IP Cores for Hardware Acceleration of Decision Tree Ensemble Classifiers

R. Struharik
University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia
rasti@uns.ac.rs

***Abstract*— This paper proposes four different hardware architectures for parallel implementation of decision trees forming an ensemble classifier are presented. Proposed architectures can accelerate ensemble classifiers composed of axis-parallel, oblique and nonlinear decision tree (DTs). Hardware architectures for the implementation of a number of combination rules are also presented, enabling the complete ensemble classifier hardware accelerators. Conducted experiments, based on 29 UCI datasets, indicate that the Field Programmable Gate Array (FPGA) implementations based on proposed architectures offer significant improvement in the instance classification time in comparison with the traditional software implementations.**

***Keywords* – machine learning, decision trees, ensemble classifiers, hardware acceleration, VHDL, FPGA**

I. INTRODUCTION

Machine learning [1-2], a branch of artificial intelligence is concerned with developing algorithms for construction of systems (predictive models) that can learn from data. A wide range of different machine learning predictive models have been introduced, including decision trees (DTs) [3-4], support vector machines (SVMs) [5] and artificial neural networks (ANNs) [6]. Machine learning predictive models have been widely used in data mining [7], among which DTs, SVMs and ANNs are most popular [8].

To reduce dependence on the peculiarities of a single training set and enable learning of more expressive concept classes, ensemble classifier systems [9] have been proposed. The main idea behind ensemble classifiers is to create a set of classifiers and combine their predictions into a collective decision. The main advantage of ensemble classifiers over single classifiers is in the higher accuracy and greater robustness. The price to be paid is manifested in large amounts of memory needed to store the ensemble classifier and high computing power.

In large-scale classification problems reduction of individual instance classification time is one of the key requirements. Reduction of classification time typically appears in data mining [7], or in classification problems requiring real-time data processing (e.g. machine vision [10], bioinformatics [11], web mining [12], etc.). Developing new algorithms or software tools is prevailing way in meeting this requirement [13-14].

In the course of reduction of individual instance classification time, hardware implementation of machine learning classifier systems is a promising alternative to traditional software-based approaches. Most of existing hardware implementations are concerned with acceleration of individual machine learning classifier systems, DTs [15-16], SVMs [17-18] and ANNs [19-20].

In this paper we present four different hardware accelerators of DT ensemble classifiers intended to be used in embedded applications. All these accelerators uses parallel ensemble member evaluation approach, which results in significant acceleration of the instance classification process, when compared with the software implementation running on the embedded processor core.

This work was partially supported by the Serbian MNTR grant No. TR32016.

I. DECISION TREE ENSEMBLE CLASSIFIERS

In a decision tree classifier, function to be learned is represented by a tree of finite depth. Each node in the tree specifies some test of problem attributes, represented by the $n+1$ element vector (A_1, A_2, A_3, …, A_n, *class*), where n designates the number of problem attributes, and each branch descending from that node matches one of the possible outcomes of that test. Instance is classified starting at the root node, testing the attribute(s) specified in this node, and then descending along the branch matching the result of the attribute(s) test. This is repeated until some leaf is reached.

Most DTs use tests that involve only a single attribute of the problem

$$A_i > a_i \tag{1}$$

and equal axis-parallel hyperplanes in the attribute space. Advantage of axis-parallel DTs is that they can be interpreted easily, since tests described by (1) are easy to interpret, but if training data is more easily partitioned by non-orthogonal hyperplanes, then inducing axis-parallel DTs may result in complicated and inaccurate trees.

If multivariate tests are allowed, then at least for some classification problems, created DTs will be much smaller and possibly have better classification accuracy when compared with the axis-parallel DTs. Multivariate tests use linear combination of several problems attributes

$$\sum_{i=1}^{n} a_i A_i + a_{n+1} > 0 \tag{2}$$

DTs that use multivariate test are called oblique DTs.

Finally, tests in every node can have a more general form than that of (2), being any function of the problem attributes. If this function is non-linear, resulting DTs are called non-linear DTs. In practice, non-linear DTs use non-linear tests that are represented as a polynomial of n attribute variables.

The main idea behind ensemble classifier is to combine predictions from the set of diverse individual classifiers (two classifiers are considered as diverse if they make different errors on new instances) using appropriate combination rule. Under these assumptions an ensemble classifier could achieve an arbitrary classification accuracy using the power of making collective decision. In the open literature there are a variety of algorithms for creation of diverse ensemble members, such as Bagging [21], Boosting [21], AdaBoost [21], Stacked generalization [22], Mixture of experts [23].

