Decision Tree-Based Multiple Classifier Systems: an FPGA perspective.

Mario Barbareschi, Salvatore Del Prete, Francesco Gargiulo, Antonino Mazzeo, and Carlo Sansone

Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione (DIETI)
University of Naples Federico II
Via Claudio 21, 80125, Naples, Italy
{mario.barbareschi, salvatore.delprete, francesco.grg, antonino.mazzeo, carlo.sansone}@unina.it

Abstract. Combining a hardware approach with a multiple classifier method can deeply improve system performance, since the multiple classifier system can successfully enhance the classification accuracy with respect to a single classifier, and a hardware implementation would lead to systems able to classify samples with high throughput and with a short latency. To the best of our knowledge, no paper in the literature takes into account the multiple classifier scheme as additional design parameter, mainly because of lack of efficient hardware combiner architecture. In order to fill this gap, in this paper we will first propose a novel approach for an efficient hardware implementation of the majority voting combining rule. Then, we will illustrate a design methodology to suitably embed in a digital device a multiple classifier system having Decision Trees as base classifiers and a majority voting rule as combiner. Bagging, Boosting and Random Forests will be taken into account. We will prove the effectiveness of the proposed approach on two real case studies related to Big Data issues.

Keywords: Multiple Classifier Systems, Decision Tree, Bagging, Boosting, Random Forest, Field Programmable Gate Array

1 Introduction

Modern applications based on data analysis have been bringing new architectural design challenges. They define in the literature a new class of problems, addressed as *Big Data*, whose characteristics are grouped in the 5 'V's (Volume, Velocity, Variety, Veracity and Value), in order to indicate the inadequacy of the current computer technologies and design techniques when, at some point in time, these 'V's are increased to an unprecedented level. In particular, in the case of data classification, machine learning and pattern recognition algorithms have to deal with large data sets and with a very high number of samples per time unit (throughput) that have to be classified. Typical examples of problems with these characteristics are in the fields of intrusion detection [12], spam detection [10] and network traffic classification [4].

The research community has made a big effort in devising not only new learning algorithms to enhance classification accuracy but also new design techniques, whose aim is to improve the classification speed, mainly exploiting hardware implementations. In the latter context, reconfigurable hardware technology, such as the Field Programmable Gate Array (FPGA), is able to realize high parallel, high speed and large digital designs, and, as it provides a programming flow that is somewhat similar to the software deploying, it allows easy and feasible hardware updates. These technological features are a promising solution for data classification tasks when a very high throughput value is required.

Among the multitude of different classification approaches proposed so far, Decision Trees (DTs) are one of the most suited for a hardware implementation, since they do not require arithmetic calculations, which are expensive to be realized, but only comparisons. The authors of [13] illustrated a high throughput DT classifier hardware accelerator design, mainly based on the pipeline technique, which reaches up to 114 times speed-up, compared with a software approach. With the aim of comparing power consumption of DT hardware and software approaches, authors of [7] introduced a methodology flow and, as result, they showed that the hardware version need only 0.03% of the energy used by the software. In [1] a hardware accelerator for the DT detailed implementation is given, accomplished by exploiting a speculative approach on the node evaluation, reaching a very high throughput value, while in [2] same authors introduced a methodological flow to automatically obtain such hardware.

On the other hand, in many pattern recognition applications achieving an acceptable accuracy is conditioned by the large pattern variability, whose distribution cannot be simply modelled. This affects the results of the classification system so that, once this has been designed, its performance cannot be improved beyond a certain bound, despite efforts at refining either the classification or the description method [9]. A possible solution is the use of a multiple classifier system: the consensus of a set of classifiers may compensate for the weakness of a single classifier.

Combining hardware accelerators with multiple classifier techniques can dramatically improve the system performance, as the multiple classifier systems are able to successfully enhance the classification accuracy and designs realized in hardware perform classification of samples with really high throughput and short latency. To the best of our knowledge, no paper in the literature takes into account the multiple classifier scheme as an additional design parameter, mainly because of lack of efficient hardware combiner architectures.

