Knowledge and Information Systems (2021) 63:1577–1596
https://doi.org/10.1007/s10115-021-01565-5

REGULAR PAPER

Advancing synthesis of decision tree-based multiple classifier systems: an approximate computing case study

Mario Barbareschi[1] · Salvatore Barone[1] · Nicola Mazzocca[1]

Received: 24 February 2020 / Revised: 15 March 2021 / Accepted: 22 March 2021 / Published online: 12 April 2021

Abstract
So far, multiple classifier systems have been increasingly designed to take advantage of hardware features, such as high parallelism and computational power. Indeed, compared to software implementations, hardware accelerators guarantee higher throughput and lower latency. Although the combination of multiple classifiers leads to high classification accuracy, the required area overhead makes the design of a hardware accelerator unfeasible, hindering the adoption of commercial configurable devices. For this reason, in this paper, we exploit approximate computing design paradigm to trade hardware area overhead off for classification accuracy. In particular, starting from trained DT models and employing precision-scaling technique, we explore approximate decision tree variants by means of multiple objective optimization problem, demonstrating a significant performance improvement targeting field-programmable gate array devices.

Keywords Approximate computing · Decision tree · Multiple classifier systems · FPGA · MOP · Genetic algorithm

1 Introduction

Nowadays, digital applications are experimenting an unprecedented growth of data to be processed and, jointly, require high efficiency in terms of computational time, power consumption, and so on. The inadequacy of classical computing system design approaches to tackle with these requirements has been already identified as one of the main Big Data challenges.

Scientific literature is mainly focusing on pattern recognition, machine learning and data classification systems since, from one side, Big Data domain gives the possibility to access

✉ Salvatore Barone
Salvatore.Barone@unina.it

Mario Barbareschi
Mario.Barbareschi@unina.it

Nicola Mazzocca
Nicola.Mazzocca@unina.it

1 Department of Electrical Engineering and Information Technology , University of Naples Federico II, Naples, Italy

to heterogeneous and huge datasets, allowing for new applications, from the other side, researches investigate about technological innovations and new methodologies to overcome performance issues, in terms of both model accuracy and timing (computing throughput and latency). As for the former, accuracy plays a fundamental role since even a small percentage of classification inaccuracy affects a very great amount of processing data samples and, for this reason, new learning algorithms are being proposed, such as classifier systems based on the combination of multiple models [1–3].

As for the latter, Van Essen et al. [4] investigated the possibility of using FPGA technology to accelerate random forest-based classifiers, proving that this technology performs significantly better than multi-core processors or GP-GPUs; thus, a considerable effort has been made to exploit hardware accelerators for getting better performance compared to software implementations.

For instance, this is the case of hardware acceleration of decision trees (DTs) classification systems based on the field-programmable gate array devices (FPGAs) [5–9].

Hardware design of multiple classification systems (MCSs) enables to take both advantages, clearly high accuracy and better computational performance, as already demonstrated by authors of [10]. There are many applications that can benefit from the performance of hardware accelerators for classification. An example is the data stream mining applications [11] in which, unlike traditional data mining applications where the data are static and can be repeatedly read many times, constraints such as bounded memory, single-pass, real-time response have to be satisfied. There are also many Fog Computing applications that make use of machine learning models to the peripheral nodes of the network. In [12], for example, a fog computing topology for delivering real-time embedded machine learning features is proposed.

However, hardware design of MCSs hides some design issues, mainly related to the highly area required by logic circuits that implement the whole classifier. Indeed, while a single classification system can be someway adapted to fit onto a feasible hardware device, MCS would require to adapt not only a set of single models, but additionally the combiner needed to merge classification outcomes into the proper classification result.

In order to deal with this issue, in this paper we propose a hardware synthesis methodology that exploits approximate computing (AxC) design paradigm: by renouncing some classification accuracy, AxC is able to reduce hardware overhead with respect to full-accurate system.

In particular, starting from a trained DT model—encoded in the Predictive Model Markup Language (PMML) [13]—we manipulate it in order to produce hardware accelerators, and then, we prove that DT-based MCSs can be successfully approximated by the precision-scaling technique and exploration of approximate DTs variants can be suitably solved by means of a heuristic multi-objective optimization problem (MOP) approach. Experimental evidences, conducted over a significant dataset, highlight the efficacy of the approach by exploring approximate DTs variants by means of a genetic algorithm (GA) and by synthesizing hardware accelerator on a Xilinx Zynq 7020 FPGA device. Additionally, we compare our experimental result against exhaustive branch and bound approach proposed in [14] demonstrating a reduction in area occupancy of about 10%.

The remainder of the paper is structured as follows: Sect. 2 gives a brief overview about AxC and scientific contribution about hardware implementation of DT-based MCSs, while Sect. 3 gives details of the hardware architecture that realizes the DT-based MCS system. In particular, the proposed approach for MCS hardware implementation is discussed, highlighting how the AxC design paradigm can be used to reduce the requirements of such an implementation both in terms of silicon area and power consumption. Section 4 formalizes the approximate DT variants exploration problem as a MOP, and in Sect. 5, we report the experimental result. Finally, Sect. 6 draws the conclusion.

2 Scientific background

The scientific literature demonstrated that imprecise calculations can be selectively exploited to enhance computing system performance, defining the AxC paradigm [15]. Indeed, due to redundancy of inner calculations, some applications are characterize by an inherent resiliency to errors. Basically, relaxing functional requirements of a computing system, AxC enables to trade output accuracy off for performance, such as calculation speed, throughput and, for integrated circuits (ICs), occupied area.

Since a naive approximation approach, such as uniform approximation, is unlikely to be efficient, different AxC techniques have been proposed [16]. Some examples are precision-scaling, loop perforation [17], memoization [18], functional approximation, and so forth. In particular, precision-scaling for input data and intermediate operands has been proposed to improve efficiency for floating-point computation in many scientific applications [19–23].

Leveraging the full potential of AxC, however, requires addressing several challenges. First of all, output quality monitoring (i.e., introduced error) must be carefully taken into account: it is mandatory to ensure that application requirements are met [24]; hence, quality metrics must be properly selected for error assessment. The selection of appropriate metrics to be used for the error estimation is not a trivial task, as they are application-dependent and workload-dependent [16]. Moreover, the measurement of the error is typically done by running the whole approximate application, or by simulations, which may require significant effort. A different approach, based on a-priori estimation using Bayesian inference, has been proposed by Traiola et al. [25,26].

Selecting a particular approximate configuration of a given application, generated by a given technique, is a major challenge. AxC techniques, in facts, may generate different approximate versions of the same application. For instance, let us consider loop-perforation [17]: the amount of skipped iterations—i.e., the approximate configuration of the target application—must be properly configured in order to find a good trade-off between introduced error and performance gains. In addition, among every realizable approximate version, only those characterized by an error that falls below a user-defined error-threshold must be taken in consideration.

2.1 Decision tree-based hardware classifiers

Classification systems are one of those applications that are characterized by inherent error resiliency, as models are retrieved by means of iterative training algorithms exploiting large datasets [24]. This is even true for multi-classification systems (MCSs), such as random forest classifier, which is well-known machine learning technique in which an ensemble of decision trees is used to assign a label (or classification) to an input sample [27].

Let us now briefly introduce how DTs are employed as classification system. DTs are tree-like predictive models in which each internal node specifies a test on a given variable, namely model feature, each branch a possible test outcome and each leaf gives information about decision outcome, namely predicted class. Models in which target variables can take a discrete set of values are called classification trees, while regression trees assign leaves to a probability distribution.

