POLITECNICO DI TORINO
Repository ISTITUZIONALE

Less-is-Better Protection (LBP) for Memory Errors in kNNs Classifiers

Original
Less-is-Better Protection (LBP) for Memory Errors in kNNs Classifiers / Liu, S., Reviriego, P., Montuschi, P., Lombardi, F.. - In: FUTURE GENERATION COMPUTER SYSTEMS. - ISSN 0167-739X. - ELETTRONICO. - 117:4(2021), pp. 401-411. [10.1016/j.future.2020.12.015]

Availability:
This version is available at: 11583/2858243 since: 2021-01-19T19:39:39Z

Publisher:
Elsevier

Published
DOI:10.1016/j.future.2020.12.015

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
Elsevier postprint/Author's Accepted Manuscript

© 2021. This manuscript version is made available under the CC-BY-NC-ND 4.0 license
http://creativecommons.org/licenses/by-nc-nd/4.0/.The final authenticated version is available online at:
http://dx.doi.org/10.1016/j.future.2020.12.015

(Article begins on next page)

07 November 2026

Less-is-Better Protection (LBP) for Memory Errors in *k*NNs Classifiers

Shanshan Liu[1], *Member, IEEE*, Pedro Reviriego[2], *Senior Member, IEEE*, Paolo Montuschi[3], *Fellow, IEEE* and Fabrizio Lombardi[1], *Fellow, IEEE*

[1]*Northeastern University, Dept. of ECE, Boston, MA 02115, USA (email: ssliu@ece.neu.edu, lombardi@ece.neu.edu)*
[2]*Universidad Carlos III de Madrid, Av. Universidad 30, 28911 Leganés, Madrid, Spain (email: revirieg@it.uc3m.es)*
[3]*Politecnico di Torino, Dipartimento di Automatica e Informatica, 10129 Torino, Italy (email: paolo.montuschi@polito.it)*

Abstract— Classification is used in a wide range of applications to determine the class of a new element; for example, it can be used to determine whether an object is a pedestrian based on images captured by the safety sensors of a vehicle. Classifiers are commonly implemented using electronic components and thus, they are subject to errors in memories and combinational logic. In some cases, classifiers are used in safety critical applications and thus, they must operate reliably. Therefore, there is a need to protect classifiers against errors. The *k* Nearest Neighbors (*k*NNs) classifier is a simple, yet powerful algorithm that is widely used; its protection against errors in the neighbor computations has been recently studied. This paper considers the protection of *k*NNs classifiers against errors in the memory that stores the dataset used to select the neighbors. Initially, the effects of errors in the most common memory configurations (unprotected, Parity-Check protected and Single Error Correction-Double Error Detection (SEC-DED) protected) are assessed. The results show that surprisingly, for most datasets, it is better to leave the memory unprotected than to use error detection codes to discard the element affected by an error in terms of tolerance. This observation is then leveraged to develop Less-is-Better Protection (LBP), a technique that does not require any additional parity bits and achieves better error tolerance than Parity-Check for single bit errors (reducing the classification errors by 59% for the Iris dataset) and SEC-DED codes for double bit errors (reducing the classification errors by 42% for the Iris dataset).

Index Terms— Classification, memories, error tolerance, *k* nearest neighbors, error control codes

1 Introduction

MACHINE learning (ML) allows computers to extract patterns from empirical data and learn from these patterns to perform classification on new data. Learning is accomplished using labelled data, such that the input for each element in the training set consists of the features used for classification and the class of the element. This is known as supervised learning; in unsupervised learning, the data is unlabeled and the algorithm has to extract the patterns from raw data [1].

Supervised classification is used in a wide range of applications, such as autonomous driving, recommender systems, and medical diagnosis [2]-[4]. The classifiers are typically implemented using electronic systems. The labelled data or trained model is stored in memory and the procedure of applying the model is performed by an arithmetic unit or run in a processor. However, these components are prone to suffer from errors/faults due to for example, radiation-induced soft errors in memories and arithmetic errors in processor or computational units [5], [6]. This can cause data corruption and thus, it may affect the classification result. Therefore, protection of classifiers against errors should be considered to support reliable operation, especially when applying the classifiers in safety or critical applications, such as disease diagnosis, cyber security, land mines detection and finance [7]-[9].

One of the simplest, yet powerful classification algorithms is *k* Nearest Neighbors (*k*NNs) that has been widely studied in the past decades [10]-[14]. It is also called instance-based or lazy learning, because only majority voting is performed among the classes of several instances (i.e. the *k*NNs selected in the training set) to predict the output class of a new element. *k*NNs can either be utilized alone to perform a classification task, or with an improved design, or combined with other classification algorithms [8], [9]. Errors in the memory that stores the elements of the training set, or in the arithmetic units that compute the distance from the new element (being classified) to each stored element to select the nearest neighbors, may have an impact on the *k*NNs set and thus modify the classification result. Therefore, when *k*NNs is used in safety or critical applications, the results must be reliable in the presence of errors, otherwise they may cause potential life and property losses.

The implementation of a classifier mainly includes two parts: i) the arithmetic circuits that perform the calculation and ii) the memory that stores the dataset or model information, and thus, error tolerance must be considered for both. A common solution to provide error tolerance in arithmetic circuits relies on either spatial, or temporal redundancy. Spatial redundancy is implemented by replicating the entire circuit or module multiple times; then data in the copies are used as inputs to a comparison or

• *Manuscript received October 6, 2020, revised December 10, 2020, accepted December 13, 2020.*
• *Corresponding author: Shanshan Liu*

majority voting unit to detect or correct errors. For example, Double Modular Redundancy (DMR, i.e., replicating the circuit twice) can detect any error in a single module by comparing the two duplicated inputs, while Triple Modular Redundancy (TMR, i.e., replicating the circuit three times) can guarantee a correct output under single module errors by taking a majority voting among the triplicated data [15]. The hardware utilization and power consumption introduced by spatial redundancy depend on the size of the circuit and for large circuits, the overheads are not acceptable for some applications (e.g., battery powered systems). Reduced Precision Redundancy (RPR) has been proposed to reduce the overhead by introducing several copies with reduced precision. However, an inexact output is obtained for some cases, making RPR only applicable to those systems that tolerate a limited range of deviation from the correct result [16], [17]. Instead of performing an operation in several replicated circuits at the same time, temporal redundancy is implemented by performing the operation several times using the same circuit. For example, in *k*NNs protected by a temporal redundancy technique, distances are computed twice and results are compared to detect errors. Once an error is detected, computation is performed for a third time to obtain the correct data by computing a majority voting among the three results. However, this still incurs in significant overhead in terms of power consumption and execution time. Recently, an algorithm-based error tolerance (ABET) technique has been proposed to protect *k*NNs with a binary classification against arithmetic errors by exploiting the intrinsic redundancy of the *k*NNs algorithm [18]. By refining and extending the property of *k*NNs that a single error cannot change the classification result when the neighbors have a voting margin, the approach is able to avoid re-computation in many cases, reducing more than 60% of the overhead compared to a traditional temporal redundancy. This approach is then extended to voting classifiers, such as *k*NNs and Random Forests with multiple classes [19]. The ABET technique has also been investigated to protect other classifiers (such as the Support Vector Machine (SVM)) against computational errors at a lower overhead [20].

In terms of protecting memories against errors, spatial redundancy can be attractive for protecting only some critical words, but not for the entire memory because in this case, the overhead is extremely large due to the replicated memories. An alternative option is to use Error Control Codes (ECCs, also referred to as Error Correction Codes) [21], [22]. By adding several memory cells in each word to store parity bits, as well as an encoder and decoder to the entire memory, one or more-bit errors can be detected or corrected. Generally, the error detection and/or correction capability is related to the number of parity bits and the complexity of the encoder/decoder; the first feature determines the memory overhead, while the second feature affects the execution time (when implemented in software) and circuit complexity. Therefore, a small size memory storing only the ML model information (e.g., the support vectors in an SVM, or the weights of the neurons in a Neural Network) can be sufficiently protected by powerful ECCs, however the large size of a memory storing the entire dataset required by *k*NNs is not viable, because the ECC redundancy further increases the additional storage requirements. This motivates this paper to investigate an error-tolerant technique to protect the *k*NNs' memory at a lower overhead, so that independently whether the *k*NNs is used alone, or combined with other algorithms in safety/critical applications, all classifiers can be efficiently protected.

