Error-Tolerant Computation for Voting Classifiers with Multiple Classes

Error-Tolerant Computation for Voting Classifiers with Multiple Classes

Shanshan Liu, *Member, IEEE*, Pedro Reviriego, *Senior Member, IEEE,* Paolo Montuschi, *Fellow, IEEE* and Fabrizio Lombardi, *Fellow, IEEE*

***Abstract*—In supervised learning, labeled data are provided as inputs and then learning is used to classify new observations. Error tolerance should be guaranteed for classifiers when they are employed in critical applications. A widely used type of classifiers is based on voting among instances (referred to as single voter classifiers) or multiple voters (referred to as ensemble classifiers). When the classifiers are implemented on a processor, Time-Based Modular Redundancy (TBMR) techniques are often used for protection due to the inflexibility of the hardware. In TBMR, any single error can be handled at the cost of additional computing either once for detection or twice for correction after detection; however, this technique increases the computation overhead by at least 100%. The Voting Margin (VM) scheme has recently been proposed to reduce the computation overhead of TBMR, but this scheme has only been utilized for *k* Nearest Neighbors (*k*NNs) classifiers with two classes. In this paper, the VM scheme is extended to multiple classes, as well as other voting classifiers by exploiting the intrinsic robustness of the algorithms. *k*NNs (that is a single voter classifier) and Random Forest (RF) (that is an ensemble classifier) are considered to evaluate the proposed scheme. Using multiple datasets, the results show that the proposed scheme significantly reduces the computation overhead by more than 70% for *k*NNs with good classification accuracy and by more than 90% for RF in all cases. However, when extended to multiple classes, the VM scheme for *k*NNs is not efficient for some datasets. In this paper, a new protection scheme referred to as *k*+1 NNs is presented as an alternative option to provide efficient protection in those scenarios. In the new scheme, the computation overhead can be further reduced at the cost of allowing a very low percentage of errors that can modify the classification outcome.**

***Index Terms*—Machine learning, voting classifier, error tolerance, *k* nearest neighbors, random forest.**

Manuscript received May 18, revised August 6, and accepted September 17, 2020. The research was supported by the ACHILLES project PID2019-104207RB-I00 and the Go2Edge network RED2018-102585-T funded by the Spanish Ministry of Economy and Competitivity, and by the Department of Research and Innovation of Madrid Regional Authority, in the EMPATIA-CM research project (Reference no. Y2018/TCS-5046), and by NSF under CCF-1953961 and 1812467 grants.

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 is with Universidad Carlos III de Madrid, Av. De la Universidad 30, Leganés, Madrid, Spain (email: revirieg@it.uc3m.es).

P. Montuschi is with Politecnico di Torino, Dipartimento di Automatica e Informatica, 10129 Torino, Italy (email: paolo.montuschi@polito.it).

I. Introduction

Machine Learning (ML) is used to analyze data for a wide range of applications, such as medicine, biology, finance, vehicle, communications or daily life [1]-[6]. By using specific algorithms, ML is the process of guiding a machine (computer) to construct a reasonable model based on a set of known data and use such model to judge (analyze) new data. When ML targets at predicting a valued result for the new data, the process is known as regression; whereas if the object is a discrete result, the process is known as classification [7], [8]. Some of the simplest and yet powerful algorithms for classification are based on voting. Once the features of the new element are input in the model, the voting classifiers predict the class for the new element by taking a majority voting among the instances (in a single voter classifiers, such as *k* Nearest Neighbors [9], [10]) or multiple voters (in ensemble classifiers, such as Random Forest [11], [12]).

ML classifiers are commonly implemented for computation either on a CPU or embedded microprocessor; these chips are prone to errors, such as soft errors due to radiation effects [13], [14]. Errors that occur in the computation process or in storage for training elements can modify the classification result. Even if errors are isolated events [13], they would not be acceptable if the classifiers are part of safety and/or critical systems [15], [16], because an error can cause a functional failure. For example, in autonomous vehicle applications, errors in the ML classifiers that are used to determine whether an object is a pedestrian or in the ML-based driver fingerprinting extraction for cash trucks [17] can have a dramatic consequence; thus error-tolerance and reliability needs to be assured [18], [19]. When algorithms for ML computation are implemented in specialized hardware units (such as arithmetic circuits), errors can be handled by employing error-tolerant techniques (such as Triple Modular Redundancy [20] or Reduced Precision Redundancy [21]), or using Error Correction Codes (that were first applied in communication [22]) for memory (storing training data) [23], [24].

However, when the ML algorithms are run on a processor, hardware implementations are difficult to modify without resorting to expensive redundancy utilization; in this case, Time-Based Modular Redundancy (TBMR) is often used to detect and correct errors. By running the computation twice, an error can be detected if the two results are mismatched. Then if

the error is detected, computation is run for a third time and the error can be corrected by majority voting among the three results. The TBMR technique can correct all single errors in arithmetic operations, but it incurs in a large computation overhead, costing approximately 2x in terms of execution time and power consumption; such approach may not be acceptable when the ML algorithms are employed in resource-limited platforms, like smartphones, low-power wearable and Internet-of Things (IoT) devices [25], [26]. Recently, a different protection scheme referred to as Voting Margin (VM) has been proposed [27] to deal with single errors for *k*NNs classifiers; VM exploits algorithmic features. The VM scheme is based on the observation that when voting among the *k*NNs has a large majority (i.e. a margin exists), then an error in one of the computed distances cannot change the classification result. Compared with TBMR, the VM scheme can drastically decrease the recomputation overhead, because when there is a VM, the classifier is always reliable (i.e., we do not need to perform the recomputation to detect/correct errors). However, the current VM technique is designed for *k*NNs classifiers with only two classes. In many cases, classification among multiple classes is needed [28], [29]; therefore, efficient error-tolerant protection techniques for those classifiers are needed.

In this paper, we address the limitations of VM by extending it to voting classifiers with multiple classes. By considering *k*NNs and Random Forest classifiers as examples, the scheme is illustrated and evaluated for both single voter classifiers and ensemble classifiers. Moreover, a new scheme (referred to as the *k*+1 NNs scheme) is also proposed as alternative for *k*NNs classifiers to improve the VM scheme of [27]. Software simulation results show that the proposed schemes achieve a significant reduction in terms of computation overhead comparing with existing techniques such as TBMR.

The remaining part of this paper is organized as follows. In Section II, the algorithms and implementations of different voting classifiers (including single voter classifiers and ensemble classifiers) are introduced; the Voting Margin protection technique for *k*NNs classifiers with only two classes [27] is also illustrated. Section III presents the proposed error-tolerant schemes for voting classifiers with multiple classes, which are evaluated in Section IV. Discussion of the different schemes is given in Section V. Finally, the paper ends in Section VI with the conclusion.

II. Preliminaries

This section first provides a brief description of voting classifiers including *k*NNs and Random Forests, and their implementations. Then, the impact of errors that occur when voting is performed, is discussed. Finally, the Voting Margin protection technique for classifiers with two classes proposed in [27] is reviewed.