For those reasons and starting from the previous considerations, in this paper we try to fill the gap by presenting an efficient hardware implementation of the majority voting rule. Moreover, we illustrate a design methodology to suitably embed in a digital device a multiple classifier system, having a DT as base classifier and a majority voting rule as combiner. In particular, Bagging, Boosting and Random Forests [11] have been considered as multiple classifier systems. By taking into account the constraints given by the specific problem, the proposed methodology is able to select the best possible hardware multiple classifier

system by considering classification accuracy, throughput and hardware latency. In order to avoid the generation of unfeasible ensembles (i.e., they cannot be synthesized in hardware), we also present an early prediction approach to do a preliminary estimate of the required hardware resources, empirically exploiting measurements and adopting as base classifier the hardware version of the DT introduced in [1] and as hardware a Xilinx Virtex 5 FPGA device.

The paper is structured as follows: in Section 2 we briefly introduce the hardware DT classifier and the combiner that we exploit to implement a hardware multiple classifier system. Section 3 contains a detailed description of the proposed design methodology. Thus, in Section 4, we demonstrate the effectiveness of the proposed approach through two real case studies, namely spam detection and IP traffic classification, discussing the main performance aspects. At the end, the Section 5 concludes the paper.

2 From Classification Model to Hardware Accelerator

In order to gather useful data about the performance of hardware classifiers and analyse them by varying the classification parameters, it is mandatory to have what is closest to a physical realization of the hardware components under test, such as the description at the *Place and Route* (PAR) level. Essentially, a PAR description has a fine grain level of physical details as it describes a digital circuit in terms of what will be realized on the technological target. For instance, a PAR description for a Xilinx FPGA contains configuration and allocation of the Look-up Tables (LUTs), Registers, Slices and routing resources involved into the design.

Towards this aim, experiments need for designs described as HDL projects, which implement the hardware accelerator for a specific classification module.

2.1 Decision Tree implemented on FPGA

Basically, the hardware implementation for a multiple classifier scheme is directly inherited from the model structure. As the multiple classifier model combines classification techniques, which are not dependent one another during the evaluation phase, and makes their predictions with a combining rule, the hardware structure is designed with parallel classification entities that execute high speed classification in parallel and with a hardware combiner which quickly organizes all the classifiers' outputs. In this paper we consider the DT as base classifier model, since it can be successfully implemented in hardware and it is suitable in a wide range of applications and domains. In particular, for the proposed multiple classifier architecture, we exploit the hardware accelerator presented in [1] because it is specifically designed for a FPGA technology, as it deeply exploits the FPGAs' parallel structure. Moreover, the authors have deployed a tool whereby hardware models can be automatically generated from formal models described in Predictor Mark-up Model Language (PMML), which represents a

standard exploitable as output artefact by a wide range of analytic frameworks (e.g. KNIME[1] or WEKA[2])[2].

The DT model mapping is accomplished by implementing each tree node as a binary comparator, which, once received the feature value, returns a boolean value. In order to exploit intrinsic hardware parallelism, all the tree nodes work in parallel and their decision values feed the boolean network that computes which tree leaf has been reached. In particular, the boolean network needs as much input as the DT nodes and gives outputs equal to the number of classes, such that only one output is high each time. This approach turns out to be speculative, since the DT model does not consider all the decisions at the same time, but only the ones that belong to a single path, since the visiting algorithm traverses the decision nodes one by one accordance with the comparison results. The speculation implies very fast computation since there are no dependencies between the decisions that can be simultaneously evaluated.

In order to support a multiple DT hardware accelerator, in the next subsection we present the majority voting rule as a hardware combiner. Furthermore, for extending the automatic hardware description generation illustrated in [2], we have successfully integrated the automatic generation of such a combiner in the previously developed tool *PMML2VHDL*.