The most commonly used and simplest ECCs for memories are Parity-Check (that uses a single parity bit to detect single errors), and Single Error Correction-Double Error Detection (SEC-DED) codes that are able to correct single bit errors and detect double bit errors [21]. When using Parity-Check protected (SEC-DED protected) memories for *k*NNs classifiers, single bit errors (double bit errors) that are the most common error pattern, can only be detected, but not corrected. In this case, discarding the element affected by the error from the training set seems to be beneficial, because the erroneous element will not be used for classification. In this paper, the effectiveness of Parity-Check and SEC-DED based protection schemes is evaluated and a more efficient technique referred to as Less-is-Better Protection (LBP) is proposed. The significant contributions of this paper are as follows:

- The evaluation results show that surprisingly, for most datasets, the traditional ECC solution is counterproductive in terms of reducing classification errors, i.e., it is better to leave the memory unprotected than to use Parity-Check; for SEC-DED, it is better not to discard the element if a double error is detected.
- The above observation is leveraged to develop the so-called LBP technique, which does not require additional parity bits and achieves better error tolerance than a Parity-check (SEC-DED) for single bit errors (double bit errors).
- The proposed LBP also reduces the memory needed compared to the unprotected *k*NNs, as well as the impact of errors on the classification results for datasets with a small number of features.

The remaining part of this paper is organized as follows. Section 2 covers the background material on the *k*NNs algorithm and the impact of errors in its implementation; the most common error tolerance memory configurations (i.e., employing Parity-Check and SEC-DED codes) are also discussed. In Section 3, the effects of memory errors in the unprotected, Parity-Check protected, and SEC-DED protected *k*NNs are evaluated, and several observations on such results are discussed; they are then leveraged to propose the Less-is-Better Protection scheme in Section 4. Section 5 evaluates the error protection capabilities of the proposed LBP as well as the memory overhead. Finally, Section 6 concludes the paper.

2 PRELIMINARIES

This section first provides a brief review of *k*NNs classifiers as well as their implementation. Then, the impact of memory errors that affect the stored features or the labels

of the *k*NNs training set on the classification results, is analyzed; traditional protection schemes for different error scenarios are also discussed.

2.1 *k* Nearest Neighbors Classifiers

As introduced previously, *k* Nearest Neighbors (*k*NNs) is one of the simplest, yet powerful classification algorithms. For classification of a new element, the distance from the element being classified to other training elements is computed by using different methods [10], [23]. One common solution is to compute the Euclidean distance based on all features of the elements (the values of the features are normalized to keep the same weight), hence the set of *k*NNs can be selected. As introduced previously, majority voting is executed among the classes of *k*NNs to determine the classification result. During the voting process, a special case occurs when there are equal votes for multiple classes, leading to a tie. When there are only two classes, a tie can be broken by selecting an odd value for *k*, such that the number of elements with each class can be simply compared with the threshold (i.e., ceil((*k*+1)/2)) to find the majority. Figure 1 a) illustrates an example of the *k*NNs algorithm with two classes, the grey element is the one being classified and its predicted class is B, because *k*=5 and there are two elements with Class A (yellow elements) and three with Class B (blue elements) in the 5NNs set.

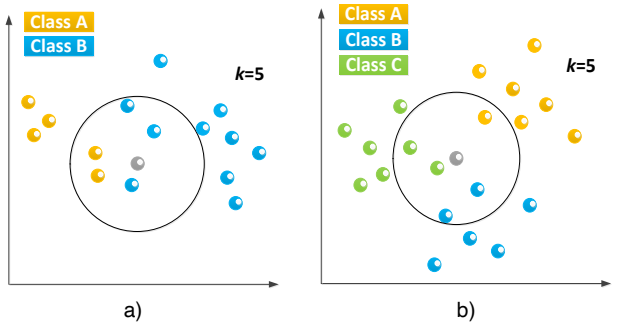

Figure 1 Illustration of the *k*NNs algorithm with two classes (in a)) and three classes (in b)). The yellow, blue and green elements are stored with their class. The grey element is the one being classified. As *k* = 5, initially the five elements closest to it are identified (shown by the solid circle).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

When there are multiple classes (i.e., more than two), voting is more complicated. Once the *k*NNs are selected, the number of elements in each class is compared. If there is only one majority class, then it is used as the final classification result; if there are two or more classes voted by the same number of neighbors (i.e., there is a tie), tie-breaking methods must be utilized to determine the classification result. A possible solution is to select the class of the nearest neighbor that belongs to the majority, as illustrated in the example shown in Figure 1 b). In Figure 1 b), the elements have three classes; Classes B and C have the same largest number (i.e., two) of neighbors in the *k*NNs set (i.e., a tie occurs). Therefore, in this case the predicted class for the grey element is C, because the nearest majority neighbor belongs to Class C.

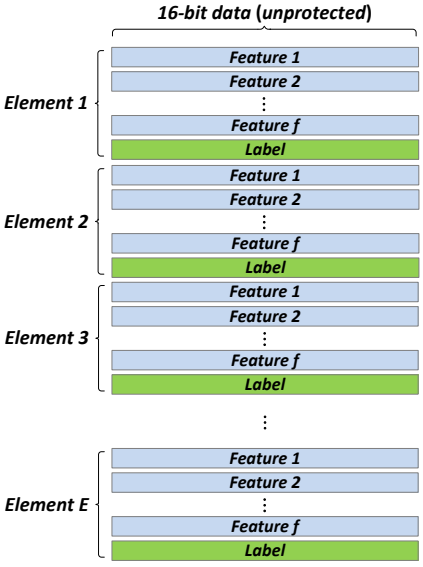

Figure 2 An unprotected memory storing *k*NNs elements.

The main components of a *k*NNs implementation include the labelled elements that are commonly stored in a memory, as well as the computation of distances and comparisons to find the nearest neighbors (commonly executed in a processor). Since memories are prone to suffer from a number of errors/faults (such as radiation induced soft errors) and computational units in a processor suffer from arithmetic errors (both causing data corruption), the classification result can be affected when leaving the memory and processor unprotected. For example, if an arithmetic error occurs during the distance computation process, a reduction in the distance from an element that should not belong to the *k*NNs set to the new element being classified, may result in the k^{th} NN being replaced if its distance to the new element is larger than the incorrect distance. In this case, the classification result may be changed if the element that is moved by the error has a different class from the k^{th} NN. To deal with errors affecting the computation of distances in *k*NNs, efficient algorithm-based error tolerance techniques have been recently proposed [18], [19]. As for memory errors affecting the value of stored features or labels of *k*NNs elements, the impact on classification results and the most common solutions used in different error scenarios will be discussed in the next subsection.

2.2 Memory Errors and Protection

The features (normalized) and label of each element in the *k*NNs training set are commonly stored in a memory; a simple organization is illustrated in Figure 2, in which the value of each feature or label is stored in a single word. For classification of a new element, the information stored in the memory is read and used to select the *k*NNs set.

Memories are prone to suffer from different types of errors or faults; they can modify the contents of a memory word, causing data corruption. For example, radiation induced soft errors are one of the reliability challenges in memories and can corrupt a stored bit from a logic "1" to

"0" (or vice versa). These errors can affect multiple cells, but the most common error patterns are the single bit error (SE) and the double adjacent bit error (DAE) [24], while double random bit errors (DREs) can also happen due to error accumulation. In a memory that stores features and labels of the *k*NNs elements, errors that occur on a feature word (label word), can affect the set of nearest neighbors (the class of a neighbor), possibly modifying the classification result. Therefore, even though the errors are rare (e.g., a radiation-induced error may take 10^5 days to occur in a 64K memory [25], and the rate may increase proportionally for a large memory), they cannot be ignored, especially in safety or critical applications in which reliable operation is required.

Error correction codes (ECCs) are widely used to protect memories against errors. By introducing some redundant cells in each memory word for storing the parity bits (computed during the encoding process of the ECCs), errors can be detected or corrected during the decoding process. The most commonly used solution to detect SEs is a single bit Parity-Check; to correct SEs and detect DAEs and DREs, Single Error Correction-Double Error Detection (SEC-DED) codes are utilized.

Parity-Check: A single parity bit that covers all data bits, can efficiently detect SEs on any bit (including the parity bit itself). There are two types of Parity-Check: even Parity-Check and odd Parity-Check. When using an even Parity-Check, in the encoding process (which is performed prior to the write operation), for each word, a *xor* operation is computed on all data bits first to obtain the parity bit; then it is stored with the data bits in the word. In the decoding process (which is performed after the read operation), a syndrome is calculated by performing a *xor* operation on all stored data bits and the parity bit, then it is used for error detection. For an odd Parity-Check, the implementation is similar; only an extra *not* operation is implemented following the *xor* gate in both the encoder and decoder. Therefore, in this paper, we consider utilizing the even Parity-Check.