A. Voting Classifiers and Implementation

Voting classifiers can be either implemented based on a single voter among different instances/elements to predict the class (this is accomplished by voting their classes), or based on an ensemble method that combines a set of voters and voting among them again to improve classification performance.

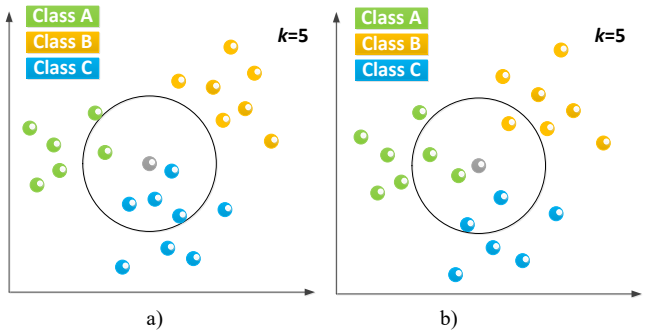

Figure 1 Illustration of the *k*NNs algorithm with three classes. The green, yellow and blue elements are stored with their class. The grey element is the one being classified. As $k = 5$, first the five elements closest to it are identified (shown in the solid circle). Then, a vote is taken among them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

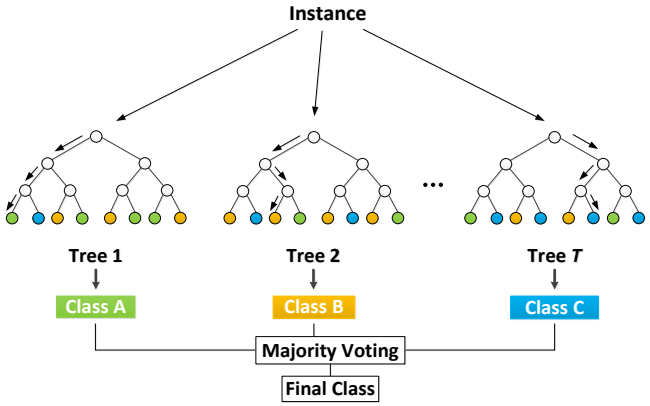

Figure 2 Illustration of the Random Forest algorithm, where *T* is the number of decision trees and the topmost node in a tree is the root node.

*k*NNs are one of the simplest yet powerful classification algorithms [9]; the algorithm's hyperparameter (i.e., the value of *k* corresponding to the highest classification accuracy) is usually obtained by using the well-known 10-fold cross-validation methodology [30]. Once the optimal *k* is determined, the features and labels of the training elements are stored and used to predict the class for a new element. To do so, the distances from all training elements to the element being classified are first computed and the set of *k*NNs is selected. Then, majority voting is executed among the *k* classes of the neighbors to establish the predicted class. A special case occurs when there are equal votes with multiple classes, leading to a tie. When there are only two classes, to break ties the value of *k* can be selected as an odd number, so that the number of elements with each class can be simply compared with the threshold (i.e., ceil((*k*+1)/2), which is (*k*+1)/2 when *k* is odd) to find the majority.

When there are at least three classes, voting is more complex; it includes two cases, which are illustrated in the example shown in Figure 1. Once the *k*NNs are selected, the number of elements with each class are compared with each other. The first case is that there is only one class with the largest number of neighbors, then it is used as the classification result. This is shown in Figure 1 a), there are one, zero and four neighbors in Class A, B and C, thus the predicted result is Class C. If there

are two or more classes voted by the same largest number of neighbors (i.e., there is a tie), we cannot provide an outcome by just using voting. In this case, other methods (such as selecting the nearest neighbor or randomly) are generally used to break ties and determine the classification result. For example, in Figure 1 b), two neighbors belong to Class A, one neighbor to Class B and two neighbors to Class C. However, as the nearest neighbor of the ones in A or C is green, then Class A will be output as predicted result if we select the class of the nearest neighbor to break a tie.

Ensemble classifiers consist of multiple classifiers that act as single voters; each of them is based on a random sample of the training elements and different feature importance. The classification procedure for each voter is the same as for the classifiers introduced above. A widely used ensemble classifier is Random Forest (RF), which utilizes T decision trees and is illustrated in Figure 2. Once the prediction model is trained, labels are stored on the leaf nodes of the decision trees while the decision thresholds on the features are stored in the root and the internal (non-leaf) nodes. The decision tree is a flow-chart-like structure, in which each internal node denotes a test on an attribute, each branch represents the outcome of a test, and each leaf node holds a class label.

For classification of an incoming element, its class is predicted firstly by each tree (for example it is predicted by Tree 1 as Class A in Figure 2). Then the final result is obtained by simply establishing a majority voting among the results of those voters for a second round. This process accomplishes a better performance of classification but incurring in a larger execution overhead than for a single tree.

B. Errors on Voting Classifiers

Errors that occur during the process of classification for an incoming element may change the voting result and thus modify the end result as outcome. For example, in the kNNs classifier, an incorrect distance from a training element to the incoming element to be classified could be calculated if there is an error in the distance computation process, likely changing the set of kNNs. In this case, voting is done among incorrect kNNs and may generate a different result. Therefore, depending on the impact on the kNNs, errors in a distance computation include the following types.

Type 1: An element that was in the set of kNNs is moved due to the error, but it is still in the set. Only the position has been changed (nearer or further away from the element being classified).

Type 2: An element that should be in the set of kNNs, has a larger distance as result of the error and thus, it is no longer in the set, the k+1th NN is in the set instead.

Type 3: An element that should not be in the set of kNNs has a smaller distance as result of the error that places it in the set. The k^{th} NN is therefore no longer in the set.

Type 4: An element that was not in the set of kNNs, is moved, but it does not change the kNNs for the element being classified and thus, it has no impact on the result.

The first three types of error can modify the classification result; however, we can detect that in some cases the result would be still correct even in the presence of an error. This will be discussed in Section III and used as basis of the proposed schemes. Intuitively, type 3 errors have more significant impact on the classification result than type 1 and 2 errors because the size (i.e. the total number of training elements) of the dataset is significantly larger than the value of k.

For a Random Forest or an ensemble classifier, the analysis is simpler; single errors on any leaf or root of a decision tree may change the comparison path, resulting in a different class. This may affect voting among the trees and finally modify the classification result.

Since in many cases, errors do not happen frequently, it is usually assumed that there is a single error on a system at a given time. This applies also to radiation-induced soft errors. In the systems considered, the distance computation in a kNNs classifier or the computation for the trees in a Random Forest classifier account for the bulk of the implementation and therefore, single errors that affect a distance computation, or one leaf/root are considered as error model in this paper. As for other operations that are not computationally intensive (e.g., voting), the traditional TBMR technique can be employed as error-tolerant scheme because it does not require large computational resources.

C. Voting Margin Scheme for Two Classes

When there are only two classes for the elements in a dataset, majority voting is simply done by comparing the number of elements belonging to each class with the threshold. For example, in a kNNs classifier, the threshold for voting is given by ceil((k+1)/2). Considering k=5 (five voters) as an example, there are three possible voting scenarios:

1) All five neighbors agree on the same class;
2) Four neighbors vote for one class, while the fifth neighbor votes for the other class;
3) Three neighbors vote for one class, while the remaining two elements vote for the other class.