Algorithms for constructing DTs usually work top-down, by choosing a variable at each step that best splits the set of samples of a training set [28]. One of the most adopted training algorithm for DTs is C4.5 proposed in [29]. It constructs, by means of an inductive approach, classification models from training databases, following a top-down and divide-and-conquer paradigm. At each step of training algorithm, the dataset is split, based on conditions defined upon a chosen feature. The feature selection involves an entropy test that establishes which feature inducts the best partitioning onto dataset. The construction recursively continues until a leaf is reached. As for testing, DT prediction algorithm performs a recursion too, as explained lately in Sect. 4.

So far, scientific literature has posed a huge effort for researching new design methodologies in order to improve both rate and accuracy of classification systems. Indeed, as for the former, new classification architectures have been proposed, mainly based on custom hardware acceleration.

The authors of [5,6] proposed an automatic methodology to generate hardware implementation of DT-based classifiers. The proposed methodology consists of three phases: (i) structuring the data coming from different sources into a single schema; (ii) using the schema to model a predictor by exploiting the C4.5 algorithm [29]; (iii) automatically converting the DT model to VHDL hardware accelerator. They demonstrated that the approach performs dramatically better than a pure software solution, guaranteeing a significant high classification throughput implementing the DT prediction onto an FPGA. As for the latter, novel training techniques, such as multi-classification systems, have been devised to deal with accuracy. Van Essen et al. [4] quantified the performance, power, and cost of DT-based MCS, trained by means of random forest algorithm, implemented over CPUs, GP-GPUs and FPGAs. The FPGA implementation outperforms the other two solutions, accordingly to results of [5], both in terms of classification rate and power dissipation, measured in classifications per second per watt consumed.

Authors of [7,8,10] proposed a novel approach for an efficient hardware implementation of MCSs: parallel classification entities—for instance, DTs—execute classification in parallel, and then, an hardware combiner organizes outputs from different DT classifiers in order to make the final decision. DT nodes work in parallel and each one is implemented as a binary comparator: once it receives the feature value, it returns a Boolean value that, in turn, is fed to a Boolean network in order to compute which leaf of the tree has been reached.

2.2 Approximate classifiers

The adoption of hardware DT MCSs is actually hindered by scalability issue, as reported in [4], and AxC techniques are mainly devoted to other classification systems.

A technique to improve energy efficiency of machine-learning classifier has been proposed by Venkataramani et al. [30]. Having noticed that, typically, only a part of the data of a given dataset really needs the full computational power of a classifier, they dynamically configure the classifier making it more or less accurate, according to the difficulty in classifying the inputs. During the training phase, instead of building one complex decision model, a cascade, or series of models with progressively increasing complexity is constructed. During the testing phase, the number of decision models applied to a given input varies depending on the difficulty of the considered input instance. Inputs are processed by classifiers in a sequential way, starting from the less accurate classifier. In order to estimate the difficulty of a certain input, a confidence level for each classification is computed. If the estimated confidence falls above a certain threshold, the classification process is terminated; otherwise, a more accurate classifier is used. The experimental results presented by them show a significant reduction in energy consumption.

In Nepal et al. [31], the precision-scaling technique has been used to reduce the power consumption of a *perceptron* classifier [32,33]. Starting from a behavioral description of the classifier to be approximate, several different hardware configurations have been generated, by making use of the Automated Behavioral Synthesis of Approximate Computing Systems (ABACUS) tool. Then, the Pareto front, which consists of configurations providing optimal trade-off between accuracy and gains, is computed by making use of an iterative stochastic greedy algorithm. Authors claim a 33% reduction in energy consumption, with an accuracy of 83%.

A very practical approach has been adopted by the authors of [34] and [35]. They replaced multipliers and adders of a support vector machine (SVM) classifier [36] to introduce approximation in it. In Van Leussen et al. [34], the exact Karnaugh multiplier needed by the classifier has been replaced by an inaccurate Karnaugh multiplier (IKM) [37], which shows a uniform error distribution over the entire range of input. The classifier has been, then, synthesized on a 28nm CMOS technology, in order to estimate its energy requirements and accuracy. Authors claim a saving of 14% for silicon area and a saving of 61% for power consumption, while maintaining the same classification accuracy. In Zhou et al. [35] a new approximate adder and a new approximate multiplier are proposed. In order to show the full potential of the arithmetic units being proposed, both the exact adder and multiplier needed by an SVM classifier are replaced by approximate versions of them. The resulting classifier synthesized on a 90nm CMOS technology exhibited an area reduction of 18%, and simulations showed energy consumption reduction of 32%, while keeping accuracy of 95%.

3 Hardware architecture of DT-based MCS

In order to implement MCSs, our approach replicates the one proposed by authors of [10]. Multiple DTs are used simultaneously for greater classification accuracy. The outcomes of all the DTs are evaluated by a majority voter: the final outcome will be the class predicted by the majority of the DTs. Section 3.1 details the hardware implementation of the DT visiting algorithm, while Sect. 3.2 discusses the concepts behind the hardware implementation of the majority voter used to choose the winning class.

3.1 Implementing DTs on hardware

In order to speed up DTs visiting, in this paper we adopt a speculative approach, proposed by authors of [6], that takes advantage of the inherent parallelism of the hardware. The speculative approach consists in a DT flattening so that the visiting is performed over every possible path. In particular, each DT node contains a condition that establishes if the visiting has to continue on left sub-tree or on right sub-tree, until a leaf is reached. Instead, in the speculative approach, predicates are performed concurrently, regardless of the position and depth at which nodes are located: a Boolean decision variable, which indicates whether a condition is fulfilled, is produced for each one of the evaluated predicates. In order to determine which leaf of the DT is reached, i.e., which class the input belongs to, a Boolean function, called *assertion*, is defined for each different class. Since a path that leads to a specific leaf is obtained by computing the logic-AND between the Boolean decision variables along that path, and since it is possible to compute the logic OR between the conditions related to different paths leading

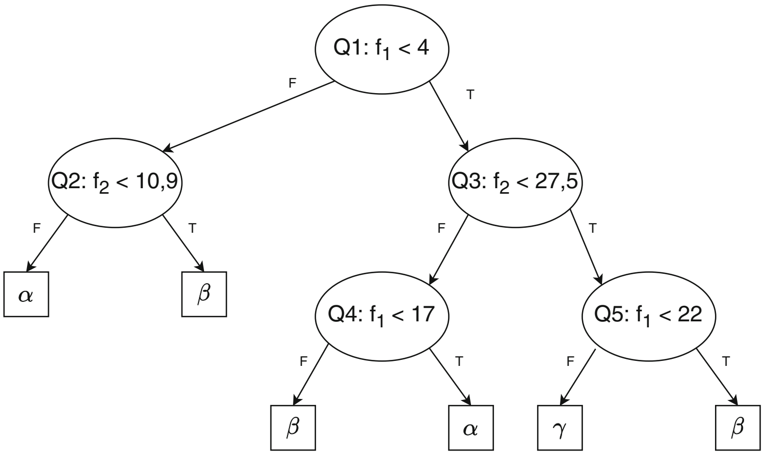


Fig. 1 An example of decision tree

to leaves belonging to the same class, assertions can be defined as a sum of products Boolean functions. For the sake of clarity, let us consider the DT depicted in Fig. 1, which evaluates two features in order to assess which one of three classes the inputs belong to. Starting from the root node, descending the DT and visiting nodes from the left to the right, the Boolean decision variables involved in the classification process are Q1, which is produced at the root node, Q2 produced at the $f_2 < 10.9$ node, and so on. Let us consider the α class: an input vector belongs to it if $f_1 \geq 4$—Q1 is false—and $f_2 \geq 10.9$—Q2 is false—or $f_1 < 4$—Q1 is true—**and** $f_2 \geq 27.5$—Q3 is false—and $f_1 < 17$—Q4 is true. In Eq. 1, we report Boolean assertions for all the classes.

$$\begin{aligned}\alpha &= (\overline{Q1} \wedge \overline{Q2}) \vee (Q1 \wedge \overline{Q3} \wedge Q4)\\ \beta &= (\overline{Q1} \wedge Q2) \vee (Q1 \wedge Q3 \wedge Q5) \vee (Q1 \wedge \overline{Q3} \wedge \overline{Q4})\\ \gamma &= Q1 \wedge Q3 \wedge \overline{Q5}\end{aligned} \tag{1}$$

Predicates are evaluated using decision boxes (DBs), i.e., comparators, while the visiting algorithm can be performed as a multi-output Boolean function. A comprehensive block schema is depicted in Fig. 2.