2.2 Hardware Combiner

In hardware, the combiner of a multiple classifier system is one of the most influential elements both for latency and area occupation.

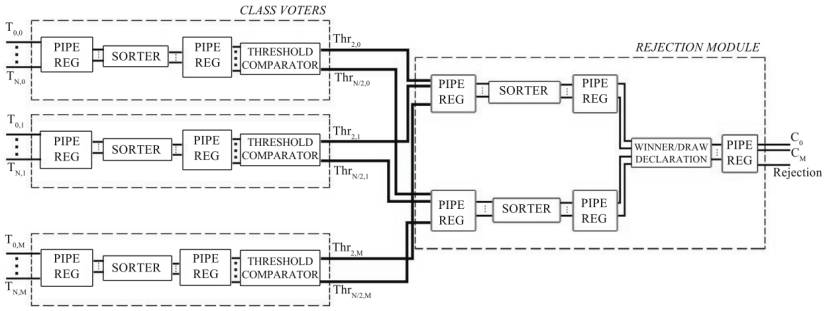

Fig. 1. Majority voter implemented as a pipelined odd-even sorter, which also embeds a rejection module.

According to its implementation, the combiner could be a bottleneck, so it is necessary to find a balanced design which can be a good trade-off between resource occupation and maximum throughput. Among possible design solutions,

[1] http://www.knime.org
[2] http://www.cs.waikato.ac.nz/ml/weka/

in this paper we adopt the one based on the *winning threshold* and on the *sorting network*. First of all, the idea is to exploit a combiner that does not care about which class is voted by which tree, but only how many votes a class gets. Each DT expresses its own decision as a decoded output, hence only one bit is high, and each class gets a certain amount of votes, which corresponds to the number of high bits received by each DT. Rather than use a binary adder to count high bits, it is simpler to collect the votes in a vector by shifting all the high bits at the beginning of it and verifying if they are enough to declare someone as the winner.

As depicted in Figure 1, we design the combiner as an odd-even sorter for each class, which is a component that implements a simple sorting algorithm closely related to bubble-sort, but in a parallel version. Indeed, the architecture compares all the couples of adjacent elements, whose first member occupies an odd position in the vector, swapping them if they are in the wrong order. Then, it repeats the operation for even-indexed couples. The whole process is iterated until the vector is totally ordered. In hardware, the algorithm is implemented as a pipelined sorting network, such that the groups of 2-selectors work in parallel reaching very high speed. The ordered vector is the input of another component, which verifies whether a threshold of votes is reached according to the product of the first X values in the vector, where X is the threshold value. The outputs of all these components are compared in the rejection module, i.e. another sorting network, in order to declare a winner or a draw situation.

2.3 Automatic generator of Classification Models

Since our goal is to control the classification models' characteristics, such as number of nodes, maximum tree depth, number of classes, number of trees and so on, in order to evaluate their influence on hardware classifier performance, we develop *PMMLGen*, namely a Java tool which builds PMML models with desired parameters. This application allows us to avoid learning for actual training sets to obtain classification models (i.e., DTs) with the desired characteristics, as it can automatically build working PMML models being able to control some tree-based classifiers parameters, while the others are randomly and coherently picked.

Exploiting *PMMLGen*, we have collected some models whose characteristics are needed for defining trends of the hardware-related parameters according to their features, and hence for estimate an *early prediction* function, as detailed in the next Section. This is useful to reduce test cases, since it avoids generation of unfeasible ensembles, i.e. models that cannot be synthesized in a specific targeted hardware.