Once the individual members of an ensemble are created, a procedure for combining their outputs to reach collective decision is needed. Many different combination rules have been proposed over time, most popular being different versions of majority voting and behavior knowledge space [24].

Majority voting is probably the most commonly used combination rule. There are many different versions of this combination rule proposed in the open literature: unanimous voting, simple majority voting, plurality voting and weighted majority voting.

When unanimous voting rule is used, all ensemble members must agree on the instance classification before ensemble makes a collective classification. If this is not possible, ensemble cannot reach a collective decision.

In simple majority voting it is sufficient that more than half of ensemble members agree on the instance classification. Plurality voting rule selects the class that has the largest number of individual ensemble member's votes. The difference from simple majority voting rule is that in this case there is no requirement that more than half individual members must vote for this particular class.

Finally, in weighted majority voting rule, each member from the ensemble has a weight assigned to it. Different members can have different weights assigned. Assigned weights most commonly represent some form of relevance that the user has in the ensemble member classification. Members that we are more confident with will have larger weights assigned to them and vice versa. When making a collective decision, ensemble classifier makes a weighted sum of individual member classifications, and selects the class with the largest weighted sum.

A different approach is taken with the behavior knowledge space combination rule. In this approach during instance classification, a classification vector, comprising from individual member classifications is stored. Every instance from the training set will have its own classification vector assigned to it. After all training set instances have been classified, a count is made to see how much the same classification vector appears at the outputs of the ensemble members for all possible class membership values. Using this information, every classification vector is then assigned to the class value for which it is most frequently observed. This data is stored in a table that is then queried during the instance classification in the deployment phase.

II. Architectures for Hardware Implementation of Decision Tree Ensembles

In order to implement ensemble classifier in hardware, two separate modules are required: ensemble evaluator module (EM) - to evaluate ensemble members; combiner module (CM) - to combine individual classifications into a collective decision. Since these two tasks are independent of each other, these two modules can operate in a pipelined fashion. To achieve maximum performance, both of these modules should process data in parallel. First, we will propose two architectures for the implementation of EM module.

A. Ensemble member evaluator architectures

In Figure 1, EM module performs the parallel evaluation of the DTs from ensemble, having *N* parallel DT evaluator modules originally proposed in [25]. Every DT from the ensemble is implemented by a separate DT evaluator module, using one of two possible architectures: Single Path (SP), which uses serial evaluation of the node tests given by (1) - (3), or Single Path 2 (SP2), which uses parallel evaluation of the node tests. SP architectures use a number of pipeline stages to implement a DT. Number of required pipeline stages is equal to the depth of the DT that is being implemented. Since individual DTs from ensemble are most frequently trees with different depths, there will be a problem that the classifications of the same instance by different DTs will not reach the CM module at same time instances, due to the different propagation delays. To overcome this problem, at the end of the pipeline chain of every DT that is shallower then the deepest DT from the ensemble, a delay line needs to be introduced in order to further delay the classification information thus ensuring that it will reach the CM module at the same time instance as the classification information from the deepest DT.

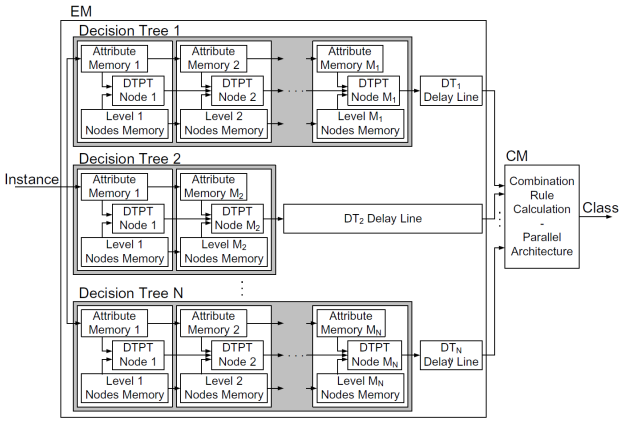

Figure 1. Parallel architecture for ensemble classifier implementation based on SP architecture

Basic SP architecture is present on Figure 1. As have been already mentioned SP architecture is a pipelined architecture for DT implementation. It consists from a number of pipeline stages, each stage evaluating all DT nodes found at the same DT level. Node evaluation requires calculation of the node test, specified by (2). Depending on the node outcome, current instance is forwarded to the next pipeline stage (next DT level) together with the information which node from next DT level should evaluate this instance in the next time instance.