A syndrome with a value of zero indicates that the word is error-free, while a syndrome with a value of one indicates that there is an error. A single bit Parity-Check can only detect SEs, because a DAE or DRE will lead to a syndrome with a value of zero, which is the same as that in the error-free case. For *k*NNs protected by a single bit Parity-Check, if an SE occurs on the feature word or label word of an element, it can be detected and the corrupted element will be discarded due to the impossibility to reconstruct the correct value, so it is not used as a candidate for nearest neighbor.

SEC-DED codes: Single Error Correction-Double Error Detection codes ensure that codewords have a minimum distance of four, so that single bit errors can be corrected, and double bit errors detected. SEC-DED codes need r parity bits to protect up to 2^{r-1}-r data bits [21]. For example, six parity bits are needed (i.e., r=6) to perform SEC-DED on a 16-bit data.

In the encoding process of SEC-DED codes, the data bits are multiplied by the Generating Matrix **G** (associated to the codes) to obtain the r parity bits, which are then stored with the data bits in each word. Figure 3 illustrates an example of protecting a memory that uses 16 bits to store features and labels of *k*NNs elements with six SEC-DED parity bits per word.

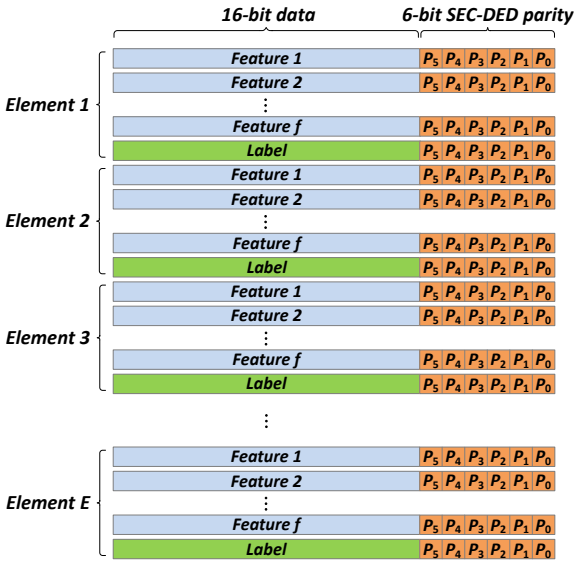

Figure 3 A memory storing *k*NNs elements protected by a (22,16) SEC-DED code.

In the decoding process, an r-bit syndrome is computed by multiplying the codeword (which includes all data and parity bits) with the Parity Check Matrix **H** (also associated to the code and commonly constructed by ensuring that the columns have an odd weight [21]). If the syndrome bits are all zeros, the codeword is error-free; when decoding is completed, the data bits are simply provided as output. If the syndrome bits have an odd number of ones, a single bit error is detected; the error pattern can be determined by comparing the syndrome with each column of the matrix **H**. In this case, a *xor* operation is performed on the codeword with the error pattern to obtain the corrected data. If the syndrome bits have an even number of ones, a double bit error is detected. In this case, the data bits will be output immediately, but an error detection flag is activated to indicate that the output is incorrect. Therefore, when using SEC-DED codes to protect *k*NNs, an element with an error detection flag will be discarded and not used as a candidate for nearest neighbor, because a double error has been detected, but not corrected.

3 Evaluation of Memory Errors in *k*NNs

The impact of errors in the memory that stores the elements, on the classification result depends on the type of error and the memory configuration. To cover the most relevant scenarios, three memory configurations are assessed: unprotected, Parity-Check protected and SEC-DED protected. As discussed in the previous section, these configurations correspond to the most frequently used memory configurations. As for errors, single bit errors (SEs), double adjacent bit errors (DAEs) and double random bit errors (DREs) are considered. Matlab is uti-

TABLE 1
Description of the Different Datasets

Dataset	Application	# Elements	# Features	# Classes	Using *k*NNs	
					Optimal *k*	Top accuracy
Pima Indians diabetes	Medicine	768	8	2	19	76.52%
Sonar [27]	Physics	208	60	2	3	85.48%
Banknote authentication	Business	1372	4	2	7	100.00%
Phishing websites [28]	Computer	2456	30	2	5	92.81%
Iris	Botany	150	4	3	5	93.95%
Forest type mapping [29]	Ecology	325	27	4	9	80.59%
Mice protein expression [30]	Biology	1080	80	8	3	98.78%
CNAE-9	Finance	1080	856	9	7	83.70%
Cervical cancer [31]	Medicine	858	36	2	5	92.25%
Nursery	Sociology	12960	8	5	17	96.19%

lized as simulation tool in this section.

3.1 Datasets

To assess the impact of memory errors on *k*NNs classification, ten widely used datasets from a public repository [26] are selected; they are described in Table 1. These datasets cover a wide range of applications, number of elements, and have different numbers of features and classes. In all cases, a simple and often used criterion is used to evaluate the classification performance, i.e., 70% of the dataset elements have been used for training, while the remaining 30% of the data is left for testing (note that a different split ratio may result in a slight different classification accuracy but it has no impact on the trend of the results evaluated for the work of this paper). The 70% training set is split into 10 blocks of equal size to run the well-known 10-fold cross-validation methodology [32] for the selection of the algorithm's hyperparameters, i.e., the optimal number of neighbors *k*. The top accuracy corresponding to the optimal *k* for each dataset is then obtained and also given in Table 1. The datasets are stored in a memory using 16 bits for each feature (the most significant bit is for the sign) and the label. The protection (Parity-Check or SEC-DED) is implemented per feature or label.

3.2 Error Injection

The impact of errors on the classification result is evaluated by using Matlab as per the following process. First an element is randomly selected with a uniform distribution and then an error (SE, DAE or DRE) is randomly injected on the stored features or label of this element (by upsetting the stored value of the selected bit/bits from "1" to "0" or vice versa). Once the error has been inserted, classification is performed for all elements in the testing set and the differences in the results versus the error-free classification are logged. Finally, the error is removed, and the procedure starts again. Since it has been found that the results for 10,000 trials were consistent with those with 100,000 trials, the process has been repeated by injecting 10,000 times for each of the three memory configurations (i.e., unprotected, Parity-Check protected and SEC-DED

TABLE 2
Percentage of Single Bit Errors (SEs) that Modify the Classification Result in the Unprotected *k*NNs

Error position		**Pima Indian diabetes**	**Sonar**	**Banknote auth-entication**	**Phishing websites**	**Iris**	**Forest type mapping**	**Mice protein expression**	**CANE-9**	**Cervical cancer**	**Nursery**
Feature	15	0.065%	0.078%	0	0.008%	0.110%	0.130%	0.009%	0	0	0.002%
	14	0.110%	0.082%	0	0.007%	0.090%	0.280%	0.009%	0.100%	0.005%	0.003%
	13	0.097%	0.081%	0	0.007%	0.040%	0.280%	0.006%	0.080%	0.005%	0.003%
	12	0.097%	0.081%	0	0.007%	0.040%	0.280%	0.006%	0.080%	0.005%	0.003%
	11	0.096%	0.070%	0	0.007%	0.040%	0.240%	0.003%	0.070%	0.005%	0.003%
	10	0.091%	0.065%	0	0.005%	0.040%	0.120%	0.003%	0.070%	0.004%	0.003%
	9	0.078%	0.065%	0	0.005%	0.020%	0.120%	0.003%	0.070%	0.004%	0.002%
	8	0.069%	0.065%	0	0.005%	0.020%	0.120%	0.003%	0.070%	0.002%	0.002%
	7	0.065%	0.061%	0	0.005%	0.020%	0.090%	0.003%	0.068%	0.002%	0.001%
	6	0.016%	0.048%	0	0	0.020%	0.050%	0.003%	0.056%	0.001%	0.001%
	5	0	0.004%	0	0	0.020%	0.031%	0	0.052%	0	0.001%
	4	0	0.004%	0	0	0	0	0	0.040%	0	0
	3	0	0	0	0	0	0	0	0.024%	0	0
	2	0	0	0	0	0	0	0	0.020%	0	0
	1	0	0	0	0	0	0	0	0.015%	0	0
	0	0	0	0	0	0	0	0	0.012%	0	0
	Ave.	0.049%	0.044%	0	0.004%	0.029%	0.109%	0.003%	0.052%	0.002%	0.002%
Label		0.225%	0.283%	0	0.015%	0.180%	0.193%	0.004%	0.060%	0.020%	0.003%