3 Methodology for Performance Evaluation: Early Prediction Function

As the space of possible classification system solutions is very large, in this Section we introduce an area-occupation Early Prediction (EP) function that is

useful to early discard models which likely lead to designs that are not feasible to be implemented in the target device, which in this paper is an FPGA. To this aim, we have obtained such a function through a step-wise regression applied on the results of a test-suite made up of artificial ensembles generated by *PMMLGen*. We have focused on the effect of each parameter, varying them one per time and keeping the others fixed at a specific value. In particular, we have considered: number of nodes for each tree and, consequently, the overall number of nodes in the ensemble ($\#Nodes$); number of trees ($\#Trees$); number of classes ($\#Classes$) and number of features ($\#Features$). Therefore, we can give a general form for the EP:

$$EP = f(\#Classes, \#Features, \#Trees, \#Nodes). \tag{1}$$

The actual expression of EP strictly depends on the device considered, since it refers to its technological characteristics.

Even if the EP should be used to predict, from the parameters of the obtained model, whether the multiple classifier system requires an amount of resources suitable for the targeted device, it might be useful even without having trained models. The first three parameters are clearly defined by the problem at hand, but the last one is tightly coupled with the dataset and with the learning algorithm. To preliminarily have a suitable estimation about the number of nodes which will likely characterize the trained models, it could be enough to get only one complete training for a significant ensemble model case, e.g. on a small ensemble with 5 trees. In this way, the $\#Nodes$ parameter of the EP function can be estimated as the averaged number of trees nodes, since the relation between the number of nodes and number of trees is quite linear in most cases. Therefore we claim that this preliminary estimation of the nodes is an overestimation since, with the growing of the involved trees, the number of nodes might linearly grow or be constant.

Performance Function - once the number of possible design solutions have been reduced, selected classifier systems have to be implemented in order to get information about their accuracy, latency and throughput values. As stated before, this step involves hardware synthesis tools which translate designs from a PAR description.

As for the accuracy, we adopt a k-fold cross validation technique for each classification model under test. Having the performance values for each feasible design, it is possible to assign a *performance value* to each involved model and, in the end, select the best one according to the requirements of the application. Towards this aim, we define a suitable performance function (P) assuming a linear dependence on *accuracy*, *latency* and *throughput*, hence its expression is given by:

$$P = \alpha \cdot Accuracy^{Norm} + \beta \cdot \frac{1}{Latency}^{Norm} + \gamma \cdot Throughput^{Norm}. \tag{2}$$

As one can notice, since *accuracy*, *latency* and *throughput* have different ranges and measurement units, there is the need for the value normalization. An effective solution could be the evaluation of the performance improvement when a multiple classifier system is used, with respect to the use of its base classifier, i.e. the single DT. Hence, in Eq. 2, the notation $< v >^{Norm}$ stands for adopting the following normalization rule:

$$< v >^{Norm} \Rightarrow \frac{< v > - < v >_{DT}}{< v >_{DT}} \tag{3}$$

where $< v >_{DT}$ represents the value from a single DT built on all the data.

In conclusion, we can also observe that, in most cases, *latency* and *throughput* are not both critical, so that one of the weights of Eq. 2 can be considered null and the remaining two can be fixed to α and (1-α), respectively. In this specific case, given α, the design which maximizes P can be implemented in hardware.

4 Experimental Results

The goal of this section is twofold: first of all we test the proposed methodology, showing how the *early prediction* function can estimate the maximum number of trees according to the chosen ensemble strategy; then, exploiting the *performance function* P, we show the best classification system to be implemented in hardware.

For the latter aim, we prove the effectiveness of the previously introduced methodology with two case studies: the first one refers to spam detection, characterized by a huge amount of data that must be typically processed in a fixed time. The second one is related to the classification of IP traffic traces: in this case the main constraint to be satisfied is real-time classification [5, 6]. Moreover, while in the former case we have a binary problem with dozens of features, in the second we consider a multi-class problem with few features.

For all the tests, we considered as target reference the hardware platform Xilinx Virtex 5 XC5VLX110T, whose characteristics are illustrated in Table 1.