SP architecture consists from three major modules. Module M1 is a small memory, used to store the attribute values of the current instance that is being evaluated. M1 modules from neighboring pipeline stages are connected

together, forming a chain streaming instances through the system.

Module M2 is the central module of the SP architecture. It is used to evaluate selected node test by evaluating (2), either in sequential or parallel fashion. During select node test evaluation, attribute values of the instance currently stored in the M1 module are used, as well as the coefficient values related to the selected DT node that are stored in the final module of the SP architecture, module M3. Please notice that the proposed architecture is capable to evaluate tests found in axis-parallel, oblique and non-linear DTs, because they all can be reduced to (2).

Module M3 is used to store the coefficients a_i from (2) for all nodes located at the same DT level. Beside this, M3 module also stores the addresses of two successor nodes, located at the next DT level. Depending on the outcome of the node test, one of these two successor node will be visited and evaluated on the current instance in the next processing cycle. Furthermore, M3 module also stores the class membership information, in case some of its successor node is actually a leaf. In this case instance classification process will finish and it will be classified into a class that is associated with the reached leaf.

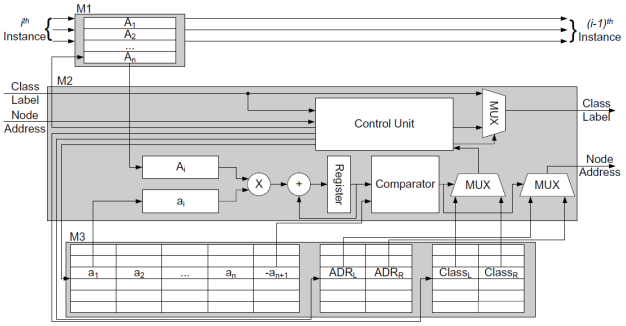

Figure 2. Detailed architecture of one pipeline stage of SP module

For more detailed explanation of SP architectures, please refer to [25].

Alternative version of the implementation of DT evaluation module is based on the architecture that sequentially traverses the DT during the instance classification process, once more originally proposed in [25]. Structure of the ensemble accelerator system, when Single Node architecture (SN) is used for the parallel implementation of DT ensemble, is shown in Figure 3. Every DT from the ensemble can be implemented using one of the two proposed SN architectures: Single Node (SN), with the serial evaluation of the node tests, specified by (2), or Single Node 2 (SN2), with the parallel evaluation of the node tests.

Since the time required for classifying instance depends on the structure of the DT that is used to classify it, different DTs from the ensemble will require different time period to classify same instance. This will result in classifications appearing at the outputs of the EM module not at the same time, which is necessary requirement for the CM module. Synchronizer module, shown in Figure 3, has the function to synchronize arrival of the classification information generated by individual DTs from ensemble at the outputs of the EM module.

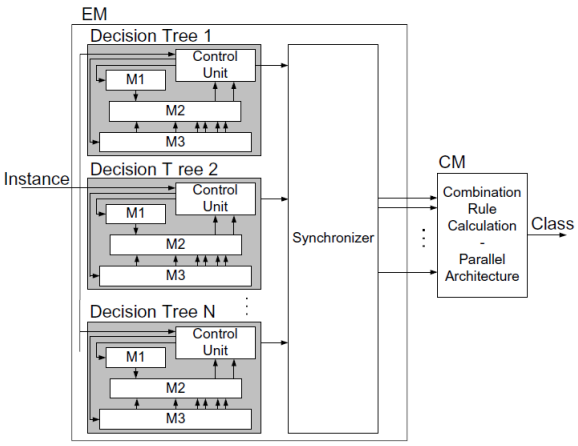

Figure 3. Parallel architecture for ensemble classifier implementation based on SN architecture

Opposed to the SP architecture, SN architecture evaluates DT in a sequential fashion one node at a time. No pipelining is used in the SN architecture, making it much slower than SP architecture, but much more area efficient. Basic structure of the SN architecture is presented in Figure 4.