As proposed by Amato et al. [5,6], the hardware circuit can be automatically synthesized starting from the training dataset. The predictor model, coded in PMML [13], is obtained from a labeled dataset by making use of the KNIME [38] tool. Then, in order to perform FPGA synthesis, the model is translated in VHDL, using the PMML2VHDL tool [5,6]. This tool generates a DT for each different predicate to be evaluated. Moreover, the tool makes use of the Berkeley SIS tool to produce an optimized version of the assertion functions.

The scalability of this approach has been formally demonstrated in [8]. In particular, the number of literals in each assertion is always less or equal to twice the size of the features set.

3.2 Hardware combiner for class selection

Outcomes of the assertion functions belonging to the same class but computed by different DTs are arranged in an array of N elements, with N being the number of DTs. A majority voter is used to state which class is the winner.

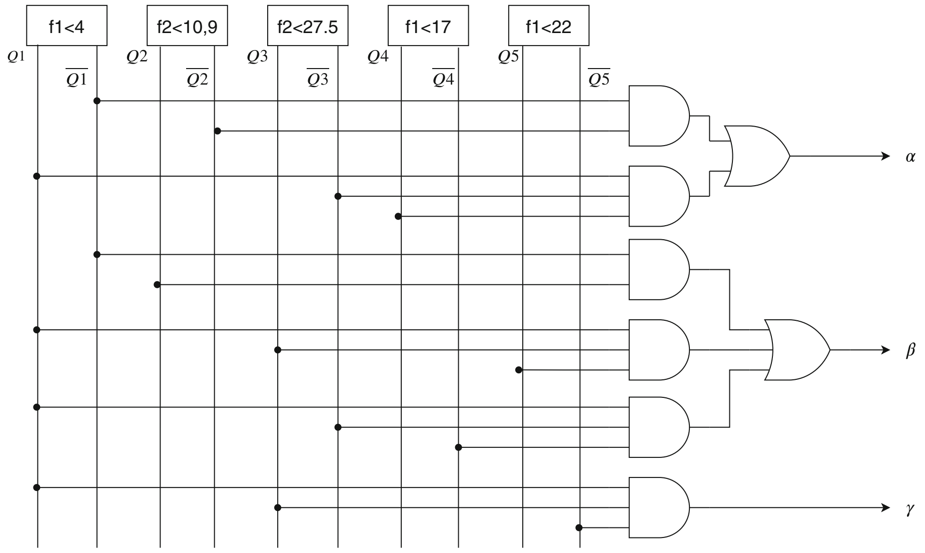


Fig. 2 Hardware implementation of a decision tree

Let $d_{i,j}$ be the preference expressed by the $i-th$ DT for the $j-th$ class, i.e., $d_{i,j}$ is a Boolean variable being equal to 1 i.f.f. the classifier input has been recognized by the $i-th$ DT to belong to the $j-th$ class; the following matrix can be defined:

$$\mathbb{D} = \begin{bmatrix} d_{0,0} & d_{0,1} & \cdots & d_{0,M-1} \\ d_{1,0} & d_{1,1} & \cdots & d_{1,M-1} \\ \vdots & \vdots & \ddots & \vdots \\ d_{N-1,0} & d_{N-1,1} & \cdots & d_{N-1,M-1} \end{bmatrix} \tag{2}$$

We define $p_j = \sum_{i=0}^{N-1} d_i, j,\ 0 \leq j < M$. Since each DT expresses just one preference (i.e., $\sum_{j=0}^{M-1} d_{i,j} = 1\ 0 \leq i < N$), it follows that the class w is the most voted i.f.f. $p_w > p_j,\ \forall j \neq w$, while we get a draw condition i.f.f. $\exists\{i, j\}$ s.t. $p_i = p_j = max_{0 \leq k < M}\{p_k\}$.

Rather than using binary adders to state which class gets the highest score, the majority voter sorts each column of the matrix $\mathbb{D}$ using a parallel sorting algorithm, pretty much like bubble-sort, by shifting all the high bits at the beginning of each column. This process is performed by exploiting a Boolean circuit called the *sorting network*, which depth is equal to n.

Let us consider a two bits array: Table 1 reports the truth table of a two bits sorting network. It is easy to recognize that $y_0 = x_0 \vee x_1$ and $y_1 = x_0 \wedge x_1$. Conversely, defining n-bits sorting network is cumbersome. However, such a network can be built using multiple two-bit sorting networks arranged in a n-stages pipeline, with even stages consisting in $N/2$ two-bit sorters—each of which compares array elements starting from even positions—and odd stages consisting in $N/2 - 1$ two-bit sorters—each of which compares array elements starting from odd positions [10]. The sorting networks need at least $N/2$ clock cycles to provide sorted arrays. An example of such a network is provided in Fig. 3.

Table 1 Truth table of a one-bit sorting network

x_1	x_0	y_1	y_0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

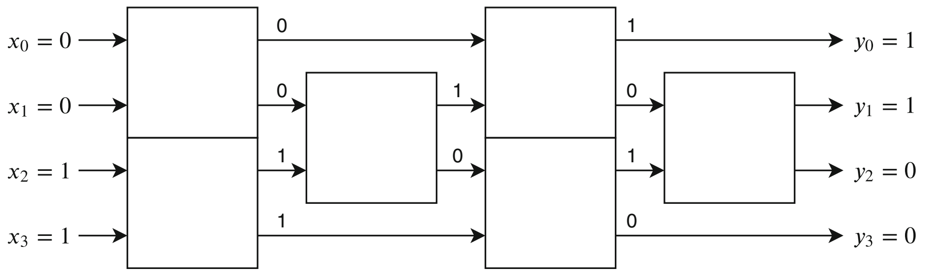


Fig. 3 Four-bit sorting network

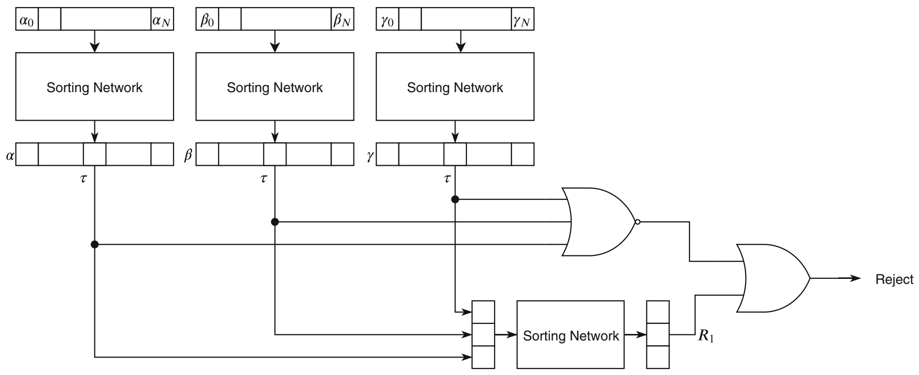


Fig. 4 Detailed block schema of the rejection module

Once the votes are sorted, the score each class has received needs to be verified. Let us define a threshold indicator as follows:

$$\tau_{i,j} = \begin{cases} 1 & i \le p_j \\ 0 & i > p_j \end{cases}, \quad 2 \le i \le N/2 \tag{3}$$

Hence, to detect the most voted class we need to find the $\tau_{i,j} = 1$ with the highest i: in case it is unique, we get a voted class, otherwise we got a draw. Exploiting the same sorting network used before, we can easily detect these two conditions.