In the first (second) scenario, the classification result is always correct under a single error, because the number of elements in the majority class is at least four (three) even after

Algorithm 1 VM scheme used in kNNs with two classes

```
1: Compute distances then obtain the classification result R1
2: if VM >= 3
3:    Output R1
4: else if k+1th NN belongs to the majority class and kth NN
          belongs to the minority class
5:    Output R1
6: else
7:    Recompute distances then obtain R2
8:    if R1 = R2
9:       Output R1
10:   else
11:      Recompute distances then obtain R3
12:      Do majority voting among R1, R2 and R3
13:      Output the voting result
14:   end if
15: end if
```

the error, so still larger than (equal to) the threshold ceil((*k*+1)/2)=3.

Consider the Voting Margin (VM) as the difference between the numbers of NNs belonging to the majority class and the minority class when a single error cannot change the classification result (i.e., VM >= 3 in the case of two classes); when there is a VM, the number of elements with the majority class is larger than the voting threshold (i.e., at least equal to (*k*+1)/2+1, where *k* is odd). Therefore, for the first two possible voting scenarios, there is a VM because a single error can never change the voting result. However, in the third scenario, a single error may change the majority class to the minority. A possible case is that under a type 2 error discussed in Section II-B, the k+1th NN with the minority class moves in the set of *k*NNs; the other possible case is that under a type 3 error, the k^{th} NN with the majority class is replaced by a further element with the minority class. In these cases, the classification result is modified; therefore, this third scenario must be considered because there is no VM.

As per the above observations, the VM protection scheme has been proposed in [27] for *k*NNs classifiers, but only with two classes (as described in Algorithm 1). When a classification procedure starts, an initial classification result R1 is obtained once the distances are computed and the *k*NNs are selected. Then voting is checked. If there is VM, or no VM but the k+1th NN belongs to the majority class and the k^{th} NN belongs to the minority class, R1 is correct and so, it will be used as outcome; otherwise, the distances must be recomputed and a classification result R2 is then obtained based on voting among the updated *k*NNs. If R1 and R2 are a match, then we can safely provide either of them as a correct output. If there is a mismatch between R1 and R2, recomputation for a third time is required and a vote is required to obtain a correct result.

Compared with the traditional time-based modular redundancy scheme, the VM protection technique [27] significantly reduces the overhead to detect errors, because the recomputation operation is only needed in some cases. The VM scheme can also be extended to protect classifiers with multiple classes and also to ensemble classifiers. This is studied in the following section.

III. Proposed Schemes for Multiple Classes

This section first presents the Voting Margin (VM) scheme applicable to voting classifiers but with multiple classes; the *k* Nearest Neighbors (*k*NNs) and Random Forest (RF) classifiers are taken as examples to discuss the detailed implementation. In the second part, the *k*+1 NNs scheme is proposed as alternative to protect the *k*NNs classifier when VM introduces a large overhead.

A. Extended Voting Margin Scheme

The VM protection scheme [27] used for *k*NNs classifiers with two classes is modified, so that it can be applied also to voting classifiers with multiple classes. As discussed in Section II-C, when there are only two classes, VM refers to the case in which the number of elements with the majority class is larger than the voting threshold (i.e., at least equal to (*k*+1)/2+1 in a *k*NNs classifier, where *k* is odd); then a single error that may occur in a distance computation, can never change the classification, because it can only decrease the number of majority elements from (*k*+1)/2+1 to (*k*+1)/2 in the worst case, so having no impact on the voting result. The smallest value of VM can also be regarded as a gap of three between the numbers of elements with the majority class and the minority class (the number of majority elements is (*k*+1)/2+1 while for the minority it is *k*-((*k*+1)/2+1), hence the gap is three). In this case, the classifier is always reliable under single errors.

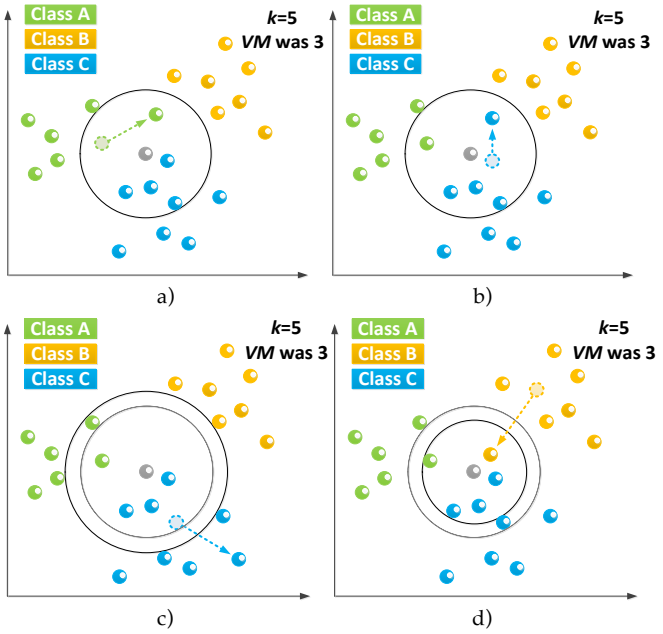

Figure 3 Illustration of *k*NNs having VM of three: a) and b) under an error of type 1; c) under an error of type 2; d) under an error of type 3.

The case of multiple classes with a VM of three also makes the classifier reliable following the same reasoning as for the case of two classes. Consider the example shown in Figure 1 to illustrate the use of VM in a *k*NNs classifier (in which there are three classes and *k*=5). From Figure 1 a) that the neighbors have a VM of 3, the following observations are applicable (when considering the cases of single errors with different types discussed previously in Section II-B that can modify the result one by one).

Under a type 1 error: Independently whether a Class A element (green) or a Class C element (blue) is moved by the error (as shown in Figure 3 a) and b)), the voting result is still Class C, because the number of neighbors within each class does not change.

Under a type 2 error: One neighbor is removed by the error from the set of *k*NNs and a Class A element that originally was in the k+1th NN, is included in such set. There are at least three blue elements (Figure 3 c)), so the voting result is still Class C, again it is not subject to modification.

Under a type 3 error: One element that was not in the set of *k*NNs, is included now, the k^{th} neighbor (blue) is then removed (Figure 3 d)). Independently whether the new neighbor belongs to a class, there will be at least three blue elements, leading to the same correct voting result.

