an error of the first type implies that the vote is taken among the *k*+1 NNs without the element affected by the error. This vote is likely to be the same as in the error free one. Therefore, errors of type 2 tend to have a dominant impact on the errors for the classification result. This is the reason for errors on the more significant bits not having the largest impact on the classification outcome.

Consider next errors of type 2 in more detail. To affect the classification result, the error must bring the erroneous element close to the element being classified, thus the bit error should change a "1" to "0" in the unsigned binary representation. This includes two cases: a bit error on one of the integer bits of the computed distance, and one of the fractional bits. Consider the first case, starting from the analysis of the upper bound of the distances of *k*NNs. The Sonar dataset is taken as an example; Table 6 shows the distribution of the k^{th} (*k*=3 in the Sonar dataset) nearest neighbor distance (only for the integer part) across all test elements that can be classified incorrectly due to a type 2 error. As per the proposed VM scheme, these elements can be identified by checking if there is no VM and the classification of the k^{th} element is a majority (there are 11 such elements in the Sonar dataset). From Table 6, the largest distances of *k*NNs have values of 7, 8 and 9 (i.e., the 9 bits, from 15^{th} to 7^{th} bits, for the integer part of the distance are "000000111", "000001000" and "000001001" in binary representation). Therefore, elements that have a distance within the range of 8 to 15 (i.e., "000001000" to "000001111") are likely to be moved into the *k*NNs by a "1" to "0" bit error on the 10^{th} bit (in bold and underlined, which will cause the distance to be smaller than the k^{th} nearest neighbor distance), and those that have a distance within the range of 16 to 25 (i.e., "0000**1**0000" to "0000**1**1001") are most likely moved into the *k*NNs by a "1" to "0" bit error on the 11^{th} bit, and so on. Figure 6 shows the distribution of distances from the 11 test elements that can fail to all training elements that are not *k*NNs. The largest value has an integer part of 19 and most of the distances have an integer part within the range of 8 to 15; therefore, a bit error on the 10^{th} bit of the distance has the largest effect on the classification result, which is consistent with the results in Table 5 (i.e. 8.21%).

TABLE 6
DISTRIBUTION OF THE K^{TH} NEAREST NEIGHBOR DISTANCES (FOR THE INTEGER PART IN DECIMAL) BETWEEN ALL TEST ELEMENTS (THAT MAY CAUSE ERRORS OF TYPE 2) AND THE TRAINING ELEMENTS (THAT DO NOT BELONG TO THE *K*NNS, WHERE *K*=3) IN THE SONAR DATASET

Distance	Frequency
5	1
6	2
7	3
8	2
9	3

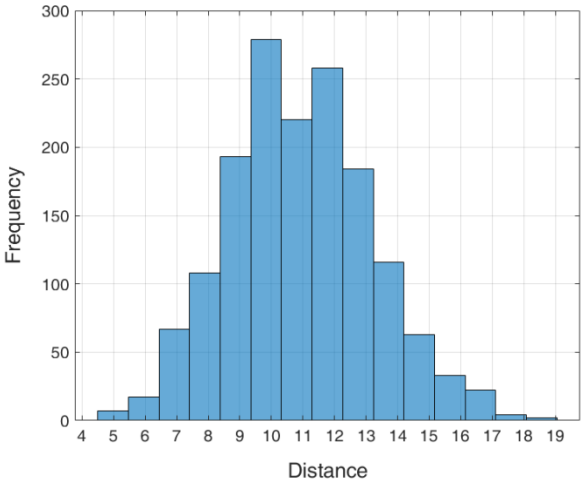

Figure 6 Distribution of the distances (in decimal) between all test elements and the training elements (that do not belong to the *k*NNs, where *k*=3) in the Sonar dataset

Consider the second case, i.e. the bit error occurs on the fractional bits. In this case, a classification failure can only be caused by the elements that are near to *k*NNs in the error free case because the distance changes slightly, so the impact is low. Moreover, a bit error on the MSBs of the decimal bits has a bigger impact than the LSBs (with an error on these bits the elements are unlikely to move). This consideration is also consistent with the simulation results of Table 5.

Therefore, the effects of a bit error are dependent on the dataset and the distribution of distances. This is shown in Table 5 in which for some datasets, the most critical bit is nine or ten. An interesting point is that the dependency of the impact of the error on the distance distribution can be exploited to design schemes that propagate errors to the upper bits so that they introduce a large error in the distance as for example in [36] for a different application.