Figure 4 shows an example of such a module for a three-class classifier.

4 Towards approximate DTs

The speculative hardware implementation of DT-based classifiers discussed in the previous section offers several opportunities for approximation.

Assertion functions, for instance, are good candidates but they typically involve only a few literals, so area savings may be negligible. Conversely, acting on comparators can lead to significant gains.

Concerning the approximate computing techniques, precision-scaling can be used to reduce the amount of bits required for representing model features: neglecting least significant bits of model features, while keeping the weight of the retained bits unaltered, leads to a reduction in the size of circuits due to the removal of parts of the logic needed by comparisons.

The impact of approximation on the classification accuracy can be assessed only through simulations and, in addition, among all the approximate configurations of a given classifier, only those providing a certain accuracy level can be taken into account for further considerations. Moreover, those configurations should be evaluated in terms of silicon area, in order to state which of them provides the best trade-off between performance and gains. This is a typical instance of a multi-objective optimization problem (MOP), because accuracy and area savings are conflicting objectives.

Although a full discussion on MOPs falls beyond the purpose of this paper, a brief introduction is reported in the following section.

4.1 Multi-objective optimization formalization

Multi-objective optimization (MO) is an area of operational research (OR) which concerns about mathematical optimization problems involving more than one objective function to be simultaneously optimized.

MO has a wide application field, which includes scientific, engineering, logistic or financial applications in which a compromise between two or more conflicting objective needs to be found.

Basically, MOP consists in a set

$$\gamma\left(\cdot\right) \;=\; \left\{\gamma_1\left(\cdot\right), \ldots, \gamma_k\left(\cdot\right)\right\}$$

of k different objective functions, or fitness functions, to be minimized. Typically a set

$$\psi\left(\cdot\right) \;=\; \left\{\psi_1\left(\cdot\right), \ldots, \psi_j\left(\cdot\right)\right\}$$

of j constraint functions defines the set of feasible solutions X, called the solution space. A vector $x^* \in X$, which minimizes each fitness function and does not violate any constraint function, is called a feasible solution. For non-trivial MOP, $|X| > 1$, where $|\cdot|$ expresses the size of a set, i.e., the number of elements it contains. Considering two different solutions, $x, y \in X$, the solution x is said to *Pareto dominate* y if and only if

$$\gamma_i\left(x\right) \leq \gamma_i\left(y\right) \quad \forall i \in [1, k]$$

and

$$\exists \; j \in [1, k] \quad | \quad \gamma_j\left(x\right) < \gamma_j\left(y\right)$$

If a solution is not dominated by any other solution belonging to the same space, it is called a *Pareto optimal* solution. All Pareto optimal solutions are considered equally good.

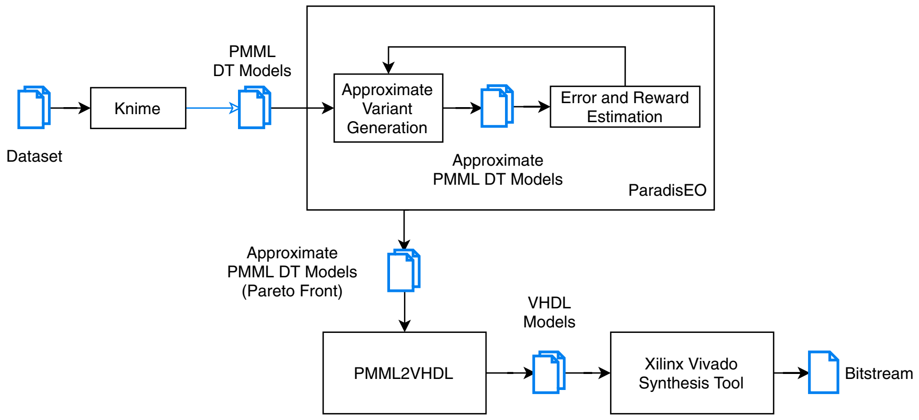


Fig. 5 Experimental flow to get FPGA bitstream from the considered dataset

Since there are multiple Pareto optimal solutions, solving a MOP is not as straightforward as it is for single-objective optimization problem (SOP). Therefore, there are several methods to solve a MOP. Many methods convert a MOP in multiple different SOP. This kind of approach is known as *scalarization*. Generally speaking, there are two different main categories of space exploration algorithms: exact and heuristic. Exact methods, such as linear optimization or branch&bound, search for a global optimum value, therefore it may be not suitable to be used in applications with a large solutions space search. On the other hand, heuristic methods aim at producing a representative set of Pareto optimal solutions, searching in a subset of the whole solution space. Evolutionary algorithms (EAs), such as Non-dominated Sorting Genetic Algorithm-II (NSGA-II), are popular approaches to generate Pareto optimal solutions to a MOP. Their use in MOP solving has been extensively researched and their efficiency demonstrated [39–41]. Authors of [42] reported a complete state of the art about (EA). The main advantage they offer when applied to solve MOPs is that they generate sets of *candidate* solutions, allowing computation of an approximation of the entire Pareto front. On the other hand, there is no upper bound to the computational time required to find such representative set of the Pareto front.

4.2 Approximate hardware implementation of DTs

In order to state the amount of error introduced by the approximation, all the combinations of precision-scaled model features have to be considered. Figure 5 sketches a detailed schematic of the proposed flow. Starting from dataset, we exploit KNIME [38] to obtain trained DT models. They are described in PMML, an XML-based predictive model interchange format. Since the KNIME tool makes use of the IEEE 754 double precision floating-point representation for features and thresholds, the size of the solution space is $52^{|F|}$, where 52 is the number of bits for the representation of the mantissa and $|F|$ is the size of the features set F. Therefore, an exhaustive exploration of the solution space may be unfeasible.