TABLE 3
Percentage of Single Bit Errors (SEs) that Modify the Classification Result in Different Schemes

Scheme	Pima Indian diabetes	Sonar	Banknote auth-entication	Phishing websites	Iris	Forest type mapping	Mice protein expression	CANE-9	Cervical cancer	Nursery
U	0.069%	0.048%	0	0.004%	0.059%	0.112%	0.003%	0.052%	0.002%	0.002%
P	0.097%	0.116%	0	0.011%	0.070%	0.223%	0.006%	0.075%	0.004%	0.003%
SEC-DED	0	0	0	0	0	0	0	0	0	0

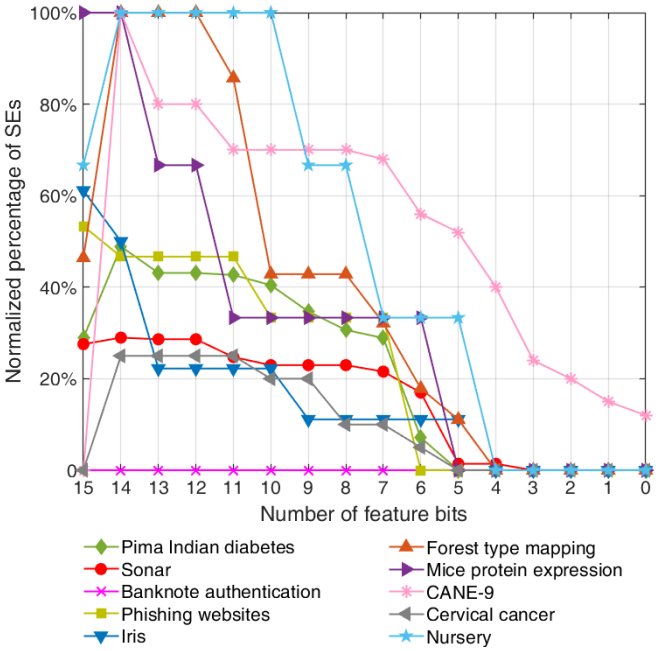

Figure 4 Normalized percentage of SEs that Modify the Classification Result in the Unprotected *k*NNs.

protected); the average results are presented next.

3.3 Single Bit Errors

The results for single bit errors in the unprotected memory configuration are presented in Table 2 for the values of feature and label words. In Table 2, errors in the label word have a significant impact, typically larger than the impact of an error on a feature. The impact of errors on each bit of a feature is also evaluated; for each dataset, the results are normalized by the largest value to clearly show the impact of each feature bit. This is shown in Figure 4; the most significant bits (MSBs, bit 15 is the MSB) tend to have a larger impact than the least significant bits (LSBs). Therefore, the label and MSBs of the features seem to be the most critical parts.

Consider the protected memory configurations; obviously, for SEC-DED protection, single errors are corrected and have no effect on the classification outcome. For Parity-Check protection, the error is detected, and the element is not used as a candidate for nearest neighbor. This prevents the error from causing an element to be one of the *k*NNs; however, the element in error is no longer used, which may also reduce the classification accuracy. The results for all three memory configurations are summarized in Table 3 (where U stands for the unprotected scheme, P stands for the Parity-Check protection scheme). As per Table 3, for all datasets considered, the unprotected configuration has a percentage of errors that modify the classification results lower than the Parity-Check protected memory. Therefore, surprisingly leaving the memory unprotected is better than using Parity-Check protection in terms of error tolerance.

3.4 Double Bit Errors

For the case of double errors, two patterns are considered: adjacent bit error and random bit errors. Double adjacent

TABLE 4
Percentage of Double Adjacent Bit Errors (DAEs) that Modify the Classification Result in the Unprotected *k*NNs

Error position		Pima Indian diabetes	Sonar	Banknote auth-entication	Phishing websites	Iris	Forest type mapping	Mice protein expression	CANE-9	Cervical cancer	Nursery
Feature	15&14	0.089%	0.130%	0	0.009%	0.110%	0.220%	0.009%	0.080%	0.001%	0.003%
	14&13	0.110%	0.160%	0	0.009%	0.096%	0.250%	0.009%	0.096%	0.005%	0.003%
	13&12	0.099%	0.140%	0	0.009%	0.071%	0.230%	0.007%	0.096%	0.005%	0.003%
	12&11	0.099%	0.140%	0	0.009%	0.067%	0.220%	0.007%	0.096%	0.005%	0.003%
	11&10	0.099%	0.120%	0	0.009%	0.067%	0.220%	0.006%	0.091%	0.005%	0.003%
	10&9	0.098%	0.120%	0	0.009%	0.067%	0.220%	0.006%	0.076%	0.005%	0.003%
	9&8	0.098%	0.100%	0	0.009%	0.064%	0.210%	0.004%	0.076%	0.004%	0.003%
	8&7	0.082%	0.100%	0	0.008%	0.064%	0.210%	0.004%	0.076%	0.003%	0.003%
	7&6	0.077%	0.083%	0	0.005%	0.058%	0.110%	0.003%	0.071%	0.003%	0.003%
	6&5	0.027%	0.019%	0	0.001%	0.056%	0.029%	0	0.050%	0.001%	0.001%
	5&4	0.011%	0.013%	0	0.001%	0.049%	0.014%	0	0.042%	0	0.001%
	4&3	0.007%	0.011%	0	0.001%	0.016%	0.003%	0	0.038%	0	0
	3&2	0	0.006%	0	0.001%	0.016%	0	0	0.033%	0	0
	2&1	0	0.003%	0	0.001%	0	0	0	0.032%	0	0
	1&0	0	0	0	0	0	0	0	0.030%	0	0
	Ave.	0.060%	0.076%	0	0.006%	0.053%	0.129%	0.004%	0.066%	0.002%	0.002%
Label		0.225%	0.283%	0	0.015%	0.187%	0.228%	0.006%	0.072%	0.020%	0.003%

TABLE 5
Percentage of Double Adjacent Bit Errors (DAEs) that Modify the Classification Result in Different Schemes

Scheme	Pima Indian diabetes	Sonar	Banknote auth-entication	Phishing websites	Iris	Forest type mapping	Mice protein expression	CANE-9	Cervical cancer	Nursery
U	0.078%	0.079%	0	0.006%	0.080%	0.133%	0.004%	0.066%	0.002%	0.002%
P	0.080%	0.080%	0	0.006%	0.084%	0.132%	0.004%	0.070%	0.002%	0.002%
SEC-DED	0.125%	0.150%	0	0.014%	0.091%	0.289%	0.008%	0.098%	0.006%	0.004%

TABLE 6
Percentage of Double Random Bit Errors (DREs) that Modify the Classification Result in the Unprotected *k*NNs

Error position	Pima Indian diabetes	Sonar	Banknote auth-entication	Phishing websites	Iris	Forest type mapping	Mice protein expression	CANE-9	Cervical cancer	Nursery
Feature	0.073%	0.077%	0	0.007%	0.062%	0.130%	0.003%	0.056%	0.003%	0.002%
Label	0.225%	0.283%	0	0.015%	0.187%	0.230%	0.003%	0.067%	0.020%	0.003%

TABLE 7
Percentage of Double Random Bit Errors (DREs) that Modify the Classification Result in Different Schemes

Scheme	Pima Indian diabetes	Sonar	Banknote auth-entication	Phishing websites	Iris	Forest type mapping	Mice protein expression	CANE-9	Cervical cancer	Nursery
U	0.090%	0.080%	0	0.007%	0.087%	0.134%	0.003%	0.056%	0.003%	0.002%
P	0.092%	0.082%	0	0.009%	0.084%	0.139%	0.003%	0.060%	0.003%	0.002%
SEC-DED	0.125%	0.150%	0	0.014%	0.091%	0.289%	0.008%	0.098%	0.006%	0.004%