Array (Row x Col)	Slices	Slice Registers	Slice LUTs	Total I/O Banks	Max User I/OBs
160 x 54	17280	69120	69120	20	640

Table 1. Xilinx Virtex 5 XC5VLX110T characteristics

4.1 Early Prediction

For defining the EP function for the hardware device we are considering, we generated several artificial ensembles by using *PMMLGen* with model parameters whose values were in the ranges detailed in Table 2.

Parameter	Lower Bound	Upper Bound
$\#Trees$	5	50
$\#Classes$	4	32
$\#Features$	4	32
$\#Nodes$	55	3375

Table 2. Value ranges of the parameters.

We trained a single DT classifier, as well as the Bagging, AdaBoost and Random Forest multiple classifier systems, by using the KNIME Analytic framework. Once obtained the artificially generated ensembles, were saved as PMML files and later translated in VHDL by using the *PMML2VHDL* framework in order to synthesize them in hardware and retrieve their performance characteristics. For the Xilinx Virtex-5 we found the following expression of the EP function: $EP = 13 * \#Classes + 33 * \#Features - 74 * \#Trees + 9 * \#Nodes^*$.
where $\#Nodes^*$ is an estimation of the actual number of nodes in the ensemble, made as described in the previous Section.

In Figure 2 we report the obtained EP function. The estimation quality is globally good as the real required hardware resources are really close to the predicted ones; hence, EP can be used to evaluate the maximum feasible number of trees for each ensemble strategy.

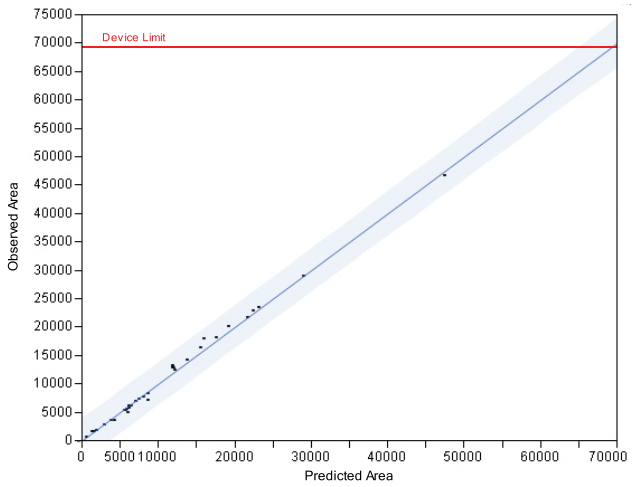

Fig. 2. Predicted vs Observed plot of Early Prediction function; the red line represents the hardware limit for the Xilinx Virtex 5 XC5VLX110T board.

In the following subsections we use the early prediction function and, for each classification system, we consider the accuracy evaluated by means of a 10-fold cross validation strategy.

4.2 Spam Detection

For this case study we used the *Spambase* dataset, publicly available on the UCI repository [3]. This dataset contains 4601 instances (1813 Spam cases) characterized by 57 continuous features. Note that, even if the number of training samples is not so significant, it is likely that a spam detection system should process a huge amount of data when operating in the field.

In order to select the best solution for this particular problem, we used the *performance function* P introduced in the previous Section. Since, in general, spam detection does not need to be performed in real-time, we are not particularly interested in minimizing *latency*, while we need to maximize *throughput*, since it is important to classify as many email as possible in a given time unit. So we can set $\beta = 0$ and $\gamma = 1 - \alpha$, thus re-writing the *performance function* as: $P = \alpha \cdot Accuracy^{Norm} + (1 - \alpha) \cdot Throughput^{Norm}$.

We considered four possible values for α, i.e., $\alpha = \{0.25; 0.50; 0.75; 1.00\}$ in order to differently weight *accuracy* and *throughput*. Note that the last value corresponds to the case in which only accuracy is considered, i.e. we are searching for the system with the best accuracy that can be implemented in hardware.