To overcome this issue, the evaluation of configurations is performed using an NSGA-II algorithm. An implementation of such algorithm is provided by the ParadisEO framework [43]. In order to suitably configure the MOP problem, we take into account two different fitness functions: (i) the amount of neglected bits and (ii) the accuracy of the model. Indeed, the fewer bit to compare, the better hardware accelerator overhead, even though the resulting

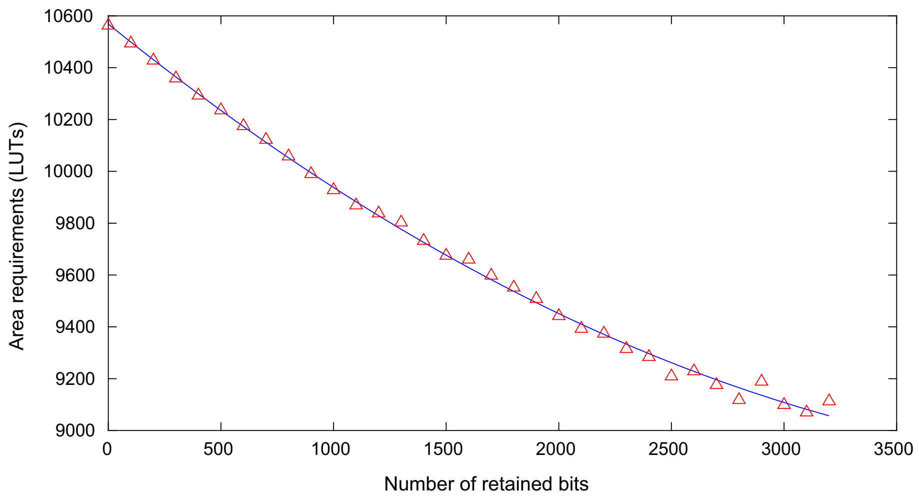


Fig. 6 Area requirements (LUTs) at increasing degree of approximation

accuracy of approximate model has to turn acceptable for the application. Each approximate configuration of the model is, in the NSGA-II terminology, a *chromosome* having as many *genes* as comparisons. The value of each gene states the amount of neglected bits at the matching comparator.

A better choice would take into account more significant hardware measurements, such as area, power consumption or maximum clock speed, rather than the number of neglected bits. Unfortunately, the estimation of such parameters is not suitable as it does not come from immediate evaluation, but requires to run synthesis tools that would make exploration not feasible in terms of computational time. Conversely, the number of neglected bits is directly expressed by chromosomes, and it is directly related to area, time and power consumption of target circuit; therefore, the considered fitness function is effective and pretty much like immediate to evaluate.

In order to evaluate choices concerning the reward fitness function, and, in particular, the correspondence between neglected bits and area savings, a classifier system has been trained on purpose. This classifier, consisting in 100 DBs, has been synthesized on a Xilinx Zynq-7020 FPGA, varying the number of neglected bits without taking accuracy into account. Figure 6 shows this preliminary result. As expected, the amount of area saved increases as the number of mantissa bits discarded grows. This is because DBs, which are bit string comparators, have fewer bits to compare; therefore, they need less combinational logic.

As for the accuracy, the fitness function is evaluated by performing the same approach exploited in KNIME, hence requires simulation of a test set onto the approximate model. In order to simulate reduced-mantissa floating point in software, we resort to the FLexible Arithmetic Library (FLAP), previously introduced in [44].

As result, approximate DT models are provided in PMML as well. In order to get VHDL code to synthesize the approximate hardware DT accelerator, a modified PMML2VHDL tool is involved into the flow [10].

Last step of the flow in Fig. 5 involves the synthesis of VHDL. As in this paper, we exploit FPGA devices as hardware configurable technology, and in particular a Xilinx Zynq 7020, we employ the Xilinx Vivado tool.

5 Experimental result and case study

In this section, we present experimental result of the proposed approach. In particular, we show two different experimental campaigns. The first takes into account 50 different classification problems in order to evaluate, at different workload, the robustness of our approximate computing methodology. Then, we give details about a case study based on the SPAM classification public dataset. Each experiment has been executed by means of the previously illustrated flow, reported in Fig. 5. It is worth noticing that as for NSGA-II algorithm involved into our proposed approach, there are some parameters that does not depend on the particular classification model, though they affect the result quality and computational time of experiments. Among the many, parameters that most affect the quality of the results and the execution time are the size of the initial population, the number of iterations and the mutation and crossover probabilities. Nevertheless, since there are only a few parameters, we have succeeded in a good configuration by successive attempts.

During the experimental phase, several campaigns were conducted, during which the configuration parameters of the GA were modified several times aiming at Pareto frontiers that were sufficiently diversified and populous. Though, as foreseeable, we realized that to obtain a populous Pareto frontier and avoid local sub-optimum it is necessary to increase the size of the initial population as much as possible. Then, in order to avoid long-run exploration around local sub-optimum, mutations have to take place frequently. In addition, we configured GA in order to discard solutions with a significantly high accuracy loss w.r.t. accurate classifier. Hence, we set our GA parameters as follows: initial population equals 2000 individuals, mutation and crossover probabilities set to 0.7 and 0.9, respectively, and accuracy loss threshold set to 4%.

5.1 Approximate MCSs

To prove the robustness of the proposed methodology, we exploited PMMLGen tool [8] to provide different workloads, in terms of models, to be approximated.

In particular, we collected 51 different datasets varying on the number of features (from 1 to 50) and number of classes (from 2 to 20). Then, we trained, for each dataset, a single DT classification model and random forest classification models with different number of DTs, namely 5, 10, 15 and 20. For each of the 255 trained classifiers, we found several approximate solutions by means of approximate exploration. Then, we synthesized the ones that belongs to the Pareto frontier bounds, i.e., the ones characterized by best reward value, meant maximum reduction of area overhead, and ones affected by minimum accuracy loss. We report the amount of resource occupation gain (in terms of FPGA LUTs and registers) and the accuracy loss evaluated in percentage w.r.t. the original synthesized model in Fig. 7 and 8, respectively, for maximum area overhead reduction and minimum accuracy loss synthesized solutions. Please, kindly note that although both in percentage, the scale for overhead gains is different w.r.t the one for accuracy loss.

For both graphs, we can state that accuracy loss decreases on the number of trees involved into the classification system. This observation confirms that random forest models are characterized by inherent resiliency property, and the greater the number of trees involved into models, the lower error introduced by approximation. Then, as for the reward, significant overhead reduction can be observed for random forest with 5, 10 and 15 trees, while the single DT model and 20 trees random forest exhibit lower area reduction values. Indeed, single DT models cannot be conveniently approximate w.r.t. random forest models for the