As per the above discussion, it is always possible to correctly

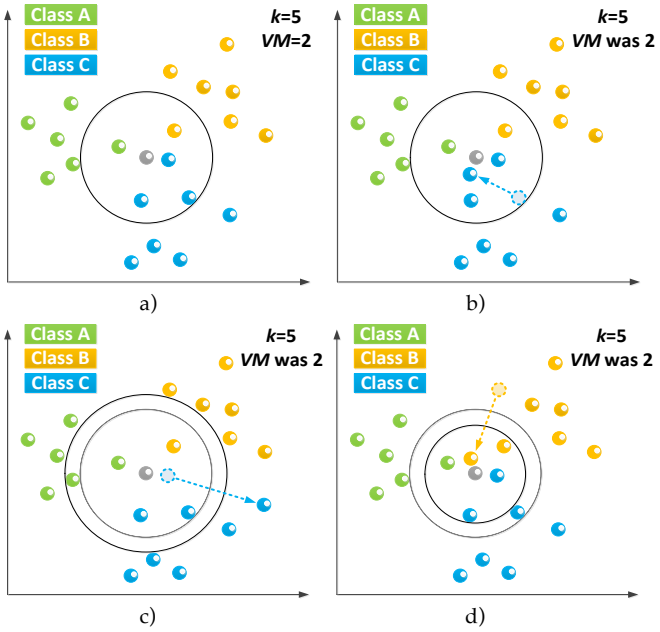

Figure 4 Illustration of the *k*NNs having VM of two: a) error free case; b) under an error of type 1; c) under an error of type 2; d) under an error of type 3.

output the classification result when the *k*NNs have a VM of three. The scheme is also applicable to other voting classifiers with the same voting approach as *k*NNs (such as Random Forest as shown in Figure 2), in which voting among the decision results of each tree is the same as among the classes of each neighbor of *k*NNs.

In the VM scheme used for *k*NNs classifiers with two classes [27], the correctness of the result can be established in some cases even if there is no VM (as discussed previously in Section II-C). When extending the VM scheme to *k*NNs classifiers with multiple classes, the vote can be also identified as correct in some cases even if there is no VM of three. Before describing the details of the proposed scheme, consider the impact of errors on classifiers with a tighter vote (having a VM of two) first.

Figure 4 shows an example of *k*NNs (three classes, VM of 2, and k=5), for the cases of error free and under different types of errors. The classification result is C in the error free case (shown in Figure4 a)). When a type 1 error occurs and moves a neighbor nearer to the element being classified (Figure 4 b)), there are still three neighbors agreeing on Class C, so the vote is unchanged. However, when a type 2 or 3 error occurs and replacing a Class C neighbor by Class A (Figure 4 c)) or B (Figure 4 d)), then there are two classes with the same largest number of neighbors in the set. In this case, the vote depends on which class the nearest neighbor belongs to (if we select the class of the nearer neighbor to break a tie). Therefore, the classification result may be affected; in Figure 4 c) the result changes from C to A, while in Figure 4 d) the result changes from C to B.

Therefore, when the *k*NNs have a VM of two, the correctness of the classification result can be determined by checking the following two constraints:

1) The class of the k+1th neighbor is majority;
2) The class of the k^{th} neighbor is minority.

Algorithm 2 VM scheme used in *k*NNs with multiple classes

```
Compute distances then obtain the classification result R1
if VM >= 3
    Output R1
else if VM = 2 and k+1th NN belongs to the majority class and
        kth NN belongs to the minority class
    Output R1
else
    Recompute distances then obtain R2
    if R1 = R2
        Output R1
    else
        Recompute distances then obtain R3
        Do majority voting among R1, R2 and R3
        Output the voting result
    end if
end if
```

The first constraint guarantees that a type 2 error (such as shown in Figure 4 c)) can only keep or increase the number of elements as majority class. The second constraint must be met to avoid the case in which a type 3 error brings a minority element in the majority and the original k^{th} majority neighbor is removed.

Once the voting result is determined to be reliable (there is a VM larger than two or equal to two with the above constraints), it can be provided as output. Otherwise, recomputation for the second time or even third time is required (like the VM scheme for two classes in [27]) to detect and correct the error. This entire process is described in Algorithm 2.

Moreover, when employing the VM scheme in Random Forest classifiers, recomputation is only needed for the trees agreeing on the majority class when there is a VM of two. This occurs because differently from *k*NNs (in which the voters may change due to an error) in a Random Forest, the voters do not change, i.e. the error can only change the vote of one of the voters. Therefore, if a majority of the voters with the same result is error free with VM two, then as an error could only reduce the number of voters in another class by one, the result is correct. The same reasoning applies in general to ensemble classifiers.

B. k+1 Nearest Neighbors Scheme

The extended VM scheme would efficiently reduce the recomputation operations for datasets with high accuracy; however, as shown in the next section, the reduction can be small for datasets with low accuracy, because in this case, voting tends to be very tight. In this scenario, VM is not capable to significantly reduce the computational overhead and thus alternative schemes are required. Therefore, we propose a second technique (referred to as the k+1 NNs scheme) for those datasets to approximately correct the error, so reducing the failure rate of classification at a small recomputation overhead.

In the k+1 NNs protection scheme, errors are detected by checking twice the k+1 nearest distances; this approximately guarantees the correct classification result by discarding the neighbor with a mismatched distance. The k+1 NNs scheme is

Algorithm 3 k+1 NNs scheme used in kNNs

```
1: Compute distances
2: Select k+1 NNs and keep their distances D1
3: Recompute distances for k+1 NNs and obtain D2
4: if D1 = D2
5:     Do majority voting among the classes of kNNs
6:     Output the classification result
7: else
8:     Recompute distance for the mismatched neighbor
9:     Do majority voting among the three computed distances
       for the mismatched neighbor
10:    Obtain the correct distances and the corresponding kNNs
11:    Do majority voting among the classes of kNNs
12:    Output the classification result
13:end if
```

accomplished by the following steps (also as described in Algorithm 3).

Step 1: When all distances from the training elements to the one being classified are computed and kNNs is selected as algorithm, then the k+1th NN and its distances are also stored.

Step 2: Distance computation is performed again but only for the k+1 nearest neighbors (instead of all training elements).

Step 3: Compare each of the k+1 distance pairs to check if there is an error. If they are the same (so there is no error changing the kNNs), a vote is taken among those kNNs to obtain the final classification result. If there is a mismatch, the error is detected, then the distance in error is recomputed for a third time and a vote is taken to obtain the correct value. Once completed, the elements with the k lower distances are selected to vote and the classification result is obtained.

This scheme fully protects the kNNs algorithm against type 3 errors, because in this case, the element brought by the error will be discarded, and so keeping the same kNNs. For type 2 errors, the scheme would not be effective, because one element of the original kNNs has been replaced by the k+1th NN; the original kNNs set cannot be recovered even if the distance for the mismatched neighbor is recomputed.

The third time of recomputation in step 3 of the algorithm is used to avoid the case in which a type 1 error (that exchanges two elements with the same classes) has no impact on the classification result and is discarded; then, the k+1th NN may change the classification result. Consider next an example of this case. Assume that there are three classes A, B, and C and k is 3, and the classes of kNNs are AAB. If a type 1 error occurs on the second A element (i.e. reducing the distance, so that it is smaller than the first A element), the updated kNNs are still AAB. Hence, there is no impact on the classification result among the kNNs (that is A). However, this error is detected when recomputation occurs for the second time; then, the second A element is discarded if no computation occurs for the third time. In this case, the final voting is modified to B if the class of the k+1th NN is B (the updated kNNs will be ABB).