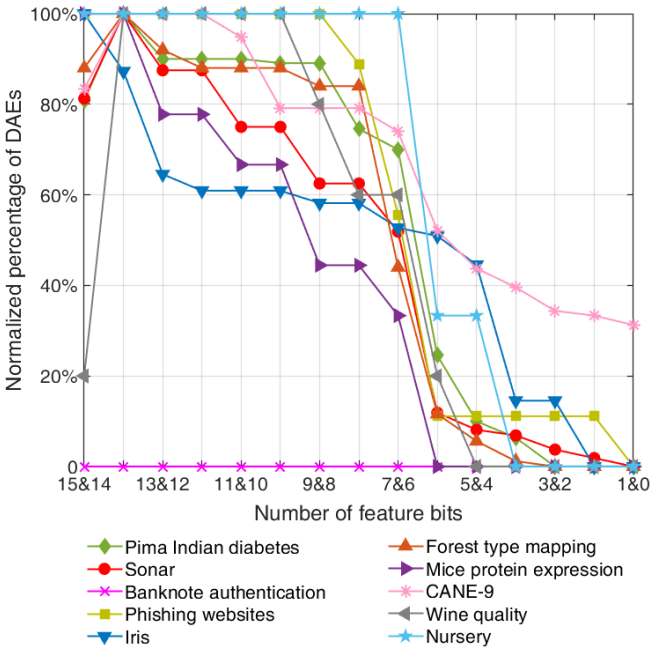

Figure 5 Normalized percentage of DAEs that Modify the Classification Result in the Unprotected *k*NNs.

bit errors (DAEs) can be caused by radiation induced Multiple Cell Upsets (MCUs) that affect nearby cells, while double random bit errors (DREs) can be due to error accumulation. The results for DAEs in an unprotected memory configuration are shown in Table 4 and the normalized results for each feature bits are plotted in Figure 5; again, errors on the MSBs of the features and on the label of each element are the ones that have a larger impact on the classification result. Compared with SEs, the impact tends to be slightly larger.

The results for the three memory configurations when they suffer DAEs are presented in Table 5. In this case the unprotected and Parity-Check protected memories have almost the same results. This occurs because a parity bit cannot detect double errors and thus, the elements are used for the classification even if they have an error. The only difference is that for a Parity-Check protected memory, the DAE can also affect the parity bit. Instead, the SEC-DED protection can only detect the DAE and remove the element from consideration as one of the *k*NNs. As per Table 5, SEC-DED protection is worse than no protection for DAEs in all datasets evaluated. Therefore, again less protection is better.

The results for double random bits errors (DREs) in an unprotected memory are presented in Table 6; again, errors on the labels have the largest impact. Finally, the impact on the classification results for DREs is summarized in Table 7. In all datasets, no protection is better than using SEC-DED; moreover, it is better to use SEC protection than SEC-DED protection because DED increases the percentage of errors that modify the classification result. This is interesting because SEC codes require one parity bit less than SEC-DED codes [21].

As for the experiments considered in subsections 3.3 and 3.4, the results show that under single bit errors, the unprotected memory configuration is better than using Parity-Check protection for all datasets considered; under double bit errors, the use of erroneous elements is in most cases better than just discarding them. These results are of a significant practical interest for designers; they are explained by using several observations and exploited to propose an efficient protection technique in the next section.

4 Less-is-Better Protection (LBP)

From the previous section, the utilization of ECCs to protect the memory of *k*NNs classifiers can be counterproductive; this is based on the following observations.

Observation 1: Errors on the MSBs of a feature tend to

move an element and then changing the *k*NNs set, but errors on the LSBs do not.

***Observation* 2**: Errors on the label modify the class of an element and tend to have a larger impact on the classification result than errors in a feature, because changing the class can be considered as moving the element. This is also the case under ECCs protection.

***Observation* 3**: For ECCs protection, when an element is discarded due to error detection, the impact on the classification result is the same as moving an element, so it is usually larger than for errors on a feature, because errors on LSBs have an extremely small probability (in few cases a zero probability) to move an element (as per Observation 1).

***Observation* 4**: The number of feature words is always greater than the label words; so, in the unprotected implementation, the errors on features account for a larger proportion.

***Observation* 5**: For ECCs protection, memory size is larger due to the additional cells that store the parity bits, so the probability of suffering from errors is increased because errors can occur also on the parity bits.

Let P_U be the probability of an incorrect classification result caused by a single bit error that affects an element stored in an unprotected memory; this is given by:

$$P_U = \frac{f}{f+c} \cdot p_f + \frac{c}{f+c} \cdot p_c \tag{1}$$

where f (c) is the number of feature (label) words for each element, and f is always much larger than c (i.e., the first term of Eq. (1) accounts for a significant proportion). p_f (p_c) is the probability of an incorrect classification result caused by an error on the feature (label) of the affected element in the unprotected configuration and $p_c > p_f$ (as Observation 2).

The probability of an incorrect classification result caused by an error that affects the same element but stored in the ECCs protected memory P_{ECCs} is given by

$$P_{ECCs} = p_d \tag{2}$$

where p_d is the probability of an incorrect classification result caused by discarding an element in the ECCs protected configuration; as per Observation 3, $p_d > p_f$.

Therefore, the impact of an error affecting the same element in the unprotected and ECCs protected configurations on the classification result can be compared by combining Eqs. (1) and (2) as following.

$$Ratio = \frac{P_U}{coef \cdot P_{ECC}} = \frac{\frac{f}{f+c} p_f + \frac{c}{f+c} p_c}{coef \cdot p_d} \tag{3}$$

where *coef* is the increase of error occurrence probability introduced by the extra memory cells for parity bits; it is equal to the ratio between the ECCs protected memory size and the unprotected memory size, so $coef > 1$.

Therefore, as $p_d > p_f$, $coef > 1$ and the second term in the numerator is small ($f >> c$), the value of Eq. (3) will In most cases be smaller than 1. This means that the impact of an error in the unprotected memory is rather small. Therefore, as per Observations 1 to 5 and the above discussion, ECCs protections that detect and discard erroneous elements, tend to be worse than no protection. However, Observation 2 suggests that the protection of the labels may reduce the impact of errors. By combining both these observations, a new scheme referred to as the Less-is-Better Protection (LBP) scheme, is proposed. This scheme protects the labels without adding any parity bit and leaves the features unprotected.

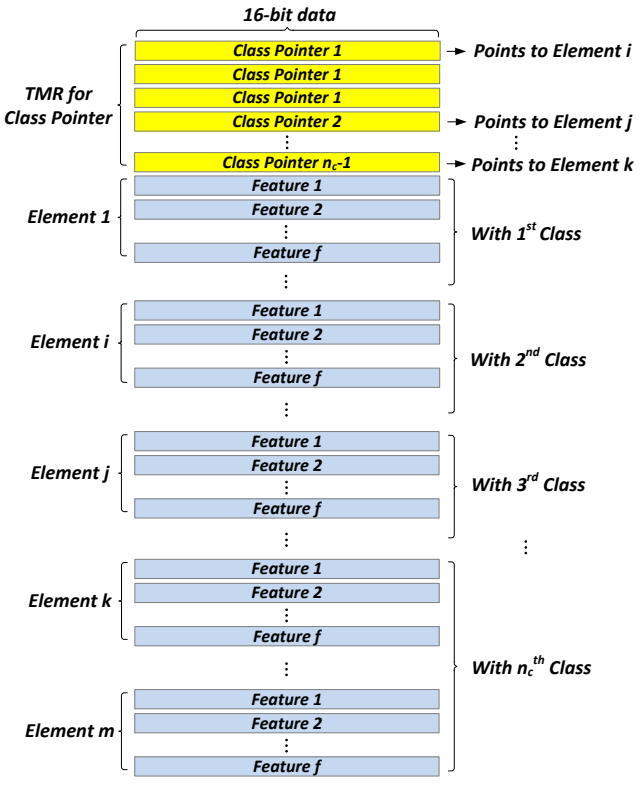

Figure 6 A memory storing *k*NNs elements protected by the proposed LBP scheme.

An interesting property of *k*NNs classifiers is that the elements in the training set can be stored in any order. This is like Content Addressable Memories (CAMs) in which the order of the elements stored can be exploited so that the addresses correspond to the parity bits [33]. However, for the memory used to store the *k*NNs elements, this only helps the protection of one feature or the label, because they are stored in different words.

An alternative option is to exploit the order to place elements that belong to the same class consecutively as illustrated in Figure 6. There is no need to store the labels on the elements; for a classifier with n_c classes, only n_c-1 pointers are required to mark the ranges that correspond to the different classes (Figure 6). Then, as elements are read to compute the distances, the pointers can be used to determine the class they belong to. Therefore, there cannot be errors on the labels, because they are not stored in memory. Errors can only affect the pointers, but since the number of pointers is small, they can be simply triplicated (i.e. employing TMR) to ensure a correct result. Combined with Observation 2 discussed previously, the impact of errors on the classification results can be reduced, unless the number of features is extremely large (as per Observation 4).

In the proposed Less-is-Better scheme, "Less" refers to two aspects.