Using the *EP* function to estimate the required resources in terms of area on the FPGA board, we obtained the maximum feasible number of trees for each ensemble strategy, i.e. Bagging (158), Random Forest (24) and Boosting (30).

The results reported in Table 3 show that if we want to take care of the *throughput* ($\alpha = 0.25$) or prefer to equally weight *accuracy* and *throughput* ($\alpha = 0.5$), we have to choose the Bagging algorithm with 25 trees. On the other hand, if we believe that accuracy is more important ($\alpha \geq 0.75$), Random Forests with 24 base classifiers should be chosen. In all cases, we have several multiple classifier systems whose overall performance (according to our definition) is better than the one obtained by the single DT.

4.3 Traffic Classification

In this case we used a dataset provided by the Lawrence Berkeley National Laboratory (LBNL) [4], already used in [3, 8]. The dataset is composed by 134246 samples, characterized by 7 continuous features and 6 different classes: *POP3, FTP, SMTP, HTTP, BIT-TORRENT, MSN*. Again, we followed the above presented methodology in order to select the best possible hardware solution for this problem. Since, unlike spam detection, traffic classification would need to be performed in real-time, we surely need to minimize *latency*, while we would not be particularly interested in maximizing *throughput*. So we can simplify the expression of the *performance function*, by setting $\gamma = 0$ and $\beta = 1 - \alpha$ as in the following: $P = \alpha \cdot Accuracy^{Norm} + (1 - \alpha) \cdot \frac{1}{Latency}^{Norm}$.

By using the EP function we obtained the maximum feasible number of trees for each ensemble algorithm, i.e. Bagging (95), Random Forest (37) and Boosting (77).

[3] https://archive.ics.uci.edu/ml/datasets/Spambase
[4] http://ee.lbl.gov/anonymized-traces.html

(a) Decision Tree

#VHDL Nodes	Max Depth	Throughput [MS/s]	Accuracy [%]	P $\alpha = 0.25$	P $\alpha = 0.50$	P $\alpha = 0.75$	P $\alpha = 1.00$
206	41	112.8796	91.806	0.000	0.000	0.000	0.000

(b) Bagging

#Trees	#VHDL Nodes	Max Depth	Throughput [MS/s]	Accuracy [%]	P $\alpha = 0.25$	P $\alpha = 0.50$	P $\alpha = 0.75$	P $\alpha = 1.00$
5	276	28	123.09	92.13	0.069	0.047	0.025	0.004
10	296	21	129.43	91.85	0.110	0.074	0.037	0.001
15	281	18	130.40	91.26	0.115	0.075	0.034	-0.006
20	282	19	126.29	91.66	0.089	0.059	0.029	-0.002
25	266	14	133.56	90.85	**0.135**	**0.086**	0.038	-0.010
50	312	11	125.75	91.38	0.084	0.055	0.025	-0.005
100	348	9	127.15	90.89	0.092	0.058	0.024	-0.010
125	364	7	131.13	90.29	0.117	0.073	0.028	-0.017

(c) Random Forest

#Trees	#VHDL Nodes	Max Depth	Throughput [MS/s]	Accuracy [%]	P $\alpha = 0.25$	P $\alpha = 0.50$	P $\alpha = 0.75$	P $\alpha = 1.00$
5	1536	45	125.33	94.00	0.089	0.067	0.046	0.024
10	3027	45	124.27	94.88	0.084	0.067	0.050	0.033
15	4496	45	113.86	95.11	0.015	0.022	0.029	0.036
20	5939	45	120.85	95.33	0.063	0.054	0.046	0.038
24	6956	43	126.28	95.81	0.100	0.081	**0.062**	**0.044**

(d) Boosting

#Trees	#VHDL Nodes	Max Depth	Throughput [MS/s]	Accuracy [%]	P $\alpha = 0.25$	P $\alpha = 0.50$	P $\alpha = 0.75$	P $\alpha = 1.00$
5	1210	59	127.83	94.02	0.105	0.078	0.051	0.024
10	2657	60	100.27	95.30	-0.074	-0.037	0.001	0.038
15	3813	60	107.82	92.96	-0.031	-0.016	-0.002	0.013
20	5109	61	119.59	94.48	0.052	0.044	0.037	0.029
30	6764	62	112.89	94.05	0.006	0.012	0.018	0.024

Table 3. SPAM e-mail detection - VHDL Classifiers. Best solutions in terms of P are reported in bold.

