An FPGA-Based Smart Classifier for Decision Support Systems

Flora Amato, Mario Barbareschi, Valentina Casola, and Antonino Mazzeo

Abstract. In recent years, the accuracy and performance of decision support systems have become a bottleneck in many monitoring applications. As for the accuracy, different classification algorithms are available but the overall performance are related to the specific software implementation. In this paper we propose a novel hardware implementation to fasten a decision tree classifier. We also present the evaluation of our architecture by putting in evidence the positive performance results obtained with the proposed implementation.

1 Introduction

In modern decision support systems there is the need to improve the performance in terms of detection, reliability and real time capabilities. These features are usually in contrast among each other. Furthermore, many monitoring systems rely on heterogeneous data acquisition tools (sensors, video, historical and simulated data,...) and on data elaboration and correlation to prune non significant information [8, 11]; this heterogeneity, even if desirable, introduces the need to further interpret what data really represents to reduce false alarms and detect even weak alarm conditions. The adoption of semantic techniques and inference knowledge is a must to cope with heterogeneity, even in large environment as in the clouds, nevertheless at the same time real-time capabilities must be ensured.

In some recent works [7, 8] we proposed an innovative approach for smart event detection and semantically enriched phenomena comprehension: the knowledge base is inferred off line for further comprehension and model prediction refinement, while a light classifier is used to quickly raise an alarm. In many monitoring applications, specific classifiers are adopted to elaborate huge amount of data and they

Flora Amato · Mario Barbareschi · Valentina Casola · Antonino Mazzeo
Department of Electrical Engineering and Information Technology
University of Naples Federico II, Napoli, Italy
e-mail: {flora.amato,mario.barbareschi,casolav,mazzeo}@unina.it

F. Zavoral et al. (eds.), *Intelligent Distributed Computing VII,*
Studies in Computational Intelligence 511,
DOI: 10.1007/978-3-319-01571-2_34,

need appropriate computational resources. A classification system could adopt hardware solutions to boost up performances but, the variable nature of classifiers (they need to periodically change their parameters when too false positives are counted or when some errors are caused by misconfigurations) does not match hardware characteristics. In fact, hardware cannot change its functionality, the only way to change hardware behaviour is to re-project the whole hardware. Indeed, approaches based on Field Programmable Gate Array (FPGA) technology, that allows to easily reconfigure hardware, can been used to implement classifiers, too. In some recent works [9] FPGA architectures were used for implementing algorithms for face detection, for image recognition and videosurveillance applications [6] or for real time object recognition. Indeed, a classifier is composed by a learner and by a predictor. The learner does not change its functionalities during application lifetime. It has to process the training set at start-up and the result of its computation defines how the predictor has to work. On the other hand, the predictor has to quickly evaluate and classify new data according the configuration parameters. In this paper we introduce an optimized hardware implementation based on reconfigurable hardware to implement a decision tree classifier. We also propose an application that automatically produces the VHDL code for FPGA synthesis, optimized for the chosen classifier. First experimental results are very promising, they report a response time of nanoseconds order. The reminder of the paper is structured as follows, in Section 2 the general architecture of a smart classifier is presented. In Section 3 some details of the adopted classifier is presented. In Sections 4 and 5 we illustrate in details the proposed hardware implementation to optimize the tuning and configuration of the Predictor component and some evaluation results will be discussed. Finally, in section 6 some conclusion and future work will be discussed.

2 A Two Steps Decision Support System

In previous works we proposed a smart Decision Support Systems (DSS) able to perform a quick classification of potential alarms and to refine the final decision based on semantic inferences on data and events [1, 2]. The system core was based on two different components:

1. a Smart Event Classifier;
2. a Post Reasoner.

In real environments, a complex potential hazard cannot be easily detected by using data from a single device, nevertheless sensor networks and data heterogeneity do not help in correlating data. So, the main idea behind that proposal was related to semantically enrich sensor data gathered by different sensors and correlate them for "operator-aware" event detection.