- ECCs protection to detect and discard erroneous elements on *k*NNs memory is counterproductive in terms of reducing classification errors, so no protection is better.
- A memory with no stored label is better, because it is the part that has a larger impact on the classification results.

Therefore, compared to ECCs protected or unprotected implementation, the LBP scheme reduces the cost to achieve a better protection, so counterintuitive; thus, it is denoted as Less-is-Better. Additionally, LBP can be used in systems in which the underlying memory is unprotected, because no additional parity bits are needed. Therefore, LBP is an attractive option to protect the memory of *k*NNs classifiers.

5 EVALUATION

The proposed LBP scheme has been evaluated and compared to both Parity-Check protection and SEC-DED protection; all schemes have been assessed in terms of error protection as well as the memory required for implementation. The results are summarized in the next subsections. In all cases, the same datasets of Section 3 are used for the evaluation.

5.1 Error Protection

To assess the impact of errors on LBP-protected *k*NNs classifiers, errors have been injected in features and labels following the same procedure used in Section 3. The results are summarized in Table 8 that also provides for comparison purposes the values for the other techniques (including the unprotected, Parity-Check protected and SEC-DED protected *k*NNs classifiers). This table shows the percentage of errors that modify the classification result and in parenthesis, the value relative to the unprotected implementation (considered as the baseline 1x). For example, a value of (1.406x) means that the corresponding technique has a percentage of errors that modify the classification result that is 1.406x times of an unprotected implementation. Therefore, values lower than 1x mean that the technique can reduce the impact of errors, while values larger than 1x mean that the technique is worse than an unprotected implementation.

As per Table 8, LBP achieves better protection than Parity-Check for SEs. LBP is also able to reduce the impact of errors compared to an unprotected implementation for some datasets with a small number of features. For example, for the Iris dataset, the percentage of errors that modify the classification result, is reduced to 0.492x (i.e., reducing 59% the error rate compared to Parity-Check protection with 1.186x). This occurs because the LBP scheme protects against errors in the labels, which typically have the largest impact on the classification results (as per Observation 2 in Section 4). The impact is reduced significantly for the Iris dataset because it has a small number of features (i.e., four as per Table 1), then errors on labels

TABLE 8
Percentage of Different Type of Errors that Modify the Classification Result in the Unprotected *k*NNs and Different Protection Schemes (Relative Results of Protection Schemes to the Unprotected *k*NNs are Also Given in the Parenthesis)

Scheme		Pima Indian diabetes	Sonar	Banknote auth-entication	Phishing websites	Iris	Forest type mapping	Mice protein expression	CANE-9	Cervical cancer	Nursery
SEs	U	0.069% (*1x*)	0.048% (*1x*)	0 (*1x*)	0.004% (*1x*)	0.059% (*1x*)	0.112% (*1x*)	0.003% (*1x*)	0.052% (*1x*)	0.002% (*1x*)	0.002% (*1x*)
	P	0.097% (*1.406x*)	0.116% (*2.417x*)	0 (*1x*)	0.011% (*2.750x*)	0.070% (*1.186x*)	0.223% (*1.991x*)	0.006% (*2x*)	0.075% (*1.442x*)	0.004% (*2x*)	0.003% (*1.5x*)
	SEC-DED	0 (*0x*)	0 (*0x*)	0 (*0x*)	0 (*0x*)	0 (*0x*)	0 (*0x*)	0 (*0x*)	0 (*0x*)	0 (*0x*)	0 (*0x*)
	LBP	**0.049%** (*0.710x*)	**0.044%** (*0.917x*)	**0** (*1x*)	**0.004%** (*1x*)	**0.029%** (*0.492x*)	**0.109%** (*0.973x*)	**0.003%** (*1x*)	**0.052%** (*1x*)	**0.002%** (*1x*)	**0.002%** (*1x*)
DAEs	U	0.078% (*1x*)	0.079% (*1x*)	0 (*1x*)	0.006% (*1x*)	0.080% (*1x*)	0.133% (*1x*)	0.004% (*1x*)	0.066% (*1x*)	0.002% (*1x*)	0.002% (*1x*)
	P	0.080% (*1.026x*)	0.080% (*1.013x*)	0 (*1x*)	0.006% (*1x*)	0.084% (*1.050x*)	0.132% (*0.993x*)	0.004% (*1x*)	0.070% (*1.061x*)	0.002% (*1x*)	0.002% (*1x*)
	SEC-DED	0.125% (*1.603x*)	0.150% (*1.899x*)	0 (*1x*)	0.014% (*2.333x*)	0.091% (*1.138x*)	0.289% (*2.173x*)	0.008% (*2x*)	0.098% (*1.485x*)	0.006% (*3x*)	0.004% (*2x*)
	LBP	**0.060%** (*0.769x*)	**0.076%** (*1x*)	**0** (*1x*)	**0.006%** (*1x*)	**0.053%** (*0.663x*)	**0.129%** (*0.970x*)	**0.004%** (*1x*)	**0.066%** (*1x*)	**0.002%** (*1x*)	**0.002%** (*1x*)
DREs	U	0.090% (*1x*)	0.080% (*1x*)	0 (*1x*)	0.007% (*1x*)	0.087% (*1x*)	0.134% (*1x*)	0.003% (*1x*)	0.056% (*1x*)	0.003% (*1x*)	0.002% (*1x*)
	P	0.092% (*1.022x*)	0.082% (*1.025x*)	0 (*1x*)	0.009% (*1.286x*)	0.084% (*0.966x*)	0.139% (*1.037x*)	0.003% (*1x*)	0.060% (*1.071x*)	0.003% (*1x*)	0.002% (*1x*)
	SEC-DED	0.125% (*1.389x*)	0.150% (*1.875x*)	0 (*1x*)	0.014% (*2x*)	0.091% (*1.046x*)	0.289% (*2.157x*)	0.008% (*2.667x*)	0.098% (*1.750x*)	0.006% (*3x*)	0.004% (*2x*)
	LBP	**0.073%** (*0.811x*)	**0.077%** (*0.963x*)	**0** (*1x*)	**0.007%** (*1x*)	**0.062%** (*0.713x*)	**0.130%** (*0.970x*)	**0.003%** (*1x*)	**0.056%** (*1x*)	**0.003%** (*1x*)	**0.002%** (*1x*)

TABLE 9
Memory Required for an Unprotected *k*NNs and Different Protection Schemes
(Relative Results of Protection Schemes to the Unprotected *k*NNs are Also Given in Parenthesis)

Scheme	Pima Indian diabetes	Sonar	Banknote auth-entication	Phishing websites	Iris	Forest type mapping	Mice protein expression	CANE-9	Cervical cancer	Nursery
U	110592 (*1x*)	203008 (*1x*)	109760 (*1x*)	1218176 (*1x*)	12000 (*1x*)	145600 (*1x*)	1399680 (*1x*)	14808960 (*1x*)	507936 (*1x*)	1866240 (*1x*)
P	117504 (*1.063x*)	215696 (*1.063x*)	116620 (*1.063x*)	1294312 (*1.063x*)	12750 (*1.063x*)	154700 (*1.063x*)	1487160 (*1.063x*)	15734520 (*1.063x*)	539682 (*1.063x*)	1982880 (*1.063x*)
SEC-DED	152064 (*1.375x*)	279136 (*1.375x*)	150920 (*1.375x*)	1674992 (*1.375x*)	16500 (*1.375x*)	200200 (*1.375x*)	1924560 (*1.375x*)	20362320 (*1.375x*)	698412 (*1.375x*)	2566080 (*1.375x*)
LBP	**98352** (*0.889x*)	**199728** (*0.984x*)	**87856** (*0.800x*)	**1178928** (*0.968x*)	**9696** (*0.808x*)	**140544** (*0.965x*)	**1382736** (*0.988x*)	**14792064** (*0.999x*)	**494256** (*0.973x*)	**1659072** (*0.889x*)

also account for an important part (as per Observation 4 in Section 4). For DAEs and DREs, LBP provides better protection than Parity-Check and SEC-DED and reduces the percentage of errors that modify the classification results for some datasets in the unprotected case. Again, the Iris dataset is the one for which LBP has the largest benefit with a classification error rate of 0.663x and 0.713x of an unprotected implementation for DAEs and DREs respectively (i.e., reducing 39% for DAEs and 26% for DREs compared to Parity-Check protection, and 42% and 32% for SEC-DED protection).

These results show that LBP can improve protection against SEs compared to Parity-Check protected and unprotected *k*NNs classifiers, and it is also better than SEC-DED for DAEs and DREs.