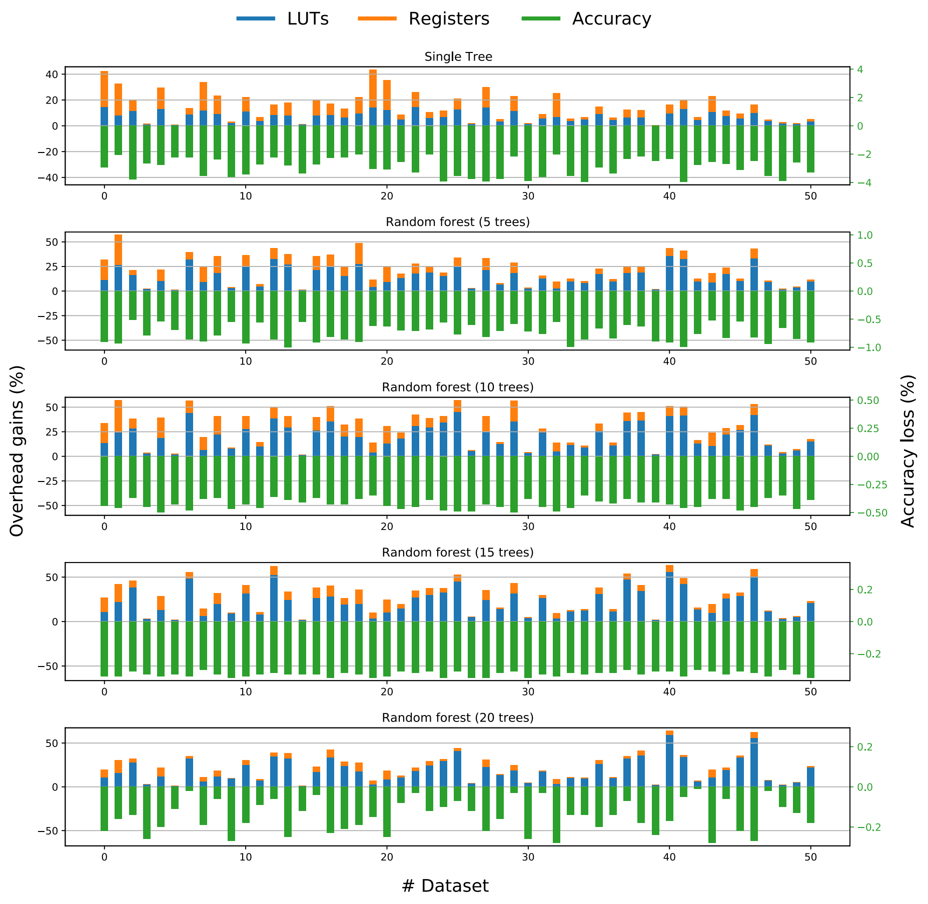


Fig. 7 Amount of resource gain and accuracy loss for 50 different classification problems for maximum area overhead reduction approximate solutions. Please, kindly note that the scale on the left differs from the one on the right

absence of a combiner that could mitigate the effect of introduced approximation. Nevertheless, contribution on area overhead of combiner circuits for random forest models with a significant number of trees makes the approximate computing technique less effective. As for solutions characterized by the minimum accuracy loss, we can see that even a small percentage of accuracy loss corresponds to a significant resource gain. As for synthesis of maximum accuracy loss solutions, we observe for some experiments area reduction of more than 50% against about 0.2% of accuracy loss.

5.2 Case study: SPAM detection

Since recognizing emails as SPAM on non-SPAM involves the classification of a large amount of information, a spam-detector case study is used to evaluate the approach introduced in this paper. The dataset used for this case study is Spambase [45], which contains 4601 emails, 1813 of which are SPAM. This dataset is freely available and makes use of 57 different

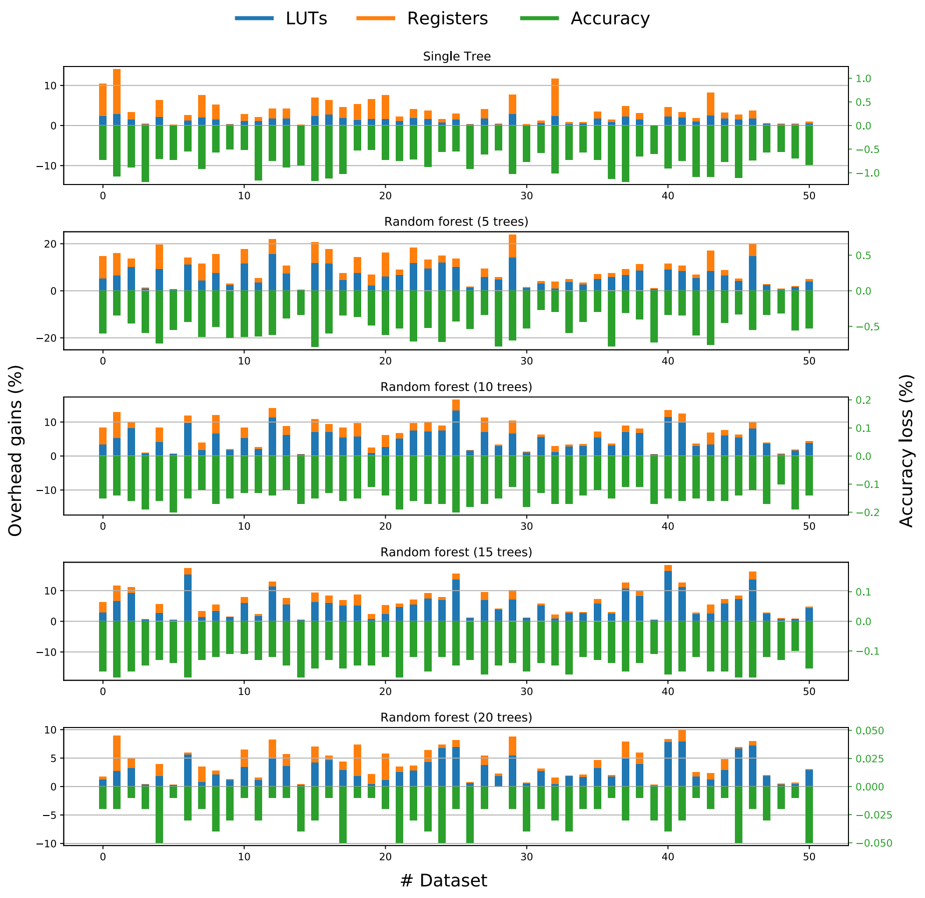


Fig. 8 Amount of resource gain and accuracy loss for 50 different classification problems for minimum accuracy loss approximate solutions. Please, kindly note that the scale on the left differs from the one on the right

features, expressed in the floating-point notation, to characterize elements that are part of the dataset. Each of the features specifies how often a word or a character appears in each element of the dataset, i.e., in an email.

During the training phase, conducted using the KNIME tool, 40 different random forest classifiers with a number of DTs ranging from 1 to 40 are trained.

The AxC exploration phase found, for each of the 40 classifiers, a certain number of approximate configurations on the Pareto frontier but for each of them only the configuration with minimum error and the one that requires less silicon area has been reported.

Figure 9 shows the area requirements in terms of LUTs, as the number of DTs used by the classifier increases. For all the measured quantities, an increasing trend, as the number of trees grows, is shown for area requirements. The growth, however, is clearly sub-linear. In addition, it can be seen that the difference between requirements of the exact classifier and the approximate one increases as the number of trees grows. This is because even if the complexity of single DTs—i.e., the number of nodes of which they consist of and the height of DTs themselves—decreases significantly as the number of trees used by the classifier

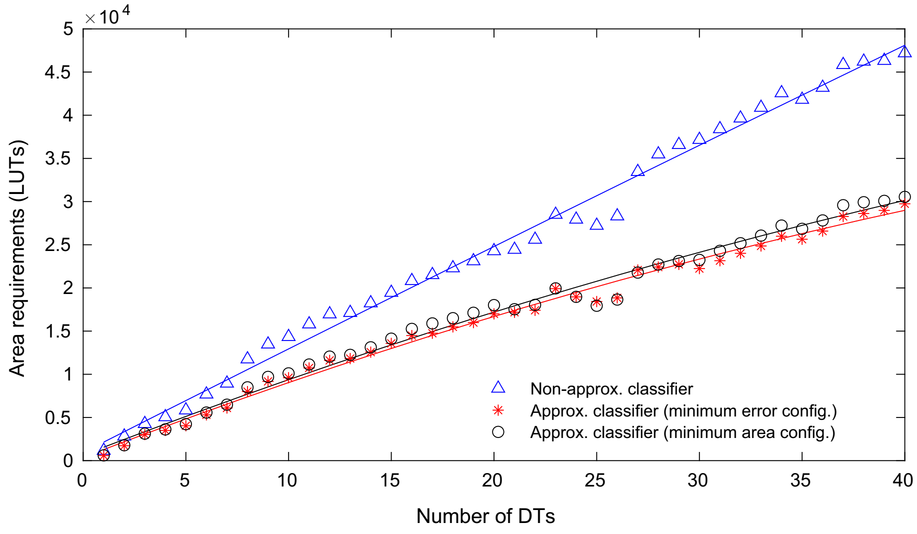


Fig. 9 Area requirements (LUTs)

increases, the total number of nodes increases, providing more approximation opportunities. This behavior can be observed also for the amount of FPGA slices and FPGA registers, and both when considering solutions providing minimum error and those requiring the minimum silicon area. Furthermore, it can be noted that the difference in terms of area requirements between the minimum error and the minimum area solutions always remains negligible.