We developed a complex architecture made of different modules to: (i) semantically enrich data, (ii) implement the smart real-time classifier, (iii) implement the post-reasoner. In Figure 1, the DSS main modules are reported. We suppose that the sensor data source consists of heterogeneous sensor networks (WSN, camera,

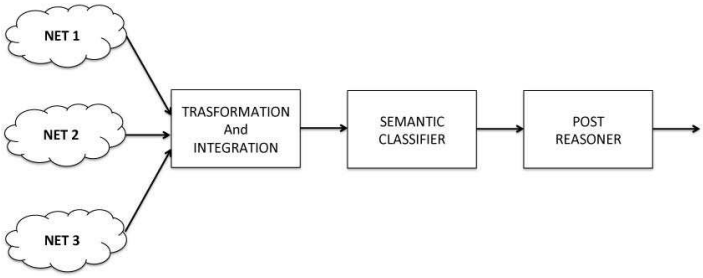

Fig. 1 Decision Support System architecture

intrusion device, etc) measuring different parameters. Data are gathered from sensor nodes and can be accessed through a specific sensor gateway. Gateway sensors code data in XML.

The system is made of three modules:

The **Transformation and integration module** gathers and integrates data coming from heterogeneous sensors. Data are here semantically enriched to add information and description about measures and sensors [1] and automatically encoded in RDF files. They contain unique references to descriptions of the measured variables and parameters of each network and are then integrated into a single RDF file, containing instances of the measured values that will populate the ontology.

The **Semantic Classifier module** implements a rule-based classifier: the combination of measurements allows classifying events, if there is a particular event, classified as critical, the system raises the corresponding alarm. The smart classifier operates on semantically enriched data, so, a rule can be expressed by combining atomic events from heterogeneous sources [3]. We chose a well-known classifier [12] to build the predictive model, it is a decision tree classifier made of a learner and a predictor component. The learner module uses a training set data in order to tune the predictor parameters, while the predictor is responsible to classify data and decide if alarm conditions occur. Furthermore, on a periodic basis or even when mis-classified events occur, the learner can recalculate the parameters of the predictor on the basis of a new training set, properly built to refine the behaviour of the classifier.

The overall performance, in terms of response time and accuracy of the whole decision system, rely on the performance and accuracy of this module and its design is very crucial. For this reason, in this paper we propose to synthesize it with a reconfigurable hardware as an FPGA. The expected advantages are twofold:

1. we expect to boost up the performance [13] and meet real time constraints,
2. we can easily reconfigure the whole predictor in case of parameter changing.

In the next Sections, we will illustrate in details this component and its hardware implementation.

The **Post Reasoner module** allows to analyze causes and reasons of the critical events that have taken place (basic function for an operator) and to refine the

procedures and rules of classification if a false alarm rate is not acceptable. Reasoning operations were performed on data in order to extract inferred knowledge recurring to a Pellet reasoner. The real time classification actions feed the knowledge base for the Post Reasoner component. The just built ontology is stored in a repository (Triple Stores) and can be used off-line through the adoption of semantic query languages as SPARQL. Through queries, the post reasoner is able to understand and explain to end users the meaning of the alarms and their causes. Details about this component are out of the scope of this paper and can be found in some previous works [7].

3 The Fast Classification Algorithm

In order to semantic classify heterogeneous sensed data, we have implemented, among the different classifiers available in literature, the decision tree classifier proposed by [12]. The well-known classifier is composed by a learner module for building the predictive model and a predictor component for performing decision activity on the data. In binary decision mechanisms and the set of decision rules are modelled as a tree where nodes represent classes associated to atomic events to be detected, and branches represent conjunctions of conditions; i.e. conditions on the sensed data that lead to composed event classes. In order to define the branch rules of the decision tree, a domain expert manually defines a training set, made of already classified data. The predictor is a parametric system: parameters, given by the learner module, determine the classification function. Periodically, or when mis-classified events happen, the learner can recalculate the parameters on the basis of a new training set in order to refine the behavior of the classifier. Indeed decision trees are chosen with the aim of easing further refinement [10]. In fact, it is a white box model and domain operators will be able to easily interpret decision tree results after a brief explanation. This kind of model allows to quickly identify if specific conditions of mis-classification occur and, consequently, to re-tune the model. Moreover if a given situation is observed in the model, it is easy to explain the conditions by recurring to the boolean logic. The Classifier takes in input the semantically enriched data codified in RDF. By using an XLST engine these data are reported in a table containing a row for each sample and a column for each relevant measured parameter. Element i, j, then, contains the measure value of the sample i for the parameter j. In the remainder of this section, we are going to give some details on the classifier algorithm and how to use it. These details will be useful to understand the different optimization that can be performed at hardware level.