The percentage of errors that modify the classification result in the unprotected kNNs algorithm (that includes errors of all types), should be reduced significantly by using the k+1 NNs scheme, because type 3 errors are the most common. This is evaluated in the following section.

IV. Evaluation

In this section, the proposed schemes have been evaluated using Matlab for kNNs and Random Forest (RF) classifiers by covering several widely used datasets with multiple classes taken from a public repository [31]. The following cases are evaluated as schemes.

Case 1: the extended VM scheme for the kNNs classifier;

Case 2: the k+1 NNs scheme for the kNNs classifier;

Case 3: the extended VM scheme for the RF classifier.

The selected datasets cover a wide range of applications and have a different number of elements, features, classes and classification performance when using the kNNs and RF algorithms. A brief description of the datasets is shown in Table I; the optimal parameters (k for kNNs and T for RF) corresponding to the top accuracy are determined by using the 10-fold cross-validation methodology on the training set (70% elements of the entire dataset). The classification accuracy is also plotted at different parameters in Figures 1 and 2 in the Appendix; when comparing the two figures, the RF classifier performs better than kNNs in most cases.

In the first evaluation, single computational errors are injected at distances for the unprotected kNNs classifiers. On the assumption that the same probability of error occurrence is applicable to every distance, the position of an error is selected randomly with a uniform distribution. The magnitude of an error is set randomly with a value from 0 to a maximum value of all computed distances, so it is able to deal with all possible errors that have an impact on the classification result (because an element with an incorrect distance larger than the maximum

TABLE I
Description of Considered Datasets

Dataset	Application	# Elements	# Features	# Classes	Using kNNs		Using RF	
					Optimal k	Top accuracy	Optimal T	Top accuracy
Iris	Botany	150	4	3	5	97.78%	80	97.78%
Student academic performance [32]	Pedagogics	131	20	3	7	41.03%	130	46.15%
Forest type mapping [33]	Ecology	325	27	4	9	77.32%	140	85.57%
Statlog (Vehicle silhouettes) [34]	Vehicle	846	18	4	11	77.17%	90	79.13%
Mice protein expression [35]	Biology	1080	80	8	3	99.38%	60	100.00%
CNAE-9	Finance	1080	856	9	7	84.57%	100	92.18%
Wine quality [36]	Life	4898	11	11	19	53.17%	120	66.67%
Thyroid disease	Medicine	7200	21	3	5	94.34%	110	99.72%
Nursery	Sociology	12960	8	5	17	96.40%	150	99.38%

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

k	Iris	Student academic performance	Forest type mapping	Statlog (Vehicle silhouettes)	Mice protein expression	CNAE-9	Wine quality	Thyroid disease	Nursery
3	1.93%	21.97%	7.68%	5.26%	**0.27%**	13.40%	7.35%	0.56%	2.98%
5	**1.33%**	16.23%	3.87%	4.65%	0.81%	7.89%	4.88%	**0.27%**	0.94%
7	0.80%	**15.13%**	6.01%	3.81%	0.84%	**6.38%**	3.65%	0.19%	0.82%
9	0.49%	18.03%	**2.82%**	4.70%	0.72%	5.58%	3.11%	0.12%	0.87%
11	0.44%	18.41%	3.10%	**4.42%**	0.77%	4.90%	2.86%	0.08%	0.74%
13	0.47%	16.10%	1.53%	2.94%	0.60%	3.29%	2.35%	0.03%	0.60%
15	0.42%	13.41%	1.93%	4.08%	0.26%	2.72%	2.04%	0.05%	0.57%
17	0	9.28%	1.24%	3.54%	0.70%	3.68%	1.83%	0.04%	**0.44%**
19	0	10.79%	1.20%	3.42%	1.23%	3.36%	**1.51%**	0.01%	0.49%
Average	0.65%	15.53%	3.26%	4.09%	0.69%	5.69%	3.29%	0.15%	0.94%

distance has the same effect as an element with the maximum distance). A single error on a distance is inserted when classifying an element and the process is repeated 10,000 times for each element. The percentages of errors that modify the classification result, are presented in Table II for *k*NNs at different *k*, as well as the average value among all *k* cases considered. The *k*NNs classifier can mask many of the errors, showing the intrinsic robustness of this algorithm. This behavior of *k*NNs shows that it can be used with no protection in some applications that can tolerate a few errors. However, for critical applications having a high accuracy requirement, additional protection is needed to completely correct, or at least reduce to a very small number the errors and their impact.

In the second evaluation, the proposed extended VM scheme is assessed. Errors are injected as in the first evaluation and are checked to establish the number of recomputations needed to detect errors (i.e., to check the cases in which the constrains presented in Section III-A are not met). Table III shows the percentage of elements for which recomputation of the *k*NNs is required. For datasets with good performance (i.e. with accuracy higher than 80%), the percentage is always below 30% for the optimal *k*. Hence, the recomputation overhead is

TABLE III
PERCENTAGE OF ELEMENTS FOR WHICH RECOMPUTATION IS NEEDED WHEN EMPLOYING THE EXTENDED VM SCHEME IN *K*NNS (RESULTS FOR THE OPTIMAL *K* ARE HIGHLIGHTED IN BOLD)

k	Iris	Student academic performance	Forest type mapping	Statlog (Vehicle silhouettes)	Mice protein expression	CNAE-9	Wine quality	Thyroid disease	Nursery
3	32.28%	76.56%	51.15%	62.31%	**23.13%**	68.86%	69.92%	11.59%	34.98%
5	**11.62%**	65.95%	34.54%	43.67%	5.15%	36.17%	58.78%	**3.74%**	14.19%
7	7.53%	**69.77%**	28.78%	42.79%	5.58%	**28.95%**	52.15%	2.12%	11.52%
9	5.56%	66.72%	**22.91%**	38.53%	5.92%	28.01%	44.20%	1.29%	10.56%
11	3.87%	68.15%	16.71%	**36.66%**	6.90%	25.82%	41.07%	0.91%	8.47%
13	2.22%	58.51%	11.57%	34.06%	6.51%	23.47%	35.29%	0.56%	7.15%
15	2.73%	52.28%	11.12%	33.08%	9.08%	22.59%	30.57%	0.51%	6.16%
17	3.04%	46.46%	9.33%	30.14%	10.25%	22.93%	27.35%	0.42%	**5.46%**
19	4.04%	40.31%	9.32%	28.64%	11.83%	21.59%	**25.07%**	0.37%	5.09%
Average	8.10%	60.52%	21.71%	38.88%	9.37%	30.93%	42.71%	2.25%	11.51%