Then, according to these results, we are able to select the best solutions for the problem at hand. As in the previous case, we considered four possible values for α, i.e., $\alpha = \{0.25; 0.50; 0.75; 1.00\}$. In this case it is interesting to note that, since the DT has a very small *latency* time, the single classifier solution should be chosen according to our *performance function*, when we want to use the *latency* parameter. On the other hand, if we are only interested in maximizing *accuracy*, a bagging ensemble with 5 trees should be implemented in hardware.

5 Conclusion

In this paper we presented a novel approach for an efficient hardware implementation of the majority voting combining rule and illustrated a design methodology to suitably embed in a digital device a multiple classifier system, having a DT as base classifier and a majority voting rule as combiner. Bagging, boosting and random forests have been considered as multiple classifier systems. Taking into account the constraints given by the specific problem, in the proposed methodology we introduced a performance function P that was able to select the best possible hardware classifier system by considering classification accuracy,

(a) Decision Tree

#VHDL Nodes	Max Depth	Latency [ns]	Accuracy [%]	P $\alpha = 0.25$	P $\alpha = 0.50$	P $\alpha = 0.75$	P $\alpha = 1.00$
111	11	49.98	91.78	0.000	0.000	0.000	0.000

(b) Bagging

#Trees	Max Depth	Latency [ns]	Accuracy [%]	P $\alpha = 0.25$	P $\alpha = 0.50$	P $\alpha = 0.75$	P $\alpha = 1.00$
5	11	91.66	92.96	-0.338	-0.221	-0.104	**0.013**
10	10	79.31	92.54	-0.275	-0.181	-0.086	0.008
30	10	84.77	91.79	-0.308	-0.205	-0.103	0.00
60	10	81.98	91.15	-0.294	-0.231	-0.119	-0.007
90	10	86.43	90.88	-0.319	-0.216	-0.113	-0.010

(c) Random Forest

#Trees	Max Depth	Latency [ns]	Accuracy [%]	P $\alpha = 0.25$	P $\alpha = 0.50$	P $\alpha = 0.75$	P $\alpha = 1.00$
5	13	85.00	91.85	-0.309	-0.206	-0.102	0.001
10	13	95.76	92.05	-0.358	-0.238	-0.117	0.003
20	13	96.22	92.19	-0.359	-0.238	-0.117	0.004
30	13	92.32	92.20	-0.343	-0.227	-0.111	0.005
35	14	96.99	92.21	-0.362	-0.240	-0.118	0.005

(d) Boosting

#Trees	Max Depth	Latency [ns]	Accuracy [%]	P $\alpha = 0.25$	P $\alpha = 0.50$	P $\alpha = 0.75$	P $\alpha = 1.00$
5	14	87.92	92.46	-0.322	-0.212	-0.102	0.007
10	13	87.20	92.33	-0.319	-0.210	-0.102	0.006
25	15	96.04	92.31	-0.358	-0.237	-0.116	0.006
50	14	98.15	92.18	-0.367	-0.243	-0.119	0.004
75	15	94.53	92.22	-0.325	-0.233	-0.114	0.005

Table 4. Internet Traffic Classification - VHDL Classifiers. Best solutions in terms of P are reported in bold.

throughput and hardware latency. We also presented an early prediction EP function to preliminary estimate the number of trees usable within the ensembles approaches according to the hardware constraints.