The algorithm describes the fundamental steps for building the model, it works recursively on data taken from the training set. At each step the data set is split, based on a condition of a chosen feature (defined by thresholds). The selection of the feature is performed on the basis of an *Entropy Test* [4].

If T is the set of samples, and $freq(C_j, T)$ is for the number of samples in T that belong to class C_j . Let $|T|$ be the number of samples in the set T. Then the entropy of the set T is defined as:

Data: Training Set T, Output Class $C = C_1, ..., C_k$
Result: Prediction Model outputted by learner
`/* Base Case */`
if *T f T contains one or more examples, all belonging to the same class* C_j **then**
  Create a single leaf in which all the sample having label C. `/* The decision tree for T is a leaf identifying class` C_j `*/`
end
if $T = \emptyset$ **then**
  Creates a single leaf that has the label of the most frequent class of the samples contained in the parent node `/* heuristic step leading to misclassification rates */`
end
`/* Recursive Case */`
if *T contains samples that belong to several classes* **then**
  foreach *Feature f* **do**
    Find the normalized information gain by splitting on f, based on an Entropy Test Value
  end
  Let f_i be the attribute with the highest normalized information gain; Create a decision node that splits on f_j;`/* node associated with the test */`
  Create a branch for each possible outcome of the test; Execute the algorithm on each branch (corresponding to a subset of sample) `/* partitioning of T in more subsets according to the test on the attribute obtained by splitting on` f_j `*/`
end

Algorithm 1. Description of the algorithm for building a decision tree model

$$Info(S) = -\sum((freq(C_j,T)/T) \cdot log_2(freq(C_j,T)/T))$$

The algorithm computes the value of *Info* for all T_i partitioned in accordance with n conditions on a feature f_i:

$$Info_{f_{i(T)}} = \sum((T_i/T) \cdot Info(T_i))$$

The features that maximizes the following Gain value are selected for the splitting:

$$Gain(f_i) = Info(T) - Info_{f_{i(T)}}$$

Running this algorithm, we built a tree data structure; evaluating the conditions requires the visiting of all paths in the tree. In the predictor module, even in the best case of a balanced tree, the complexity of visiting a tree path, in order to evaluate all the conditions leading to a leaf, is $O(log_2(n))$, with n the number of tree nodes. Consequently, implementing the predictor in software implies that the node conditions of a given tree path are evaluated sequentially. We propose a reconfigurable hardware architecture to reduce the time spent in the decision tree path discovery, by elaborating the whole algorithm in only 2 sequences. Hardware implementation, intrinsically concurrent, of the predictor allows us to elaborate the conditions of the

nodes of the tree branches in a concurrent way. Furthermore hardware implementation allow us to set real time constraints, too. In fact, during the hardware design phase, we can estimate the delay associated with the critical path and we can build the real time conditions on the basis of such values. In the next sections we will show that our hardware approach increase the throughput of 10^3 times, providing results of the order of $\approx 10ns$.

4 An Optimized Hardware Implementation of the Decision Tree Predictor

To get an hardware description of the predictor, first of all we needed the training set. So we collected some meaningful data and we manually labelled them with a classification value. The training set is the input of an automated process which produced the required hardware.

In this process we can identify two phases. The first one aims at defining the learner, that computes the predictor parameters. We choose KNIME [5], a data mining framework, to complete the first phase. With the decision tree blocks of KNIME, we are able to retrieve the predictor parameters. Generally this phase is done off-line, periodically or when errors, caused by a misconfiguration, occur. For this reason there are not strict time constraints in this phase. The second phase aims at building and using the predictor. Due to real time constraints, we will implement it in hardware. In this paper we proposed an application that automatically produces the VHDL code for FPGA synthesis, optimized for the chosen classifier.