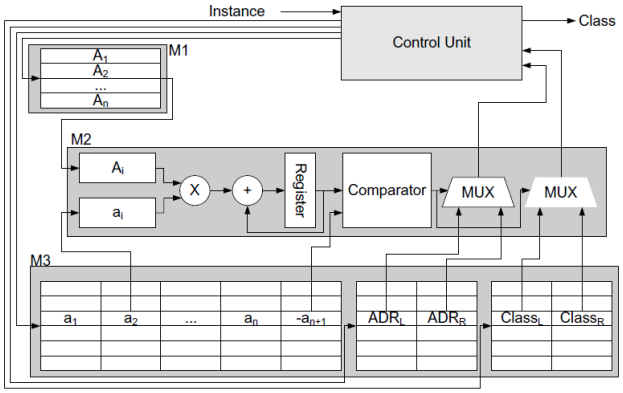

Figure 4. Detailed architecture of the SN module

Major modules of the SN architecture are instance attribute memory module (M1), currently active node test evaluation module (M2), DT structural information storage module (M3), and Control Unit module.

Module M1 is similar to the M1 module found in every pipeline stage in SP architecture. It is used to store attribute values of the instance that is currently being classified by the DT. Since in SN architecture there is only one M1 module, there is no pipeline chain of these modules as was the case in the SP architecture.

Module M2 is identical to the M2 module from the SP architecture, with one exception; there is no FSM inside it. It is used to evaluate the selected node's test. Similar to the SP architecture, in SN architecture also this test can be evaluated in sequential manner or in parallel manner, resulting in two variations of the basic SN architecture, SN and SN2.

Module M3 stores the structural information of the DT that is being implemented by the SN architecture. Structural information contains: coefficient values a_i from (2) specified for every DT node, next node address values and class information values for all leaf nodes. For more detailed explanation of SN architecture, please refer to [25].

B. *Architectures for combination of individual member's predictions*

In order to build a complete ensemble classifier in hardware, besides implementing individual ensemble members, additional module that will combine individual ensemble member's prediction is needed. This is the function of CM module shown in Figures 1 and 3. Please notice that the CM module operation is independent of the type of individual ensemble members. They can be of any type of machine learning predictive models (DTs, SVMs or ANNs), until they produce a class membership information at their output. In this sense, architectures for hardware implementation of different combination rules are universal, meaning that they can be used to combine class membership values coming from different type of machine learning classifiers, not only DTs. Since proposed architectures for DT ensemble classifier hardware implementation presented in Figures 1 and 3 implement individual ensemble members in parallel, CM module must also be able to process the incoming classification information in parallel in order not to compromise the speed improvement gained by implementing individual members in parallel.

For majority voting rules, Majority Vote (MV) architecture presented in Figure 5, able to process the individual classifier votes in parallel, should be used. MV architecture can be used to implement all majority voting rules, except the weighted majority rule.

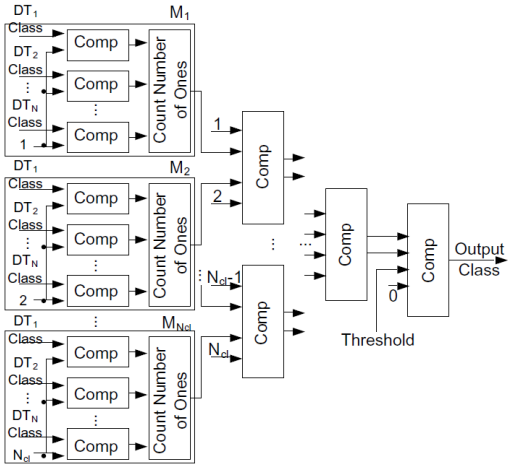

Figure 5. Architecture for parallel implementation of majority voting rules

At the inputs of the MV architecture instance classification from individual ensemble members (DT_i_Class) arrives in parallel. This classification information is fed to a set of class counter modules (M_1, M_2, M_{Ncl}). Number of class counter modules equals the number of classes instance can be classified into. Each class counter module counts the number of ensemble members that have classified the current instance into a given class. Since this operation must be performed in parallel, each class counter module uses a set of N equality comparators, N being the number of members of ensemble, to determine which member has classified the current instance into a current class. At the outputs of the comparators a N-bit binary vector appears which is then converted into a binary number of $\lceil \log(N) \rceil$ bits by the simple combinatorial network that counts the number of ones in this vector.