5.2 Memory Overhead

In most cases, the encoder and decoder circuits of Parity-Check and SEC-DED based protection schemes are small compared to the additional memory, i.e. the hardware overhead is mostly due to the additional memory. Let the training set of a dataset used to select the *k*NNs have E elements and n_c classes; each element has f features, and the memory consists of w-bit words. In this subsection, the memory overhead in terms of total number of memory cells required for each protection scheme is evaluated and compared.

Consider an unprotected *k*NNs; as per the memory organization shown in Figure 2, the total number of memory cells is given by:

$$N_U = E \cdot (f + c) \cdot w \tag{4}$$

where c is the number of words that store the label of each element (c=1 for the datasets considered in this paper, because there are at most 11 classes).

When using a Parity-Check to protect the *k*NNs memory against single errors, an additional data bit is required in each word to store the parity bit. Therefore, the total number of memory cells needed for the Parity-Check protected *k*NNs is given by:

$$N_P = E \cdot (f + c) \cdot (w + 1) \tag{5}$$

When SEC-DED codes are used to protect the *k*NNs memory against double errors, r parity bits (as discussed in Section 2.2) are needed to be stored with the data bits in each memory word. Therefore, in this case, the required number of memory cells is given by:

$$N_{SEC-DED} = E \cdot (f + c) \cdot (w + r) \tag{6}$$

In the proposed LBP scheme, labels of the elements do not need to be stored in the memory; n_c-1 class pointers that mark each range of classes are required instead. Moreover, to provide error tolerance for the pointers, TMR can be implemented to deal with any error in one pointer; hence 3(n_c-1) words are needed for the entire memory in the LBP scheme. Therefore, the total number of memory cells is given by:

$$N_{LBP} = E \cdot f \cdot w + 3(n_c - 1) \cdot w \tag{7}$$

Since the number of elements is significantly larger than the number of their classes (i.e., $E >> n_c$), the first term of Eq. (7) is dominant. Therefore, compared to the Parity-Check and SEC-DED based protection schemes, the proposed LBP scheme incurs in a lower memory overhead; this is also the case when compared to the unprotected *k*NNs.

For all datasets considered in this paper, the number of memory cells needed in the different schemes are shown and compared in Table 9 (for w=16 as example, and thus r=6). The proposed LBP can reduce up to 20% the memory overhead of an unprotected *k*NNs, 25% of the Parity-Check protected *k*NNs, and 42% of the SEC-DED protected *k*NNs.

5.3 Comparison to Selective ECCs Protection

Since discarding the incorrect element tends to have a larger impact on the classification result than leaving it in the dataset due to LSBs of features having a negligible effect as discussed previously, it is interesting to also evaluate the error tolerance capability of selective ECCs solutions that only cover several MSBs.

Table 10 presents the percentage of errors that modify the classification results by using ECCs against errors (Parity-Check against SEs and SEC-DED codes against DAEs and DREs) in different solutions: ECCs that protect features only covering the 4MSBs, 8MSBs and 12MSBs of each word. Results in Table 10 show that the percentage of SEs and DAEs that modify the classification results increases with more MSBs covered by ECCs, because the impact is low on LSBs as discussed previously. The situation for DREs is more complex. There are three cases under DREs, including: i) both the double errors occur on

TABLE 10
Percentage of Different Type of Errors that Modify the Classification Result in the Selective ECCs Protection Scheme

Schemes*		Pima Indian diabetes	Sonar	Banknote auth-entication	Phishing websites	Iris	Forest type mapping	Mice protein expression	CANE-9	Cervical cancer	Nursery
SEs	P_4MSBs	0.055%	0.054%	0	0.005%	0.037%	0.109%	0.003%	0.054%	0.002%	0.002%
	P_8MSBs	0.058%	0.066%	0	0.006%	0.045%	0.126%	0.003%	0.055%	0.002%	0.002%
	P_12MSBs	0.075%	0.087%	0	0.008%	0.056%	0.169%	0.005%	0.061%	0.003%	0.002%
	U	0.069%	0.048%	0	0.004%	0.059%	0.112%	0.003%	0.052%	0.002%	0.002%
	LBP	**0.049%**	**0.044%**	**0**	**0.004%**	**0.029%**	**0.109%**	**0.003%**	**0.052%**	**0.002%**	**0.002%**
DAEs	SD_4MSBs	0.073%	0.080%	0	0.007%	0.062%	0.150%	0.004%	0.067%	0.003%	0.002%
	SD_8MSBs	0.080%	0.090%	0	0.008%	0.067%	0.169%	0.005%	0.072%	0.004%	0.003%
	SD_12MSBs	0.103%	0.121%	0	0.011%	0.077%	0.233%	0.006%	0.085%	0.005%	0.003%
	U	0.078%	0.079%	0	0.006%	0.080%	0.133%	0.004%	0.066%	0.002%	0.002%
	LBP	**0.060%**	**0.076%**	**0**	**0.006%**	**0.053%**	**0.129%**	**0.004%**	**0.066%**	**0.002%**	**0.002%**
DREs	SD_4MSBs	0.092%	0.083%	0	0.007%	0.071%	0.157%	0.004%	0.072%	0.004%	0.002%
	SD_8MSBs	0.083%	0.084%	0	0.007%	0.067%	0.157%	0.004%	0.071%	0.004%	0.002%
	SD_12MSBs	0.092%	0.116%	0	0.011%	0.085%	0.222%	0.006%	0.083%	0.005%	0.003%
	U	0.090%	0.080%	0	0.007%	0.087%	0.134%	0.003%	0.056%	0.003%	0.002%
	LBP	**0.073%**	**0.077%**	**0**	**0.007%**	**0.062%**	**0.130%**	**0.003%**	**0.056%**	**0.003%**	**0.002%**

* P_4MSBs (SD_4MSBs), P_8MSBs (SD_8MSBs) and P_12MSBs (SD_12MSBs) refer to the selective Parity-Check (SEC-DED codes) protection by covering 4MSBs, 8MSBs and 12MSBs, respectively.

the protected MSBs, so the incorrect element is discarded, causing a large impact on the classification results; ii) one error affects one protected MSBs (which will be corrected by using the SEC-DED codes) and the other affects one unprotected LSB, causing a small impact on the results; iii) both bit errors occur on the unprotected LSBs, causing also a small impact. Therefore, the selective protection with 12 MSBs covered tends to have a large impact due to the dominance of case i). For the solution with 8MSBs (in which all of the three cases occur with an equal probability) and with 4MSBs covered (in which case iii) is dominant but some unprotected upper LSBs still have a large impact), the results depend on different datasets thus the trend is not clear. However, as shown in Table 10, the proposed LBP scheme still provides a higher error tolerance capability. Additionally, since no parity bits are introduced and label words are saved in the LBP scheme, it also achieves lower memory overhead.

6 Conclusion

Machine learning (ML) has been widely used to perform classification tasks. The components of a ML implementation are prone to suffer from errors that may change the classification results. Therefore, efficient error-tolerant techniques must be employed, especially when the classifiers are used in safety or critical applications, otherwise errors may potentially cause life/property loss. In this paper, the impacts of errors on the memory of *k*NNs classifiers have been considered. Initially, errors have been injected to evaluate their effects on the classification results when using an unprotected, a Parity-Check protected and an SEC-DED protected memory that stores *k*NNs elements. The results have shown that for single bit errors, it is better to leave the memory unprotected than to use Parity-Check protection and discard the erroneous element. The same occurs for double adjacent and double random bit errors in SEC-DED protection; it is better not to use double error detection and discard erroneous elements. This is an interesting result, because the provision of protection to detect errors is counterproductive. Therefore, Less-is-Better Protection (LBP) has been proposed based on observations obtained from such evaluation. LBP leaves the memory unprotected except for the label that is stored implicitly by storing elements of the same class consecutively and using pointers to identify the classes.

The proposed LBP scheme has been evaluated and compared to both Parity-Check protection and SEC-DED protection. The results have shown that LBP outperforms Parity-Check protection (SEC-DED protection) in terms of protection against single bit errors (double bit errors) using no parity bit (e.g., reducing 59% of single bit errors that modify the classification results for the Iris dataset, while saving 24% of the memory compared to Parity-Check protection, and 42% of the impact of double bit errors while 41% of the memory compared to SEC-DED protection). This is also applicable to a comparison with an unprotected implementation because no labels are stored.

Overall, LBP achieves better protection than traditional techniques, while reducing the memory overhead. It also reduces some memory requirements compared to the unprotected scheme. Moreover, LBP does not need any change to the underlying memory and thus, it can be used in systems that use unprotected memories. These advantages make the LBP a very efficient scheme to protect the memory of *k*NNs and make *k*NNs more attractive, because it reduces the original memory requirements.