4.1 Predictor Phase

As previously said, a decision tree predictor is typically implemented as a visiting algorithm, that is an exploration of the tree, from the root to leaves, node by node in a sequential way. In fact the algorithm has to evaluate each decision to determine in which branch it has to continue until it reaches a leaf. We propose a hardware architecture to dramatically reduce the time spent in the decision tree path discovery, computing the whole algorithm in only 2 sequences. Thanks to the hardware features, we implemented the visiting algorithm with a different approach:

1. We simultaneously evaluate all predicates contained in all tree nodes;
2. We automatically define a boolean function that implements the visit and verify which leaves are effectively reached according to the classified class.

All the decisions are not computed in sequence, but in parallel. Each decision is a boolean value: for this reason the visiting algorithm can be treated as a multi-output boolean function, in which the inputs are the decisions and the outputs the classes. Also this boolean function could be optimized and reduced in order to increase the performance. It can be implemented as a sum of products (SOP); in particular a path that leads to a given class C_i is implemented as an *anding* function of the boolean

decisions along the path. All *anding* functions associated to different paths that lead to the same class C_i are in *oring*:

$$C_i = \bigvee_{P \in Path\ set\ of\ C_i} \left(\bigwedge_{Decision\ Node \in P} decision \right)$$

We can characterize a single class function by the *Class Characteristic*, that is the maximum path length plus the number of paths that lead to the class. As shown in Figure 2, our architecture computes all the decisions (rounded rectangles) in parallel, and each result is given in input to the boolean net, that decides which class the input belongs. In particular, to compute all decisions very quickly, we have implemented in VHDL a generic *Decision Box*. It computes a predicate between two values: $D = A\ \rho\ C$. The effective input of the *Decision Box* is only the first operand (A) representing the new data, while the operation (ρ) and the second operand (C) are pre-defined, decided during the learning phase. The output D is a boolean variable.

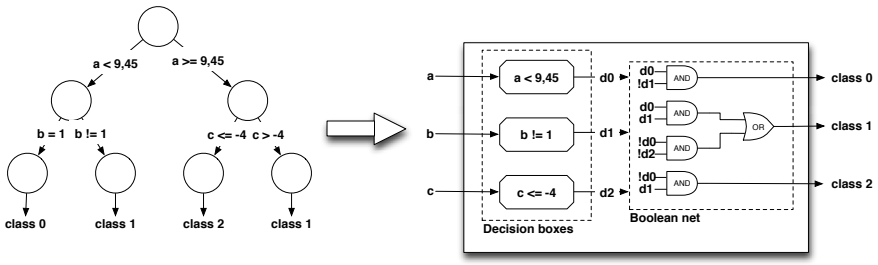

Fig. 2 Predictor hardware architecture

The *Decision Box* has 3×6 different configurations, as it supports 3 different data types (single precision floating point, 32 bit signed integer, 32 bit unsigned integer) and 6 different comparing operations ($<$, $<=$, $==$, $!=$, $>=$, $>$).

The *Decision Boxes* outputs, the decisions, have to be analyzed by the second part of the architecture to classify the input data. In our architecture this task is executed by the *Boolean Net*.

These 2 components are automatically produced by the PMML2VHDL tool in 2 VHDL files at the end of the Learning phase, described in the next section.

4.2 Learning Phase

In the Learning phase, we used the KNIME tool that supports us for the entire analysis process and to realize the learner, accordingly to the algorithm previously described. It offers data access, data transformation, initial investigation, powerful predictive analytics, visualization and reporting in an integrated graphical interface.