TABLE IV
PERCENTAGE OF ERRORS THAT MODIFY THE CLASSIFICATION RESULT WHEN EMPLOYING THE PROPOSED *K*+1 NNS SCHEME IN *K*NNS (RESULTS FOR THE OPTIMAL *K* ARE HIGHLIGHTED IN BOLD)

k	Iris	Student academic performance	Forest type mapping	Statlog (Vehicle silhouettes)	Mice protein expression	CNAE-9	Wine quality	Thyroid disease	Nursery
3	0	0.49%	0.05%	0.06%	**0**	0.07%	0.02%	0	0
5	**0**	1.08%	0.05%	0.04%	0	0.04%	0.01%	**0.01%**	0
7	0	**0.87%**	0.06%	0.06%	0.01%	**0.06%**	0.01%	0	0
9	0	1.00%	**0.09%**	0.04%	0.01%	0.04%	0.02%	0	0
11	0.02%	0.90%	0.05%	**0.04%**	0.01%	0.04%	0.02%	0	0
13	0.02%	0.90%	0.02%	0.01%	0.01%	0.03%	0.02%	0	0
15	0	1.21%	0.05%	0.06%	0	0.02%	0.02%	0	0
17	0	0.56%	0.02%	0.06%	0.01%	0.03%	0.02%	0	**0**
19	0	1.33%	0.02%	0.06%	0.01%	0.04%	**0.02%**	0	0
Average	0	0.93%	0.05%	0.05%	0.01%	0.04%	0.02%	0	0

TABLE V
PERCENTAGE OF ELEMENTS THAT MODIFY THE CLASSIFICATION RESULT IN THE UNPROTECTED RANDOM FOREST SCHEME (RESULTS FOR THE OPTIMAL *T* IS HIGHLIGHTED IN BOLD)

# Decision trees	Iris	Student academic performance	Forest type mapping	Statlog (Vehicle silhouettes)	Mice protein expression	CNAE-9	Wine quality	Thyroid disease	Nursery
50	0.09%	0.41%	0.26%	0.72%	0	0	0.11%	0	0
60	0.02%	0.49%	0.24%	0.59%	**0**	0	0.09%	0	0
70	0.04%	0.38%	0.18%	0.43%	0	0	0.09%	0	0
80	**0.09%**	0.21%	0.18%	0.33%	0	0	0.09%	0	0
90	0.05%	0.28%	0.07%	**0.29%**	0	0	0.07%	0	0
100	0.02%	0.15%	0.05%	0.21%	0	**0**	0.05%	0	0
110	0.02%	0.15%	0.08%	0.28%	0	0	0.06%	**0**	0
120	0.07%	0.31%	0.05%	0.22%	0	0	**0.04%**	0	0
130	0.07%	**0.15%**	0.07%	0.09%	0	0	0.05%	0	0
140	0.02%	0.26%	**0.05%**	0.08%	0	0	0.04%	0	0
150	0.04%	0.23%	0.08%	0.07%	0	0	0.04%	0	**0**
Average	0.04%	0.22%	0.11%	0.30%	0	0	0.06%	0	0

TABLE VI
PERCENTAGE OF ELEMENTS FOR WHICH RECOMPUTATION IS NEEDED WHEN EMPLOYING THE RANDOM FOREST SCHEME (RESULTS FOR THE OPTIMAL *T* IS HIGHLIGHTED IN BOLD)

# Decision trees	Iris	Student academic performance	Forest type mapping	Statlog (Vehicle silhouettes)	Mice protein expression	CNAE-9	Wine quality	Thyroid disease	Nursery
50	1.44%	14.28%	2.02%	7.00%	0.02%	5.04%	8.97%	0.06%	0.38%
60	0.71%	12.05%	2.01%	5.78%	**0.02%**	4.19%	7.52%	0.04%	0.23%
70	0.78%	9.92%	1.43%	5.15%	0.01%	3.36%	6.53%	0.03%	0.24%
80	**0.10%**	8.79%	1.14%	4.52%	0.01%	3.13%	5.64%	0.02%	0.19%
90	0.31%	7.64%	1.03%	**4.20%**	0	2.68%	5.02%	0.02%	0.17%
100	0.73%	7.72%	0.84%	3.68%	0	**2.47%**	4.51%	0.02%	0.16%
110	0.60%	6.77%	0.87%	3.37%	0	2.40%	4.02%	**0.02%**	0.13%
120	0.51%	6.26%	0.62%	3.24%	0	2.02%	**3.66%**	0.01%	0.13%
130	0.60%	**5.74%**	0.57%	2.93%	0	1.87%	3.37%	0.01%	0.11%
140	0.47%	5.63%	**0.45%**	2.69%	0	1.66%	3.12%	0.01%	0.10%
150	0.53%	5.18%	0.64%	2.47%	0	1.62%	2.93%	0.01%	**0.09%**
Average	0.76%	7.76%	0.95%	4.09%	0.01%	2.77%	5.03%	0.02%	0.18%

less than 1.3x that of the unprotected implementation, and significantly reduced when compared to the traditional TBMR technique (that is twice the unprotected case). However, for datasets with poor performance (the accuracy is lower than 80%), the percentage of elements for which recomputation is needed, is up to approximately 70%; thus, the extended VM scheme is not very efficient in these cases. For these datasets, the *k*+1 NNs scheme is evaluated next.

The *k*+1 NNs scheme is simulated in the third evaluation. As mentioned previously errors are rare events, so it is reasonable to inject errors only in the initial *k*NNs implementation; this also refers to the worst case because if errors occur in the recomputation for the *k*+1 NNs, they can be fully handled by the proposed *k*+1 NNs scheme (considered as Type 1 or Type 3 errors). The evaluation results are shown in Table IV; compared with the results for unprotected *k*NNs (Table II), the percentage of errors that modify the classification result has been reduced by more than 10x, so now lower than 1% in all cases (and even 0 for several datasets). Therefore, the *k*+1 NNs scheme provides very good error protection. As discussed in Section III-B, the *k*+1 NNs scheme only recomputes the distances for the *k*+1 NNs to detect errors, therefore the overhead for the newly performed computation is very low, or even negligible when compared with the unprotected scheme, because the size of the dataset (the number of elements) is generally far larger than *k*.

The first two evaluations presented above have been conducted to assess the extended VM scheme in an ensemble classifier, in particular, Random Forest (RF). The percentage of elements that modify the classification result in the unprotected RF classifier is shown in Table V. The result is lower than 0.30% for the optimal *T* in all cases, showing the extreme intrinsic robustness of the RF algorithm. This makes possible to use it with no protection in applications that allow some errors. Moreover, in applications that require a complete error tolerance, the extended VM scheme also has an excellent performance, correcting all errors at a very low execution overhead. Table VI shows the percentage of elements for which recomputation is needed by the extended VM scheme; for datasets with good performance, the percentage is lower than 3%. Even for datasets with poor performance, the percentage is also always lower than 6% for the optimal *T*. Therefore, errors can be detected at low cost.

TABLE VII
PERFORMANCE OF DIFFERENT SCHEMES

	***k*NNs classifiers**		**RF classifiers**
	VM	***k*+1 NNs**	**VM**
Error correction rate	100%	Above 99%	100%
Recomputation overhead	Dependent on the dataset	Very low	Low

V. DISCUSSION

The important features of the three schemes analyzed and evaluated in Section IV are summarized in Table VII in terms of single errors tolerant capability (that are considered in this paper) and recomputation overhead.