Figure 10 compares the levels of classification accuracy, as the number of trees used by the classifier increases, provided by the precise version—without approximation—and by the approximate version that has minimum area requirements. It is evident, from the graph, that there is only a small difference in accuracy between the configurations. Moreover, it remains very small as the number of trees used for classification varies. On the other hand, the increase in the number of DTs used in the classification process makes smaller contribution as the number of DTs grows. This asymptotic behavior can be seen in exact and approximate classifiers, and it is due to the fact that by increasing the number of models, datasets involved for training turn out simpler and corresponding DTs get less branched, which leads to a saturation of the accuracy level provided by the classifier model.

5.3 Comparison with previous approaches

In [14], a similar approach to the one presented in this paper has been adopted, but instead of exploring the solutions space with heuristics, the use of an exact algorithm, namely Branch & Bound (B&B), was proposed. While, on the one hand, the use of an exact algorithm for the solution of a MOP allows to reach a global optimal solution, on the other hand its use becomes prohibitive with large solutions spaces. Despite numerous improvements that the authors have made to the algorithm (B&B), such as pre-pruning of the tree and grouping features by information gain, they have managed to evaluate only a few classifiers, and for each of them only a few approximate configurations. This has greatly limited the quality of the solutions obtained. Table 2 shows classification error and hardware requirements in terms of LUTs for both approaches. As it can be observed, when compared to those obtained using

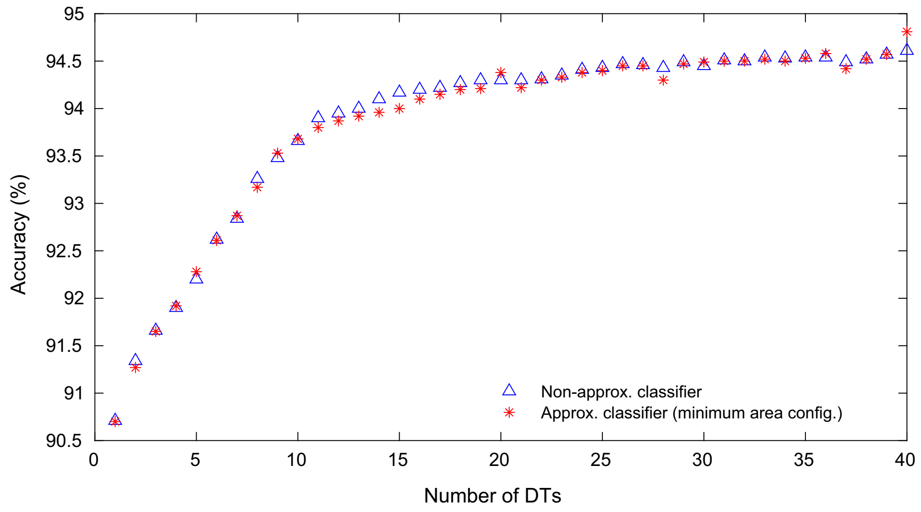


Fig. 10 Accuracy

Table 2 Comparison of results obtained from previous approaches

	Approximate Minimum Error				Approximate Minimum Area			
	Error		LUTs		Error		LUTs	
DTs	B&B	GA	B&B	GA	B&B	GA	B&B	GA
1	0	0	635	610	8.692E-4	1.0E-4	630	586
10	0	0	11853	10106	8.692E-4	3.0E-4	9646	9605
20	0	0	18091	18012	0	2.0E-4	18091	16996
30	0	0	23357	23243	4.346E-4	1.0E-4	23330	22243
40	0	0	33811	30544	8.692E-4	0.0E+0	30847	29747

GA, solutions provided by the B&B approach are worse. The difference in quality does not depend on the search algorithm itself, but on the amount of approximate configurations that have been taken into account during the space exploration phase.

6 Conclusion

This paper addresses the design of DTs-based MCSs. Leveraging the AxC design paradigm, the classification accuracy is traded off for a reduction in the silicon area requirements of hardware-implemented MCSs. By referring to automatic approaches for MCSs hardware implementation, proposed in the scientific literature, approximation has been added directly in MCS models, acting on the number of bits used to represent the value of each model feature.

To prove the validity of the proposed approach, a spam-detector case study is provided. Several classifiers, with a number of trees ranging between 1 and 40, have been trained. Then, the optimal number of bits to be used to represent each of the features of the model is searched by means of NSGA-II genetic algorithm. Among all Pareto-optimal hardware configurations, the one providing minimum classification error configuration and the one requiring the minimum amount of silicon area were taken into account for further consideration. Experimental results show a significant reduction in area requirements, for both the minimum error and minimum area configuration. Since the classification is very resistant to error, those configurations are very similar both in terms of area requirements and classification error.

Funding Open access funding provided by Università degli Studi di Napoli Federico II within the CRUI-CARE Agreement.

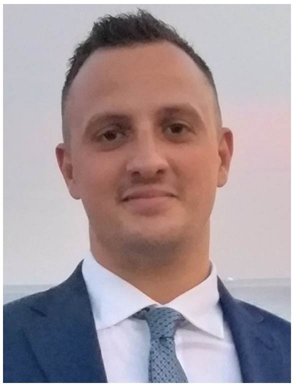
References

1. Gargiulo F, Mazzariello C, Sansone C (2013) Multiple classifier systems: theory, applications and tools. In: Handbook on neural information Processing, Springer, pp 335–378
2. Mohammed AM, Onieva E, Woźniak M (2019) Vertical and horizontal data partitioning for classifier ensemble learning. In: International Conference on Computer Recognition Systems, Springer, pp 86–97
3. Bellmann P, Thiam P, Schwenker F (2018) Multi-classifier-systems: architectures, algorithms and applications. In: Computational Intelligence for Pattern Recognition, Springer, pp 83–113
4. Van Essen B, Macaraeg C, Gokhale M, Prenger R (2012) Accelerating a random forest classifier: Multi-core, GP-GPU, or FPGA? Proceedings of the 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines, FCCM 2012 pp 232–239, https://doi.org/10.1109/FCCM.2012.47
5. Amato F, Barbareschi M, Casola V, Mazzeo A (2014) An fpga-based smart classifier for decision support systems. Intel Distrib Comput VII. Springer, Cham, pp 289–299
6. Amato F, Barbareschi M, Casola V, Mazzeo A, Romano S (2013) Towards automatic generation of hardware classifiers. International Conference on Algorithms and Architectures for Parallel Processing. Springer, Cham, pp 125–132
7. Barbareschi M, Mazzeo A, Miranda S (2016b) Adopting decision tree based policy enforcement mechanism to protect reconfigurable devices. Intelligent Interactive Multimedia Systems and Services 2016. Springer, Cham, pp 73–81
8. Barbareschi M (2016) Implementing hardware decision tree prediction: a scalable approach. In: 2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA), IEEE, pp 87–92
9. Tong D, Qu YR, Prasanna VK (2017) Accelerating decision tree based traffic classification on fpga and multicore platforms. IEEE Trans Parallel Distrib Syst 28(11):3046–3059
10. Barbareschi M, Del Prete S, Gargiulo F, Mazzeo A, Sansone C (2015) Decision tree-based multiple classifier systems: An fpga perspective. International Workshop on Multiple Classifier Systems. Springer, Cham, pp 194–205
11. Nguyen HL, Woon YK, Ng WK (2015) A survey on data stream clustering and classification. Knowl inf syst 45(3):535–569
12. O'donovan P, Gallagher C, Bruton K, O'Sullivan DT, (2018) A fog computing industrial cyber-physical system for embedded low-latency machine learning industry 4.0 applications. Manufacturing Letters 15:139–142
13. (2014) The data mining group. http://dmg.org/
14. Barbareschi M, Papa C, Sansone C (2017) Approximate Decision Tree-Based Multiple Classifier Systems. Springer Proceedings in Mathematics and Statistics 217:39–47
15. Xu Q, Mytkowicz T, Kim NS (2015) Approximate computing: A survey. IEEE Design & Test 33(1):8–22