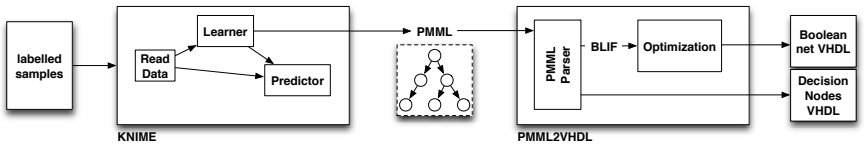

Fig. 3 Learning phase

As illustrated in Figure 3, in the learning phase, KNIME loads the training set file containing data that were manually labelled and previously classified. According to the classification process, KNIME produces in output an XML file, the PMML, that describes the obtained tree.

We developed an application, named PMML2VHDL, that automatically produces the VHDL files describing the Predictor. The Predictor hardware has been optimized for this specific classifier, for this reason we have developed two different components and, consequently, the PMML2VHDL application will produce two different hardwware description files: the*Decision Nodes VHDL* and *Boolean Net VHDL.*

The first one contains all predicates that have to be evaluated. The second describes the optimized boolean net for class assignment, so it contains a function for each class defined into the decision tree predictor. To produce an optimized boolean net we used Berkeley SIS tool. So PMML2VHDL tool produces an intermediate result of the boolean net, that is the raw net described in BLIF format. With heuristic method, SIS minimizes and optimizes the net, returning another BLIF file. This last file is translated in a VHDL data flow description, ready to be synthesized.

5 Evaluation of the Proposal

In this section we are going to illustrate some experimental results about our proposal. We took about 100 samples from a local sensor network. The sensors collected temperature, humidity, pressure, battery level and GPS position every 10 seconds. We manually labelled data with an alarm value to define the training set and configure the predictor parameters with KNIME. We generated and optimized the VHDL with our application.

We have performed several experiments by changing each time the number of alarm classes and the training set size and we used Xilinx ISE and a Virtex-5 XC5VLX110T to synthesize and test the architecture. We evaluated the response time (the throughput) of our FPGA-based architecture and the area used to implement the components. Note that algorithm precision is not evaluated as it is not affected by this implementation.

The **Decision Box** implementation can have 3×6 different configurations. To obtain a higher throughput we implemented *Decision Box* in pipelined version with depth 2. In Table 1 the main implementation characteristics are reported for the

Table 1 Evaluation of *Floating Point Decision box* with latency 2

Parameter	<	==	<=	>	!=	>=
Slices LUTs	43	27	44	32	44	44
Slices Flip-Flops	43	28	45	33	45	45
Delay (ns)	1,779					
Throughput (float/s)	562113546,9					

6 different operations; the interesting result is that we get about 560 millions of computed decisions per second, with an used area of about 44 LUTs per box.

The **Boolean Net** delay depends on the predictor tree implementation, as all inputs (the decisions) have the same arrival time. Our hardware implementation computes a classification for a given class as a SOP. For this reason the delay time for an output class depends on the number of paths that leads to the class and on the maximum path length. In fact in the SOP implementation the delay is a function of the higher fan-in of product terms and of number of product terms. In the previous section, we defined the *Class Characteristic* to get a meaningful parameter for comparing the *Boolean Net* delays. As shown in Figure 4, the delay does not linearly increase with the *Class Characteristic* but it has a step behavior. This is primary due to the the chosen FPGA, as the gate delay is the same of a LUT if the fan-in is less than or equal 6, so for similar *Class Characteristics* the delay was the same.

It is also possible to note that the delay time is significantly longer than *Decision Boxes* time, about 7ns, but this value is considerably better than a software implementation of the predictor module; indeed, with KNIME [5], we obtained a delay of $\approx 10ms$.

To avoid bottle-neck effect on the throughput, it's possible to implement a high performance pipelined version, too.

Finally, we considered the *Boolean Net* used area in terms of FPGA LUTs. By using the Virtex5 Family we have 4 LUTs for each slice. As illustrated in Figure 5, results show a linear trend with the growing of function literals: the mean slope is about 11,84 lits/LUTs.