Once we have the information how many ensemble members have classified current instance into every possible class, next step is to find the class with maximum number of votes. This process is achieved by a tree of comparators shown in Figure 5. Each comparator has four inputs and two outputs. Two inputs are the actual number of ensemble members that have voted the current instance into a given class, but each of them is also accompanied with the class value. Comparator selects the larger number of votes and transfers it to one of its outputs. The second output holds the class value with the larger number of votes. Since the process of finding the class with maximum number of votes must be performed in parallel, a tree of these comparators is needed. After each level in the comparator tree there are pipelined registers, which are not explicitly shown in Figure 5, in order to achieve maximum throughput. After final comparator of the comparator tree there is one additional comparator that compares the maximum vote count value with a programmable threshold. By carefully choosing this threshold value different versions of majority voting rule, discussed earlier, can be implemented by the architecture from Figure 5.

If threshold value is set to N-1, unanimous voting rule can be implemented. Only if all N ensemble members have voted for the same class, its code will appear on the *Output Class* port. Otherwise, class code zero will appear on the Output Class port, indicating that ensemble cannot make a decision about current instance's class membership.

In case of simple majority voting rule implementation, threshold value should be set to $\lfloor N/2 \rfloor$, and plurality voting rule can be implemented if we set the threshold value to zero.

To be able to implement weighted majority voting rule, small modification of the class counter modules is required. New architecture of these modules is shown in Figure 6. After making the comparison of every ensemble member with the specified class value, instead of generating a simple 0 or 1 output, weight assigned to the appropriate classifier that has voted for this class is transmitted instead. These weights are then summed using a pipelined adder tree. Calculated weighted sum of all ensemble members is generated at the output of every class count module and forwarded to the comparator tree from Figure 5.

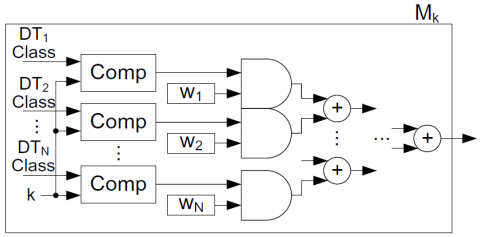

Figure 6. Structure of the module M_k responsible for the total weighted vote calculation for the class k in case of weighted majority voting rule

In case of Behavior Knowledge Space (BHS) rule implementation a lookup table is needed that will hold the mappings of generated classification vectors into appropriate class membership values. This look-up table is

implemented as a common single port RAM memory. The most difficult part in implementing BHS rule is the look-up table address calculation process, described by the (3).

$$adr = \sum_{i=1}^{N} \left(DT_i\_class - 1\right) \cdot \left(N_{cl}\right)^{i-1} \qquad (3)$$

Since classification vector is calculated in parallel, we need an architecture that can perform address calculation for the look-up table in parallel also. One architecture, capable of parallel calculation of the look-up table address, is shown in Figure 7. Proposed architecture has an array of multipliers as its first layer. These multipliers perform the necessary multiplications from (3) in parallel. After multiplication results are calculated they are summed-up in parallel by a pipelined adder tree. At the output of the pipelined adder tree, address of the look-up table location where the class membership information for the current instance is stored appears every clock cycle.

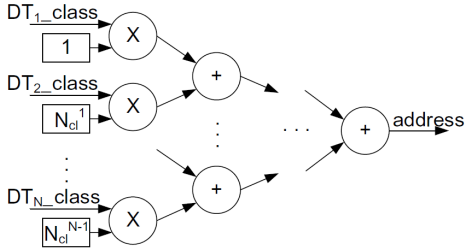

Figure 7. Architecture for the parallel calculation of the look-up table address value

III. EXPERIMENTS

Hardware accelerated ensemble classifiers should have superior performance in terms of instance classification speed when compared with the pure software implementations. To measure possible classification speedup of proposed hardware accelerators over traditional software implementation following experiment has been conducted.

Software implementation of DT ensemble classifier system was developed using C programming language, compiled with the GCC compiler and executed on the Xilinx MicroBlaze embedded processor. Four proposed hardware accelerators of DT ensembles (EM-SN, EM-SN2, EM-SP and EM-SP2), as well as embedded system based on the MicroBlaze processor, were implemented using the Xilinx Kintex-7 FPGA device. All four hardware ensemble accelerators have used the same CM module, based on the MV architecture. All hardware architectures were implemented using the Xilinx Vivado Design Suite and SDK v2014.4 software.