An interesting observation is that for some datasets (for example the EEG eye state) increasing the value of *k* shows a significant reduction in re-computations (from 35.11% to 10.30% in Table 4), while classification accuracy is only slightly worse (as shown in Fig. 5). Therefore, in some cases, selecting a near optimal *k* instead of the optimal value can be of interest to improve performance at the cost of a minor accuracy loss in classification.

Table 7 summarizes the evaluation results for the comparison of the three options considered. Obviously, an unprotected implementation has the lowest (none) computational overhead. However, it can result in classification

errors of up to 4% depending on the dataset. Instead, both TBMR and VM can effectively protect against soft errors in a distance computation and ensure that they do not affect the classification result. However, this comes at a cost of 2x for TBMR. The proposed VM scheme is able to reduce such cost to less than 1.4x in all the datasets considered. So, at least 60% of the additional computation introduced by TBMR is saved. This clearly illustrates the benefits of the proposed scheme.

TABLE 7
COMPARISON OF DIFFERENT KNN PROTECTIONS

Scheme	Impact on classification accuracy	Computational overhead
Unprotected	0-4%	1x
TBMR	None	2x
Proposed VM	None	< 1.4x

5 CONCLUSION AND FUTURE WORK

In this paper, Voting Margin (VM) has been proposed; VM, a technique to protect *k* Nearest Neighbors (*k*NNs) classifiers against errors in arithmetic operations, has been presented and evaluated. The proposed scheme exploits the intrinsic redundancy of *k*NNs to drastically reduce the number of re-computations needed to detect errors. This is achieved by noting that when voting among the *k* nearest neighbors has a large majority, an error in one of the voters cannot change the result. This observation has been refined and extended in the proposed VM scheme to also avoid recomputations in some cases in which the majority vote is tight.

The VM scheme has been evaluated using publicly available datasets that cover a wide range of applications, number of features and dataset sizes. The results show that in all configurations tested it can significantly reduce the protection overhead. Specifically, for the optimal *k*, the proposed scheme is able to reduce the overhead by at least 60%.

Although this paper has focused on a particular classification algorithm (*k*NNs), we think that the properties of classification algorithms and more generally ML algorithms can be efficiently used to provide protection against errors at a lower cost than existing techniques. This opens a new and interesting area for future research on algorithmic-based error-tolerance. In particular, some of the techniques presented in this manuscript could be applicable to classification ensembles that use several classifiers and vote among them to obtain a better result [34]. In the case, when there is a VM, then there would be no need to recompute the classification result to detect errors. Moreover if there is no VM, recomputing the classifiers that vote with the majority should be sufficient to detect errors that can affect the final result. The extension of the proposed ideas to classification ensembles seems therefore an interesting area for future work.

ACKNOWLEDGMENT

Pedro Reviriego and José Alberto Hernández would like to acknowledge the support of the TEXEO project TEC2016-80339-R funded by the Spanish Ministry of Economy and Competitivity and of the Madrid Community research project TAPIR-CM grant no. P2018/TCS-4496.

REFERENCES

[1] T. Hastie, R. Tibshirani, and J. Friedman, "The Elements of Statistical Learning". *Springer Series in Statistics Springer New York Inc., New York, NY, USA,* 2001.

[2] I. Goodfellow, Y. Bengio, A. Courville, "Deep Learning" *The MIT Press,* 2016.

[3] Q. Hu, D. Yu, and Z. Xie, "Neighborhood Classifiers," in *Expert Systems with Applications,* vol. 34, pp. 866-876, Feb. 2008.

[4] E. Fix and J. Hodges, Jr., "Discriminatory Analysis: NonparaMetric Discrimination: Consistency Properties," USAF School of Aviation Medicine, San Antonio, TX, USA, Tech. Rep. no. 4, Feb. 1951.

[5] N. Kanekawa, E. H. Ibe, T. Suga, and Y Uematsu, "Dependability in Electronic Systems: Mitigation of Hardware Failures, Soft Errors, and Electro-Magnetic Disturbances," New York, NY, USA: Springer-Verlag, Nov. 2010.

[6] "ISO 26262: Road vehicles — Functional safety." International Standard. 2011.