***Failure rate*:** Compared with the unprotected case, the extended VM scheme can reduce the percentage of errors that modify the classification result for both *k*NNs and RF to 0, providing full error tolerance for single errors on distance/tree computations. The *k*+1 NNs scheme reduces the percentage of errors to values lower than 1%, because only Type 2 errors cannot be handled. In addition to tolerating single errors, it is of interest to mention that the proposed schemes can also handle multiple errors as long as they affect only one distance computation in a *k*NNs or the computation for one tree in a RF; otherwise they may fail in some cases when the errors affect multiple computations and change the voting result.

Recomputation overhead: As presented in the introduction, the proposed schemes are utilized for ML algorithms that run on a processor (in which hardware implementations are difficult to modify); only the computational overheads in terms of execution time and power consumption are incurred. To detect errors, the extended VM scheme used in *k*NNs classifiers recomputes the distances from all elements to the one being classified, therefore the recomputation overhead depends on the size of the dataset (the number of training elements). When employing the *k*+1 NNs scheme, the recomputation overhead is very low because only the distances for *k*+1 nearest neighbors must be computed again. In a RF classifier protected by the extended VM scheme, the recomputation is done only for the trees that have the majority class when VM is two. Therefore, the number of trees having a majority class should be at most (approximately) half of the total number, i.e., at a level of a dozen. The cases in which full recomputation is required, occur only when the margin between the majority class and the second majority class is <2. Therefore, the recomputation overhead is still low, but higher than for the *k*+1 NNs scheme.

VI. CONCLUSION

In this paper, error-tolerant schemes for voting classifiers using Machine Learning algorithms with multiple classes have been proposed. Both single voter classifiers (*k* Nearest Neighbors, *k*NNs) and ensemble classifiers (Random Forest, RF) have been considered.

Initially, we have proposed the extended Voting Margin (VM) scheme to protect the classifiers against single errors that occur during the classification procedure. The scheme is extended from the existing VM scheme of [27] used for *k*NNs classifiers with two classes only. By checking a VM of three or VM of two with some additional constraints, the classification result is determined to be correct, such that single errors have no impact on the classification and the recomputation operation for error detection can be saved. Due to the intrinsic robustness of the classifiers, the extended VM scheme can significantly save recomputation overhead compared with the conventional Time-Based Modular Redundancy (TBMR) technique. *k*NNs and RF classifiers have been used to evaluate the application of the extended VM scheme in different types of classifiers in this paper. By utilizing several widely used datasets with multiple classes taken from a public repository, the extended VM scheme has been shown to be efficient only for datasets with good performance (having an accuracy > 80%) using *k*NNs, saving of at least 70% in recomputation overhead is accomplished. For datasets with poor performance the reduction has been limited to less than 40%. When using RF classifiers, the scheme works well for all datasets, saving more than 90% of the recomputation overhead. Subsequently, another scheme (referred to as *k*+1 NNs) has been proposed as an alternative, targeting the cases in which the extended VM scheme is not very efficient for *k*NNs classifiers. In the *k*+1 NNs scheme, the classification is the same as the original *k*NNs algorithm, but once all distances are established, the $k+1^{th}$ NN is also retained. Then the distances for all *k*+1 neighbors are recomputed to check with those kept from the first time to detect errors. To correct errors and have a correct classification result, distance recomputation for a third time is required when an error is detected. The proposed scheme can deal with all errors that move an element that was not in the set of *k*NNs and now is included; this scheme provides good protection for *k*NNs classifiers, because the correctable errors account for most of the cases. This has been corroborated by the simulation results, showing that the percentage of elements that can modify the classification results, is lower than 1%; for some datasets, it can be 0. The computation overhead is also significantly reduced compared with the extended VM scheme. Overall, in single voter classifiers such as *k*NNs, the extended VM scheme (*k*+1 NNs scheme) is the best option for datasets with good (poor) classification accuracy. In ensemble classifiers such as RF, the extended VM scheme always has good performance.

REFERENCES

[1] S. Das, A. Dey, A. Pal, *et al.*, "Applications of Artificial Intelligence in Machine Learning: Review and Prospect", *International Journal of Computer Applications*, vol.115, no.9, pp.31-41, 2015.

[2] V. Sze, Y.H. Chen, J. Emer, *et al*., "Hardware for Machine Learning: Challenges and Opportunities", *IEEE Custom Integrated Circuits Conference (CICC)*, pp. 1-8, 2017.

[3] Z. Ma, H. Yu, W. Chen, *et al*., "Short Utterance Based Speech Language Identification in Intelligent Vehicles With Time-Scale Modifications and Deep Bottleneck Features", *IEEE Trans. on Vehicular Technology*, vol. 68, no. 1, pp. 121-128, 2019.

[4] Z. Ma, D. Chang, J. Xie, *et al*., "Fine-Grained Vehicle Classification With Channel Max Pooling Modified CNNs", *IEEE Trans. on Vehicular Technology*, vol. 68, no. 4, pp. 3224-3233, 2019.

[5] F. Tang, Z. M. Fadlulah, N. Kato, *et al*., "AC-POCA: Anticoordination Game Based Partially Overlapping Channels Assignment in Combined UAV and D2D-Based Networks", *IEEE Trans. on Vehicular Technology*, vol. 67, no. 2, pp. 1672-1683, 2018.