MicroBlaze-based embedded system was running at 100 MHz, while EM-SN, EM-SN2, EM-SP and EM-SP2 hardware accelerators were operating at, on average, 180.71, 111.17, 182.74 and 218.43 MHz respectively.

For all hardware accelerators, as well as for the software implementation, average instance classification time has been measured. In the speedup experiment, selected 29 datasets from the UCI Machine Learning Repository database [26] have been used.

For software implementation, each dataset has been loaded into the memory before initiating the classification process. All instances from the selected dataset have been classified by the ensemble classifier and total classification time has been measured by a hardware timer present in the MicroBlaze system. Dividing this total classification time by the selected dataset size average instance classification time was estimated.

In case of hardware accelerators selected dataset was once more loaded into the on-chip memory before starting the classification process. All instances from the selected dataset have been classified by the hardware accelerator currently implemented in the FPGA and total dataset classification time was measured by a hardware timer present in the system. Dividing this time by the dataset size average instance classification time for hardware accelerators has been estimated. This procedure was then repeated for all four hardware accelerators.

Collected average instance classification times for 29 different UCI datasets and five different hardware and software implementations of ensemble classifier have been then used to calculate the average classification speedups for four proposed hardware accelerators over the pure software implementation. Estimated classification speedups are presented in the Table 1.

TABLE I. SPEEDUP RESULTS FOR DIFFERENT HARDWARE ARCHITECTURES OVER PURE SOFTWARE IMPLEMENTATION

Dataset	EM-SN	EM-SN2	EM-SP	EM-SP2
ausc	31.24	268.47	1168.37	21620.08
bc	24.29	144.68	939.90	11594.85
bcw	110.13	656.69	510.99	6303.75
bsc	25.67	115.47	946.74	5839.62
car	25.14	154.75	981.14	8472.51
cmc	48.57	288.92	2261.50	25480.95
ger	36.79	519.96	1396.57	39339.01
gls	33.49	218.16	1349.46	18312.09
hep	290.98	3220.26	389.91	9620.18
hrtc	31.21	267.25	1166.65	20148.93
hrts	23.83	204.02	821.52	14188.36
ion	22.41	413.30	757.55	29874.21
irs	11.74	52.88	394.02	2430.36
liv	36.74	226.14	1463.89	12641.25
lym	20.87	219.35	702.06	16455.57
page	34.18	222.47	1475.91	20027.97
pid	37.33	265.12	1528.24	16967.52
son	19.86	607.27	648.17	44549.27
thy	14.55	73.79	478.84	3544.28
ttt	31.18	185.82	1141.01	14075.84
veh	39.84	422.98	1674.14	35839.81
vote	14.36	139.27	463.55	9721.44
vow	41.77	272.01	1645.89	22334.57
w21	42.44	518.70	1952.37	52986.93
w40	43.87	918.44	2754.92	90688.46
wdbc	16.82	276.22	556.07	21265.66
wine	14.20	121.69	448.87	7752.40
wpbc	24.36	436.33	823.28	31539.03
zoo	19.09	244.50	630.98	14011.14
Average	**40.24**	**402.58**	**1085.26**	**21642.28**

From Table 1 it can be seen that all proposed hardware accelerators offer improvement in the classification speed over the software implementation running on MicroBlaze soft embedded microprocessor. Average speed-up of the EM-SN accelerator over the software implementation is 40.24 times. EM-SN2 accelerator has on the average 402.58 times faster classification time than the software implementation. EM-SP accelerator is on the average 1085.26 times faster than the software implementation, while EM-SP2 accelerator offers the largest improvement from all accelerators. On the average, it classifies instances 21642.28 times faster than the software implementation.

IV. SUMMARY AND DISCUSSIONS

In this paper four different hardware architectures for individual DT ensemble classifier member implementation have been proposed. Furthermore, several architectures for the hardware implementation of the most popular ensemble combination rules have also been proposed. Using these two components hardware accelerators of complete DT ensemble classifier can be built, with varying area/speed ratios enabling selection of the most appropriate implementation based on the current application requirements. All proposed architectures can be implemented using both ASIC and FPGA technology.

Results of the experiments, based on 29 datasets from standard UCI machine learning repository, indicate that the proposed hardware accelerators of DT ensemble classifiers implemented using FPGA technology, offer significant improvement in the average instance classification time when compared with the traditional software implementation.

REFERENCES

[1] P. Flach, "Machine Learning: The Art and Science of Algorithms that Make Sense of Data", *Cambridge University Press*, 2012.