We presented the results of the proposed approach by considering two different problems: spam detection, a binary classification problem where both throughput and accuracy need to be maximized and the classification of IP traffic traces, a multi-class problem where latency need to be minimized, while preserving an high accuracy. In both cases, we considered different scenarios, by weighting in different ways the contribution of throughput (or latency) and accuracy to the overall system performance, by varying a suitably defined parameter.

In case of spam detection, we found that there was always a multiple classifier system that outperforms the single DT classifier, as could be expected. On the other hand, when the traffic classification problem was addressed, it happened that a single classifier solution is the most suitable one for several scenarios. The multiple classifier approach in this case should be preferred for an hardware implementation if we are only interested in maximizing accuracy.

As future work we are planning to investigate the possibility of extending our methodology to other multiple classifier systems.

Acknowledgments

The research leading to these results has been partially supported by the RoDyMan project, which has received funding from the European Research Council (FP7 IDEAS) under Advanced Grant agreement number 320992. The authors are solely responsible for its content. It does not represent the opinion of the European Community and the Community is not responsible for any use that might be made of the information contained therein.

References

1. Amato, F., Barbareschi, M., Casola, V., Mazzeo, A.: An fpga-based smart classifier for decision support systems. In: Intelligent Distributed Computing VII, pp. 289–299. Springer (2014)
2. Amato, F., Barbareschi, M., Casola, V., Mazzeo, A., Romano, S.: Towards automatic generation of hardware classifiers. In: Algorithms and Architectures for Parallel Processing, pp. 125–132. Springer (2013)
3. Dainotti, A., Gargiulo, F., Kuncheva, L.I., Pescapè, A., Sansone, C.: Identification of traffic flows hiding behind TCP port 80. In: ICC. pp. 1–6. IEEE (2010)
4. Dainotti, A., Pescapè, A., Claffy, K.C.: Issues and future directions in traffic classification. IEEE Network 26(1), 35–40 (2012)
5. Dainotti, A., Pescapè, A., Sansone, C.: Early classification of network traffic through multi-classification. In: Third International Workshop on Traffic Monitoring and Analysis, Vienna, Austria, April 27, 2011. pp. 122–135 (2011)
6. Dainotti, A., Pescapè, A., Sansone, C., Quintavalle, A.: Using a behaviour knowledge space approach for detecting unknown IP traffic flows. In: 10th International Workshop on Multiple Classifier Systems, Naples, Italy, June 15-17, 2011. pp. 360–369 (2011)
7. Franca, A.L.P.d., Jasinski, R.P., Pedroni, V.A., Santin, A.O.: Moving network protection from software to hardware: An energy efficiency analysis. In: 2014 IEEE Computer Society Annual Symposium on VLSI. pp. 456–461. IEEE (2014)
8. Gargiulo, F., Kuncheva, L.I., Sansone, C.: Network protocol verification by a classifier selection ensemble. In: Multiple Classifier Systems, pp. 314–323. Springer (2009)
9. Gargiulo, F., Mazzariello, C., Sansone, C.: Multiple classifier systems: Theory, applications and tools. In: Handbook on Neural Information Processing, pp. 335–378 (2013)
10. Guzella, T.S., Caminhas, W.M.: A review of machine learning approaches to spam filtering. Expert Systems with Applications 36(7), 10206 – 10222 (2009)
11. Kuncheva, L.: Combining Pattern Classifiers: Methods and Algorithms, 2nd edition. Wiley-Interscience (2014)
12. Mitchell, R., Chen, I.R.: A survey of intrusion detection techniques for cyber-physical systems. ACM Comput. Surv. 46(4), 55:1–55:29 (mar 2014)
13. Saqib, F., Dutta, A., Plusquellic, J., Ortiz, P., Pattichis, M.: Pipelined decision tree classification accelerator implementation in fpga (dt-caif). IEEE Transactions on Computers 64(1), 280–285 (Jan 2015)