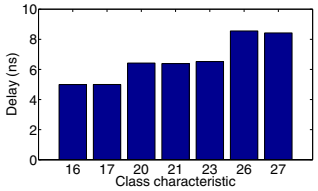

Fig. 4 Step behavior of *Class Characteristic* and net delay

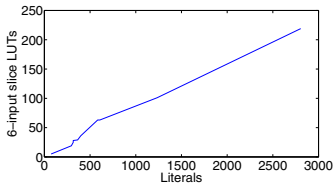

Fig. 5 Used FPGA LUTs on number of literals

6 Conclusions and Future Work

In recent years, the need for smart monitoring systems has grown and, in particular, the accuracy and performance of decision support system has become a bottleneck. As for the accuracy, different classification algorithms are available but the overall performance is related to the specific software implementation.

In this paper we proposed a process to implement in hardware a reconfigurable decision tree classifier. Furthermore we proposed an innovative architecture to fasten the classification process, it is composed by a set of *Decision Boxes* to compute in parallel all decisions and by a *Boolean Net*, to effectively compute the classification. We evaluated the proposal from different prospectives, putting in evidence the great performance obtained with the hardware implementation. In future work we intend to enhance the proposal by introducing automatic pruning rules in the tree model with the application, in order to improve performance of the predictor *Boolean Net*, that is the most critical component.

References

1. Amato, F., Casola, V., Gaglione, A., Mazzeo, A.: A common data model for sensor network integration. In: Proceedings of the 4th International Conference on Complex, Intelligent and Software Intensive Systems, pp. 1081–1086 (2010)
2. Amato, F., Casola, V., Gaglione, A., Mazzeo, A.: A semantic enriched data model for sensor network interoperability. Simulation Modelling Practice and Theory 19(8), 1745–1757 (2011)
3. Amato, F., Casola, V., Mazzeo, A., Romano, S.: A semantic based methodology to classify and protect sensitive data in medical records. In: 2010 Sixth International Conference on Information Assurance and Security (IAS), pp. 240–246. IEEE (2010)
4. Berger, A.L., Pietra, V.J.D., Pietra, S.A.D.: A maximum entropy approach to natural language processing. Computational linguistics 22(1), 39–71 (1996)
5. Berthold, M.R., Cebron, N., Dill, F., Gabriel, T.R., Kötter, T., Meinl, T., Ohl, P., Thiel, K., Wiswedel, B.: Knime - the konstanz information miner: version 2.0 and beyond. SIGKDD Explor. Newsl. 11(1), 26–31 (2009)
6. Bhowmik, D., Amavasai, B.P., Mulroy, T.J.: Real-time object classification on fpga using moment invariants and kohonen neural networks. In: IEEE SMC UK-RI Chapter Conference 2006 on Advances in Cybernetic Systems, pp. 43–48 (2006)
7. Casola, V., Esposito, M., Mazzocca, N., Flammini, F.: Freight train monitoring: A case-study for the pshield project. In: Proceedings - 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2012, pp. 597–602 (2012)
8. Casola, V., Gaglione, A., Mazzeo, A.: A reference architecture for sensor networks integration and management. In: Trigoni, N., Markham, A., Nawaz, S. (eds.) GSN 2009. LNCS, vol. 5659, pp. 158–168. Springer, Heidelberg (2009)
9. Cho, J., Mirzaei, S., Oberg, J., Kastner, R.: Fpga-based face detection system using haar classifiers. In: FPGA, pp. 103–112 (2009)
10. Deng, H., Runger, G.C., Tuv, E.: Bias of importance measures for multi-valued attributes and solutions. In: Proceedings of the ICANN, pp. 293–300 (2011)
11. Ficco, M.: Security event correlation approach for cloud computing. Journal of High Performance Computing and Networking 7(3) (2013)

12. Liu, B., Ma, Y., Wong, C.K.: Improving an association rule based classifier. In: Zighed, D.A., Komorowski, J., Żytkow, J.M. (eds.) PKDD 2000. LNCS (LNAI), vol. 1910, pp. 504–509. Springer, Heidelberg (2000)
13. Wittig, R.D., Chow, P.: Onechip: An fpga processor with reconfigurable logic. In: IEEE Symposium on FPGAs for Custom Computing Machines, pp. 126–135. IEEE (1996)