[2] K. P. Murphy, "Machine Learning: A Probabilistic Perspective", *MIT Press*, 2012.

[3] L. Rokach and O. Maimon, "Data Mining with Decision Trees: Theory and Applications",*World Scientific Publishing*, 2008.

[4] L. Rokach and O. Maimon, "Top-down induction of decision trees - A survey", *IEEE Transactions on Systems, Man and Cybernetics*, vol. 35, 2005, pp. 476-487.

[5] S. Abe, "Support vector machines for pattern classification", *Springer,* 2010.

[6] S. Haykin, "Neural Networks and Learning Machines, 3rd Ed", *Prentice Hall*, 2008.

[7] I. H. Witten and E. Frank, "Data Mining: Practical machine learning tools and techniques, 3rd Ed.", *Morgan Kaufmann*, 2011.

[8] X. Wu and V. Kumar, "The Top Ten Algorithms in Data Mining", *Chapman and Hall*, 2009.

[9] C. Zhang and Y. Ma. "Ensemble Machine Learning: Methods and Applications", *Springer*, 2012.

[10] S. J. D. Prince, "Computer Vision: Models, Learning, and Inference",*Cambridge University Press*, 2012.

[11] A. Lesk, "Introduction to Bioinformatics, 4th Ed", *Oxford University Press,* 2014.

[12] B. Liu, "Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data, 2nd Ed", *Springer*, 2011.

[13] R. Bekkerman, M. Bilenko and J. Lanqford, "Scaling Up Machine Learning: Parallel and Distributed Approaches", *Cambridge University Press*, 2011.

[14] A. N. Choudhary, D. Honbo, P. Kumar, B. Ozisikyilmaz, S. Misra and G. Memik, "Accelerating data mining workloads: current approaches and future challenges in system architecture design", *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, 2011, pp. 41-54.

[15] R. Struharik, "Implementing Decision Trees in Hardware", *9th IEEE International Symposium on Intelligent Systems and Informatics*, 20-22 September 2012, pp. 41-46.

[16] S. Fareena, A. Dutta, J. Plusquellic, P. Ortiz and M. S. Pattichis, "Pipelined Decision Tree Classification Accelerator Implementation in FPGA (DT-CAIF)", *IEEE Transactions on Computers*, vol. 64, no. 1, 2015, pp. 280-285.

[17] M. Papadonikolakis and C. S. Bouganis, "Novel Cascade FPGA Accelerator for Support Vector Machines Classification", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 7, 2012, pp. 1040-1052.

[18] A. Davide, L. Carlino, A. Ghio and S. Ridella, "A FPGA core generator for embedded classification systems", *Journal of Circuits, Systems, and Computers*, vol. 20, no. 2, 2011, pp. 263-282.

[19] A. Savich, M. Moussa and S. Areibi, "A scalable pipelined architecture for real-time computation of MLP-BP neural networks", *Microprocessors and Microsystems*, vol. 36, no. 2, 2012, pp. 138-150.

[20] D. Vainbrand and R. Ginosar, "Scalable network-on-chip architecture for configurable neural networks", *Microprocessors and Microsystems*, vol. 35, no. 2, 2011, pp. 152-166.

[21] P. Bühlmann, "Bagging, boosting and ensemble methods", *Handbook of Computational Statistics*, *eds. T. Moris and J. E. Gentle*, *Springer*, 2012.

[22] M. Ozay, F. Vural, "Performance analysis of stacked generalization classifiers", *IEEE 16th Signal Processing, Communication and Application Conference*, 20-22 April 2008, pp. 1-4.

[23] R.A. Jacobs, M.I. Jordan, S.J. Nowlan, and G.E. Hinton, "Adaptive mixtures of local experts", *Neural Computation*, vol. 3, 1991, pp. 79-87.

[24] Y. S. Huang and C. Y. Suen, "The behavior-knowledge space method for combination of multiple classifiers", *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 15-17 June 1993, pp. 347-347.

[25] R. Struharik, "Implementing Decision Trees in Hardware", *9th IEEE International Symposium on Intelligent Systems and Informatics*, 20-22 September 2012, pp. 41-46.

[26] C. L. Blake, and C. J. Merz, "UCI Repository of Machine Learning Databases", Available online at: www.ics.uci.edu/~mlearn/ML-Repository.http, 2014.