[7] C- Yung-Chang, et al. "Assessing Automotive Functional Safety Microprocessor with ISO 26262 Hardware Requirements," in *IEEE International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pp.1-4, Apr. 2014.

[8] M. Nicolaidis, "Design for Soft Error Mitigation," in *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 405–418, Sep. 2005.

[9] S. J. Wang and N. K. Jha, "Algorithm-based Fault Tolerance for FFT Networks," in *IEEE Transactions on Computers*, vol. 43, no. 7, pp. 849-854, Jul. 1994.

[10] B. Shim and N. R. Shanbhag, "Energy-Efficient Soft Error-Tolerant Digital Signal Processing," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 4, pp. 336-348, Apr. 2006.

[11] S. Liu, G. Sorrenti, P. Reviriego, F. Casini, J. A. Maestro and M. Alderighi, "Increasing Reliability of FPGA-Based Adaptive Equalizers in the Presence of Single Event Upsets," in *IEEE Transactions on Nuclear Science*, vol. 58, no. 3, pp. 1072-1077, Jun. 2011.

[12] Y. Zhao, W. Li and P. Shi, "A Real-Time Collision Avoidance Learning System for Unmanned Surface Vessels," *Neurocomputing*, vol. 182, pp. 255-266, Mar. 2016.

[13] A. L. Buczak, E. Guven, "A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection", in *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153-1176, Oct. 2015.

[14] I. Cara, E. d. Felder, "Classification for Safety-Critical Car-Cyclist Scenarios Using Machine Learning", in *IEEE 18th International Conference on Intelligent Transportation Systems*, pp. 1995-2000, Sep. 2015.

[15] P. N. Yianilos, "Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces," *in SODA*, pp. 311–321, Jan. 1993.

[16] G. Song, J. Rochas, L. El Beze, F. Huet, and F. Magoules, "K nearest neighbour joins for big data on MapReduce: A theoretical and experimental analysis," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 9, pp. 2376–2392, May. 2016.

[17] V. Garcia, E. Debreuve and M. Barlaud, "Fast K Nearest Neighbor Search Using GPU," *in Proc of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1-6, Jun. 2008.

[18] E. S. Manolakos and I. Stamoulias, "IP-cores Design for the KNN Classifier," *in Proc. of the IEEE International Symposium on Circuits and Systems*, pp. 4133-4136, May 2010.

[19] N. Attaran, A. Puranik, J. Brooks and T. Mohsenin, "Embedded Low-Power Processor for Personalized Stress Detection," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 12, pp. 2032-2036, Dec. 2018.

[20] S. Lin and D. J. Costello, "Error Control Coding," 2nd ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 2004.

[21] P. Reviriego, S. Pontarelli, J.A. Maestro, M. Ottavi, "A Method to Construct Low Delay Single Error Correction (SEC) Codes for Protecting Data Bits Only," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 3, pp. 479-483, Mar. 2013.

[22] K. Chen, L. Chen, P. Reviriego and F. Lombardi, "Efficient Implementations of Reduced Precision Redundancy (RPR) Multiply and Accumulate (MAC)," in *IEEE Transactions on Computers*, vol. 68, no. 5, pp. 784-790, 1 May 2019.

[23] I. Alzaher-Noufal and M. Nicolaidis, "A CAD Framework for Generating Self-Checking Multipliers Based on Residue Codes", *in Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, Mar.1999.

[24] M. Nicolaidis and R. O. Duarte, "Fault-Secure Parity Prediction Booth Multipliers", in *IEEE Design & Test of Computers*, vol. 16, no. 3, pp. 90-101, 1999.

[25] I. Oz and S. Arslan, "A Survey on Multithreading Alternatives for Soft Error Fault Tolerance", in *ACM Computing Surveys*, vol. 52, pp. 1, 2019.

[26] Y. Ma and H. Zhou, "Efficient Transient-Fault Tolerance for Multithreaded Processors Using Dual-Thread Execution," *in Proceedings of the International Conference on Computer Design*, San Jose, CA, 2006, pp. 120-126.

[27] D. Dua and C. Graff "UCI Machine Learning Repository", *Irvine, CA: University of California, School of Information and Computer Science*, 2019.

[28] A. Tsanas, M.A. Little, C. Fox and L.O. Ramig, "Objective Automatic Assessment of Rehabilitative Speech Treatment in Parkinson's Disease," in *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 22, no. 1, pp. 181-90, Dec. 2013.