[6] D. Chang, Y. Ding, Y. Xie, *et al*., "The Devil is in the Channels: Mutual-Channel Loss for Fine-Grained Image Classification", *IEEE Trans. on Image Processing*, vol. 29, pp. 4683-4695, 2020.
[7] G. B. Huang, H. Zhou, X. Ding, *et al*., "Extreme Learning Machine for Regression and Multiclass Classification", *IEEE Trans. on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, no. 2, pp. 513-529, 2011.
[8] A. Liaw and M. Wiener, "Classification and Regression by Random Forest", *R news*, vol.2, no.3, pp.18-22, 2002.
[9] Q. Hu, D. Yu and Z. Xie, "Neighborhood Classifiers", *Elsevier, Expert Systems with Applications,* vol. 34, no.2, pp. 866-876, 2008.
[10] S. Zhang, X. Li, M. Zong, *et al*., "Efficient kNN Classification with Different Numbers of Nearest Neighbors", *IEEE Trans. on Neural Networks and Learning Systems*, vol. 29, no. 5, pp. 1774-1785, 2018.
[11] L. Breiman, "Random Forests", *Springer, Machine Learning*, vol.45, no.1, pp.5-32, 2001.
[12] G. Gui, F. Liu, J. Sun, *et al.*, "Flight Delay Prediction Based on Aviation Big Data and Machine Learning", *IEEE Transactions on Vehicular Technology*, vol. 69, no. 1, pp. 140-150, 2020.
[13] M. Nicolaidis, "Soft Errors in Modern Electronic Systems", *Springer Science & Business Media*, 2010.
[14] T. Karnik, P. Hazucha, "Characterization of Soft Errors Caused by Single Event Upsets in CMOS Processes", *IEEE Trans. on Dependable and Secure Computing*, vol. 1, no. 2, pp. 128-143, 2004.
[15] A. L. Buczak and E. Guven, "A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection", *IEEE Communications Surveys & Tutorials*, vol.18, no. 2, pp. 1153-1176, 2016.
[16] I. Cara, and E. d. Felder, "Classification for Safety-Critical Car-Cyclist Scenarios Using Machine Learning", *IEEE 18th International Conference on Intelligent Transportation Systems*, pp. 1995-2000, 2015.
[17] Y. Xun, J. Liu, N. Kato, *et al*., "Automobile Driver Fingerprinting: A New Machine Learning Based Authentication Scheme", *IEEE Trans. on Industrial Informatics*, vol. 16, no. 2, pp. 1417-1426, 2020.
[18] P. Koopman, M. Wagner, "Autonomous Vehicle Safety: An Interdisciplinary Challenge", *IEEE Intelligent Transportation Systems Magazine*, vol. 9, no. 1, pp. 90-96, 2017.
[19] "ISO 26262: Road Vehicles-Functional Safety", International Standard, 2011.
[20] M. Augustin, M. Goessel, R. Kraemer, "Reducing the Area Overhead of TMR-Systems by Protecting Specific Signals", *IEEE 16th International On-Line Testing symposium*, pp. 268-273, 2010.
[21] K. Chen, L. Chen, P. Reviriego, *et al*., "Efficient Implementations of Reduced Precision Redundancy (RPR) Multiply and Accumulate (MAC)", *IEEE Trans. on Computers*, vol. 68, no. 5, pp. 784-790, 2019.
[22] P. J. Mabey, "Mobile Radio Data Transmission-Coding for Error Control, *IEEE Trans. on Vehicular Technology*, vol. 27, no. 3, pp. 99-109, 1978.
[23] S. Lin and D. J. Costello, "Error Control Coding", *2nd ed. Englewood Cliffs*, NJ, USA: Prentice-Hall, 2004.
[24] C. Borchert, H. Schirmeier, and O. Spinczyk, "Generic Soft-Error Detection and Correction for Concurrent Data Structures", *IEEE Trans. on Dependable and Secure Computing*, vol.14, no. 1, pp. 22-36, 2017.
[25] D. Ravi, C. Wong, B. Lo, *et al*., "Deep Learning for Human activity Recognition: A Resources Efficient Implementation on Low-Power Devices", *IEEE 13th International Conference on Wearable and Implantable Body Sensor Networks*, 2016.
[26] D. Ravi, C. Wong, B. Lo, *et al*., "A Deep Learning Approach to no-Node Sensor Data Analytics for Mobiles or Wearable Devices", *IEEE Journal of Biomedical and Health Informatics*, vol. 21, no. 1, pp.56-64, 2017.
[27] S. Liu, P. Reviriego, J.A. Hernández, *et al*., "Voting Margin: A Scheme for Error-Tolerant *k* Nearest Neighbors Classifiers", *IEEE Trans. on Emerging Topics in Computing*, 2019 (Early Access).
[28] T. Li, C. Zhang and M. Ogihara, "A Comparative Study of Feature Selection and Multiclass Classification Methods for Tissue Classification Based on Gene Expression", *Bioinformatics*, vol.20, no.15, pp.2429-2437, 2004.
[29] M. Aly, "Survey on Multiclass Classification Methods", *Neural Networks*, vol.19, pp.1-9, 2005.
[30] M. Khun and K. Johnson, "Applied Predictive Modeling", *Springer* 2013.
[31] D. Dua and C. Graff, "UCI Machine Learning Repository", Irvine, CA: University of California, School of Information and Computer Science, 2019.
[32] S. Hussain, N.A. Dahan, F.M. Ba-Alwib, *et al*., "Educational Data Mining and Analysis of Students, Academic Performance Using WEKA", *Indonesian Journal of Electrical Engineering and Computer Science*, vol.9, no.2, pp. 447-459.2018.
[33] B. Johnson, R. Tateishi and Z. Xie, "Using Geographically-Weighted Variables for Image Classification", *Remote Sensing Letters*, vol.3, no.6, pp. 491-499, 2012.
[34] J. P. Siebert, "Vehicle Recognition Using Rule Based Methods", Turing Institute Research Memorandum TIRM-87-018, 1987.
[35] C. Higuera, K.J. Gardiner and K.J. Cios, "Self-Organizing Feature Maps Identify Proteins Critical to Learning in a Mouse Model of Down Syndrome", *PLOS ONE*, vol.10, no.6, 2015.
[36] P. Cortez, A. Cerdeira, F. Almeida, *et al*., "Modeling Wine Preferences by Data Mining from Physicochemical Properties", *Elsevier, Decision Support Systems,* vol.47, no.4, pp.547-553, 2009.

APPENDIX

The dependency of the classification accuracy on the number of neighbors used in the *k*NNs algorithm and the number of trees in the Random Forest is shown in the next two figures for the datasets considered in this paper.

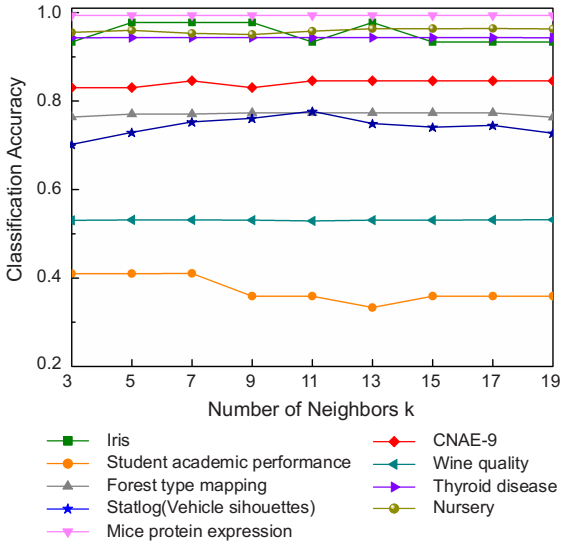

Figure 1 Classification accuracy of different datasets when using *k*NNs at different number of neighbors *k*.

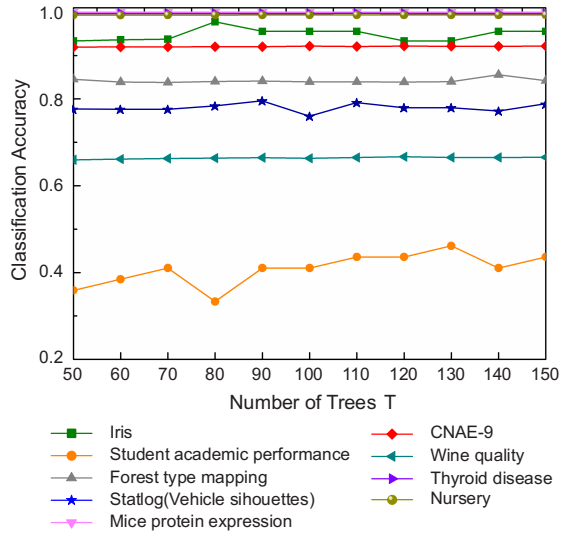

Figure 2 Classification accuracy of different datasets when using Random Forest at different number of trees *T*.