16. Mittal S (2016) A survey of techniques for approximate computing. ACM Comput Surv 48(4):62:1–62:33. https://doi.org/10.1145/2893356
17. Sidiroglou-Douskos S, Misailovic S, Hoffmann H, Rinard M (2011) Managing performance vs. accuracy trade-offs with loop perforation. In: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, ACM, pp 124–134
18. Keramidas G, Kokkala C, Stamoulis I (2015) Clumsy value cache: An approximate memoization technique for mobile gpu fragment shaders. In: Workshop on Approximate Computing (WAPCO'15)
19. Tong JYF, Nagle D, Rutenbar R (2000) Reducing power by optimizing the necessary precision/range of floating-point arithmetic. Very Large Scale Integration (VLSI) Systems. IEEE Trans 8(3):273–286
20. Fang F, Chen T, Rutenbar R, et al. (2002) Floating-point bit-width optimization for low-power signal processing applications. In: Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on, IEEE, vol 3, pp III–3208
21. Yeh T, Faloutsos P, Ercegovac M, Patel S, Reinman G (2007) The art of deception: Adaptive precision reduction for area efficient physics acceleration. In: 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007), IEEE, pp 394–406
22. Osborne WG, Coutinho J, Luk W, Mencer O (2008) Power-aware and branch-aware word-length optimization. In: Field-Programmable Custom Computing Machines, 2008. FCCM'08. 16th International Symposium on, IEEE, pp 129–138
23. Rubio-González C, Nguyen C, Nguyen HD, Demmel J, Kahan W, Sen K, Bailey DH, Iancu C, Hough D (2013) Precimonious: Tuning assistant for floating-point precision. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ACM, p 27
24. Chippa VK, Chakradhar ST, Roy K, Raghunathan A (2013) Analysis and characterization of inherent application resilience for approximate computing. In: Proceedings of the 50th Annual Design Automation Conference, ACM, p 113
25. Traiola M, Savino A, Barbareschi M, Di Carlo S, Bosio A (2018) Predicting the Impact of Functional Approximation: From Component- to Application-Level. 2018 IEEE 24th International Symposium on On-Line Testing and Robust System Design, IOLTS 2018 (iii):61–64, https://doi.org/10.1109/IOLTS.2018.8474072
26. Traiola M, Savino A, Di S (2019) Microelectronics Reliability Probabilistic estimation of the application-level impact of precision scaling in approximate computing applications. Microelectron Reliab 102:113309. https://doi.org/10.1016/j.microrel.2019.06.002
27. Breiman L (2001) Machine learning. Random forests 45(1):5–32
28. Rokach L, Maimon O (2005) Top-down induction of decision trees classifiers-a survey. IEEE Trans Syst, Man, and Cybernetics, Part C (Applications and Reviews) 35(4):476–487
29. Quinlan JR (2014) C4. 5:programs for machine learning. Elsevier, Netherlands
30. Venkataramani S, Raghunathan A, Liu J, Shoaib M (2015) Scalable-effort classifiers for energy-efficient machine learning. In: Proceedings - Design Automation Conference, Institute of Electrical and Electronics Engineers Inc., vol 2015-June, https://doi.org/10.1145/2744769.2744904
31. Nepal K, Li Y, Bahar RI, Reda S (2014) ABACUS: A technique for automated behavioral synthesis of approximate computing circuits. Proceedings -Design, Automation and Test in Europe, DATE pp 1–6, https://doi.org/10.7873/DATE2014.374
32. Aiserman M, Braverman EM, Rozonoer L (1964) Theoretical foundations of the potential function method in pattern recognition. Avtomat i Telemeh 25:917–936
33. Mohri M, Rostamizadeh A (2013) Perceptron mistake bounds. arXiv preprint arXiv:1305.0208
34. Van Leussen M, Huisken J, Wang L, Jiao H, De Gyvez JP (2017) Reconfigurable Support Vector Machine Classifier with Approximate Computing. Proceedings of IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2017-July:13–18, https://doi.org/10.1109/ISVLSI.2017.13
35. Zhou Y, Lin J, Wang Z (2018) Energy efficient SVM classifier using approximate computing. Proceedings of International Conference on ASIC 2017-Octob:1045–1048, https://doi.org/10.1109/ASICON.2017.8252658
36. Cortes C, Vapnik V (1995) Machine learning. Support-vector networks 20(3):273–297
37. Kulkarni P, Gupta P, Ercegovac M (2011) Trading accuracy for power with an underdesigned multiplier architecture. In: 2011 24th Internatioal Conference on VLSI Design, IEEE, pp 346–351
38. (2007) Knime. https://www.knime.com/
39. Palesi M, Givargis T (2002) Multi-objective design space exploration using genetic algorithms. In: Proceedings of the tenth international symposium on Hardware/software codesign, ACM, pp 67–72
40. Konak A, Coit DW, Smith AE (2006) Multi-objective optimization using genetic algorithms: A tutorial. Reliab Eng Syst Safe 91(9):992–1007

41. Erbas C, Cerav-Erbas S, Pimentel AD (2006) Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design. IEEE Trans Evolut Comput 10(3):358–374
42. Van Veldhuizen DA, Lamont GB (2000) Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. Evolut comput 8(2):125–147
43. (2014) Paradiseo. http://paradiseo.gforge.inria.fr
44. Barbareschi M, Iannucci F, Mazzeo A (2016a) An extendible design exploration tool for supporting approximate computing techniques. In: 2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS), IEEE, pp 1–6
45. (1998) Spambase. https://archive.ics.uci.edu/ml/datasets/Spambase

Mario Barbareschi Mario Barbareschi received the PhD in Computer and Automation Engineering in 2015 and the masters degree in Computer Engineering cum laude in 2012, both from the University of Naples Federico II. His research interests include hardware security and trust, cyber-physical security, approximate computing and embedded systems design based on the FPGA technology. He has authored more than 30 peer-reviewed papers published in leading journals and international conferences.



Salvatore Barone Salvatore Barone received the masters degree in Computer Engineering cum laude in 2018 from the University of Naples Federico II, Italy, where he is currently a PhD student. His research interests include safety-critical systems, railway systems, approximate computing and embedded systems based on the FPGA technology.

Nicola Mazzocca Nicola Mazzocca is full professor of Computer Systems at the Department of Electrical Engineering and Information Technology of the University of Naples “Federico II”. Since 1994, he has held numerous university courses, in mater and in professional training activities on topics related to electronic computers, high-performance systems, distributed systems, embedded systems, security and reliability of computer systems. The research activities of Prof. Mazzocca concern computer architecture, distributed systems, high-performance systems, models for the evaluation of the specifications of processing systems in critical applications. He is author of more than 250 papers on international journals, books and conference proceedings.