Acknowledgment

S. Liu and F. Lombardi would like to acknowledge the support of NSF grants CCF-1953961 and 1812467, and P. Reviriego would like to acknowledge the support of the ACHILLES project PID2019-104207RB-I00 and the

Go2Edge network RED2018-102585-T funded by the Spanish Ministry of Science and Innovation and by the Madrid Community research project TAPIR-CM P2018/TCS-4496.

REFERENCES

[1] OECD, "Artificial Intelligence in Society", OECD Publishing, Paris, https://doi.org/10.1787/eedfee77-en, 2019.

[2] G. D. L. Torre, P. Rad, K. R. Choo, "Driverless Vehicle Security: Challenges and Future Research Opportunities", *Elsevier, Future Generation Computer Systems*, vol. 108, pp. 1092-1111, 2020.

[3] D. H. Park, H. K. Kim, I. Y. Choi, et al, "A Literature Review and Classification of Recommender Systems Research", *Elsevier, Expert Systems with Applications*, vol. 39, no.11, pp. 100059-100072, 2012.

[4] A. Fhoneim, F. Muhammad, M. S. Hossain, "Cervical Cancer Classification using Convolutional Neural Networks and Extreme Learning Machines", *Elsevier, Future Generation Computer Systems*, vol. 102, pp. 643-649, 2020.

[5] V. Vargas, P. Ramos, V. Ray, et al, "Radiation Experiments on a 28 nm Single-Chip Many-Core Processor and SEU Error-Rate Prediction", *IEEE Transactions on Nuclear Science*, vol. 64, no. 1, pp. 483-490, 2017.

[6] R. Baumann, ''Soft Errors in Advanced Computer Systems,'' *IEEE Design and Test of Computers*, vol. 22, no. 3, pp. 258–266, 2005.

[7] W. M. Shaban, A. H. Rabie, A. I. Saleh, et al, "A New COVID-19 Patients Detection Strategy (CPDS) based on hybrid feature selection and enhanced KNN Classifier", Elsevier, *Knowledge-Based Systems*, vol. 205, no. 12, pp. 1-18, 2020.

[8] H. Frigui and P. Gader, "Detection and Discrimination of Land Mines in Ground-Penetrating Radar Based on edge Histogram Descriptors and a Possibilistic K-Nearest Neighbor Classifier", *IEEE Transactions on Fuzzy Systems*, vol. 17, no. 1, pp. 185-199, 2009.

[9] A. Altaher, "Phishing Websites Classification using Hybrid SVM and KNN Approach", *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 6, pp. 90-95, 2017.

[10] J. G. Lopez, S. Ventura, A. Cano, "Distributed Nearest Neighbor Classification for Large-Scale Multi-Label Data on Spark", *Elsevier, Future Generation Computer Systems*, vol. 87, pp. 66-82, 2018.

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

[12] Z. Shao, D. Taniar, K. M. Adhinugraha, "Voronoi-based Range-*k*NN Search with Map Grid in a Mobile Environment, *Elsevier, Future Generation Computer Systems*, vol. 67, pp. 305-314, 2017.

[13] H. Frigui, P. Gader, "Detection and Discrimination of Land Mines in Ground-Penetrating Radar Based on Edge Histogram Descriptors and a Possibilistic *k*-Nearest Neighbor Classifier", *IEEE Transactions on Fuzzy Systems*, vol. 17, no. 1, pp. 185-199, 2009.

[14] W. Wang, Y. Li, X. Wang, et al, "Detecting Android Malicious Apps and Categorizing Benign Apps with Ensemble of Classifiers", *Elsevier, Future Generation Computer Systems*, vol. 78, pp. 987-994, 2018.

[15] D. P. Siewiorek., R. S. Swarz, "'The Theory and Practice of Reliable System Design," Digital Press, Bedford, 1982.

[16] K. Chen, L. Chen, P. Reviriego, et al, "Efficient Implementations of Reduced Precision Redundancy (RPR) Multiply and Accumulate (MAC)", *IEEE Transactions on Computers*, vol. 68, no. 5, pp.784-790, 2018.

[17] B. Shim, S. R. Sridhara, N. R. Shanbhag, "Reliable Low-Power Digital Signal Processing via Reduced Precision Redundancy", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol.12, no.5, pp.497-510, 2004.

[18] S. Liu, P. Reviriego, J.A. Hernández, et al, "Voting Margin: A Scheme for Error-Tolerant *k* Nearest Neighbors Classifiers", *IEEE Transactions on Emerging Topics in Computing*, 2019 (Early Access).

[19] S. Liu, P. Reviriego, P. Montuschi, et al, "Error-Tolerant Computation for Voting Classifiers with Multiple Classes", *IEEE Transactions on Vehicular Technology*, vol. 69, no. 11, pp. 13718-13727, 2020.

[20] S. Liu, P. Reviriego, P. Montuschi, et al., "Results-Based Re-Computation for Error-Tolerant Classifiers by a Support Vector Machine", IEEE Transactions on Artificial Intelligence, vol. 1, no. 1, pp. 62-73, 2020.

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

[22] E. Fujiwara, "Code Design for Dependable Systems: Theory and Practical Applications", Wiley-Interscience, 2006.

[23] H. Wang, "Nearest Neighbors by Neighborhood Counting", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 6, pp. 942-953, 2006.

[24] S. Baeg, S. Wen, R. Wong, "SRAM Interleaving Distance Selection with A Soft Error Failure Model," *IEEE Transactions on Nuclear Science*, vol. 56, no. 4, pp. 2111–2118, 2009.

[25] J. L. Autran, D. Munteanu, P. Roche, et al, "Soft-Errors Induced by Terrestrial Neutrons and Natural Alpha-Particle Emitters in Advanced Memory Circuits at Ground Level", Elsevier, Microelectronics Reliability, vol. 50, pp. 1822-1831, 2010.

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

[27] 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.

[28] 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.

[29] B. Johnson, R. Tateishi, Z. Xie, "Using Geographically-Weighted Variables for Image Classification", *Remote Sensing Letters*, vol. 3, no. 6, pp. 491-499, 2012.

[30] C. Higuera, K.J. Gardiner, 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.

[31] K. Fernandes, J. S. Cardoso and J. Fernandes, "Transfer Learning with Partial Observability Applied to Cervical Cancer Screening", *Iberian Conference on Pattern Recognition and Image Analysis*, Springer, Cham, 2017.

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

[33] P. Reviriego, S. Pontarelli, J.A. Maestro, et al, "Reducing the Cost of Implementing Error Correction Codes in Content Addressable Memories", *IEEE Transactions on Circuits and Systems II*, vol. 60, no. 7, pp. 432-436, July 2013.

Shanshan Liu (M'19) received the M.S. degree and Ph.D. degree 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 Engineer with Teldat, Madrid, working on router implementation. In 2000, he joined Massana to work on the development of 1000BASE-T 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.

Paolo Montuschi (M'90-SM'07-F'14) is a Full Professor in the Department of Control and Computer Engineering and a Member of the Board of Governors at Politecnico di Torino, Italy. His research interests include computer arithmetic and architectures, computer graphics, electronic publications. He is an IEEE Fellow, and an IEEE Computer Society (CS) Golden Core member. He is currently serving as the 2017-20 IEEE Computer Society Awards Chair, as a Member-at-Large of the Publication Services and Products Board (PSPB) (2018-20), and as the Chair of its Strategic Planning Committee (2019-20). He is serving as the 2020-21 Chair of the IEEE TAB/ARC (TAB/Awards and Recognitions Committee), as a Member of the IEEE Awards Board, as a Member (2020) of the IEEE PRAC (Periodicals Review and Advisory Committee), and as a Vice Chair of the 2020 Computer Society Fellows Committee, Previously, he served, among all, as the Editor-in-Chief of the IEEE Transactions on Computers, and as the 2019 Acting (interim) Editor-in-Chief of the IEEE Transactions on Emerging Topics in Computing. He is a life member of the International Academy of Sciences of Turin and of Eta Kappa Nu (the Honor Society of IEEE). In March 2017 he co-founded the first HKN Student Chapter in Italy and in Europe, Chapter.

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, IEEE TRANSACTIONS ON NANOTECHNOLOGY from 2014 to 2019. He is currently the Vice President for Publications of the IEEE Computer Society, the 2021 President-elect of the IEEE Nanotechnology Council and a member of the IEEE Publication Services and Products Board.