[29] R.P. Gorman, T.J. Sejnowski, "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets. *Neural networks*, vol. 1, no. 1, pp.75-89, Jan. 1988.

[30] V.G. Sigillito, S.P. Wing, L.V. Hutton, K.B. Baker, "Classification of Radar Returns from the Ionosphere Using Neural Networks", *Johns Hopkins APL Technical Digest*, vol. 10, no. 3, pp.262-266, Jul. 1989.

[31] V. Arzamasov, K. Bohm, P. Jochem, "Towards Concise Models of Grid Stability", in *IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (Smart Grid Comm)*, pp. 1-6, Oct. 2018.

[32] M.J. Kim, I. Han, "The Discovery of Experts' Decision Rules from Qualitative Bankruptcy Data Using Genetic Algorithms. *Expert Systems with Applications*, vol. 25, no. 4, pp. 637-646, Nov. 2003.

[33] R.M. Mohammad, F. Thabtah, L. McCluskey, "Intelligent Rule-based Phishing Websites Classification", *IET Information Security*, vol. 8, no. 3, pp. 153-160, Mar. 2014.

[34] D. D. Lucas, R. Klein, J. Tannahill, D. Ivanova, S. Brandon, D. Domyancic and Y. Zhang, "Failure Analysis of Parameter-Induced Simulation Crashes in Climate Models", in *Geosci. Model Dev. Discuss.*, vol. 6, pp. 585-623, Aug. 2013.

[35] M. Khun, K. Johnson, "Applied predictive modeling", *Springer* 2013.

[36] A. Sánchez-Macián, L. A. Aranda, P. Reviriego, V. Kiani and J. A. Maestro, "Enhancing Instruction TLB Resilience to Soft Errors," in *IEEE Transactions on Computers*, vol. 68, no. 2, pp. 214-224, 1 Feb. 2019.

Shanshan Liu (M'19) received the M.Sc. and Ph.D. degrees in microelectronics and solid-state electronics from Harbin Institute of Technology, Harbin, China, in 2012 and 2018, respectively. She is currently a Post-doctoral researcher with the Department of Electrical and Computer Engineering, Northeastern University, Boston, US.

Her current research interests include fault tolerant design in high performance computer systems.

Pedro Reviriego (M'04-SM'15) received the M.Sc. and Ph.D. degrees in telecommunications engineering from the Technical University of Madrid, Madrid, Spain, in 1994 and 1997, respectively. From 1997 to 2000, he was an R&D Engineer with Teldat, Madrid, working on router implementation. In 2000, he joined Massana to work on the development of 1000BaseT transceivers. From 2004 to 2007, he was a Distinguished Member of Technical Staff with the LSI Corporation, working on the development of Ethernet transceivers. From 2007 to 2018 he was with Nebrija University. He is currently with Universidad Carlos III de Madrid working on high speed packet processing and fault tolerant electronics.

José Alberto Hernández (M'19) received the BEng degree in Telecommunications from Univ. Carlos III de Madrid (Spain) in 2002 and the PhD degree in Computer Science from Loughborough University (UK) in 2005. At present, he is a senior lecturer at Univ. Carlos III de Madrid where he combines teaching and research in analysis of communication protocols and networks, network optimization and machine learning. Dr. Hernández has published over 100 scientific articles and participated in several national and European projects in the areas of optical WDM networks, optical transport for 5G and Machine Learning applications in communications.

Fabrizio Lombardi (M'81-SM'02-F'09) received the B.Sc. degree (Hons.) in electronic engineering from the University of Essex, U.K., in 1977, the master's degree in microwaves and modern optics and the Diploma degree in microwave engineering from the Microwave Research Unit, University College London, in 1978, and the Ph.D. degree from the University of London in 1982. He is currently the International Test Conference (ITC) Endowed Chair Professorship with Northeastern University, Boston, USA. His research interests are bio-inspired and nano manufacturing/computing, VLSI design, testing, and fault/defect tolerance of digital systems. He has extensively published in these areas and coauthored/edited seven books. He was the Editor-In-Chief of the IEEE TRANSACTIONS ON COMPUTERS from 2007 to 2010 and the inaugural Editor-in-Chief of the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING from 2013 to 2017. He is the Editor-in-Chief of the IEEE TRANSACTIONS ON NANOTECHNOLOGY.