A Review Of Boosted Machine Learning Techniques For Particle Identification At ALICE

JEDDA BOYLE, University of Cape Town

Machine learning techniques are fundamental to the automation of particle identification for ALICE. This literary review discusses the advantages that boosted machine learning algorithms have with respect to achieving more accurate and efficient particle identification. The review begins with an introduction to the principles of boosting and then provides a rigorous specification of a boosting algorithm. This is used as a framework to compare boosting with artificial neural networks - another common machine learning technique used for particle identification - and to discuss optimisations. We focus the optimisations section on techniques which improve classification speeds. This is because the scheduled upgrades to ALICE are going to increase the amount of data that needs to be classified. By the end of this review it will be clear to the reader that boosting has many advantages, which are pertinent to the ALICE experiment, and therefore should be considered as a part of the particle identification framework.

Not enough is known about the very early universe. Roughly a millionth of a second into the universe's existence the mixture of quarks and gluons, known as quark-gluon plasma, began to cool and form hadron particles - the fundamental building blocks of matter [Trafton. 2010]. Naturally, the study of quark-gluon plasma has become an imperative for physicists. However, quark-gluon plasma doesn't occur naturally so scientists have to make it by accelerating and colliding heavy ions - the nuclei of heavy particles - at nearly light speed [Trafton. 2010]. These experiments are being done at the Large Hadron Collider (LHC) in Europe and at the Relativistic Heavy Ion Collider (RHIC) in the United States.

A Large Ion Collider Experiment (ALICE) is the dedicated heavy ion experiment at the LHC. The Transition Radiation Detector (TRD) is a layer of the ALICE detector which monitors the temporal charges of particles, emitted from heavy ion collisions, as they pass through it [Wilk. 2010]. Amongst these particles are electrons, negatively charged elementary particles, and pions, short-lived particles made up of quarks and and anti-quarks. In this review we consider machine learning techniques used to classify electrons and pions passing through the TRD. Due to scheduled upgrades to ALICE, which will increase the number of detectable heavy ion collisions, we emphasise techniques used to reduce the computational cost of classifications.

The use of artificial networks is the standard method for particle identification at ALICE and has been shown to be successful [Wilk. 2010]. In this review we focus on boosting, an ensemble algorithm, which has been successfully implemented at other particle identification experiments such as MiniBooNE at Fermilabs [Roe et al. 2005]. Boosting has many advantages. It has a tendency to not over fit data [Freund and Schapire. 1999], it can efficiently utilise many explanatory variables [Roe et al. 2005], its classification is easily parallelised, and it can dynamically alter the amount of computation required for classification relative to the complexity of a particular instance [Sun and Zhou. 2013]. All of these features make it an accurate and computationally efficient classifier. During the course of this review the advantages of boosting will made clear.

Boosting falls into a family of machine learning algorithms called ensemble algorithms. Ensemble algorithms aim to construct a strong learner from a set of weak learners - by compiling the weak learners' classifications - using slightly altered versions of the training data [Dietterich. 2000]. A weak learner only needs to be slightly more accurate than random. This technique has been well tested and many authors have shown its effectiveness [Dietterich. 2000] [Bauer and Kohavi. 1999]. Implementations often use decision trees or support vector machines as weak learners but this can vary. The two main classes of ensemble algorithms are the AdaBoost family (boosting) and bootstrap aggregation (bagging).

Both bagging and boosting alter the training data between weak learners to emphasise certain features. This specialises the weak learners by making each individual more responsive to different features of the training data and ultimately improving the strong learner. Bagging achieves this by using bootstrapped samples of the training data for each weak learner and boosting annotates the training data for each weak learner with different weights [Dietterich. 2000].

Both bagging and boosting rely on the instability of the weak learners - in that they exploit the variance of the weak learners resulting from the small changes in the training data [Dietterich. 2000]. This should impact the choice of weak learner. Boosting is often more effective than bagging. This is because boosting is less reliant on the instability of weak learners, since it can use weights to make more emphatic changes to the training data [Dietterich. 2000].

AdaBoost was the first implementation of boosting which dealt with the practical problems of early attempts [Freund and Schapire. 1999]. It has become the template for the entire class of algorithms and is still a very effective [Roe et al. 2005].

The notation we use to formalise AdaBoost is going to be used throughout the review. This specification of AdaBoost is from [Freund and Schapire. 1999]. Let's consider a set X where each element is classified as either -1 or 1. The task of the machine learning algorithm is to construct a function $h : X \to \{-1, 1\}$ which correctly classifies every element of X. To achieve this we have our training data $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ where $x_i \in X$ and $y_i \in \{-1, 1\}$ for all $i \leq n$. We are going to construct h from a linear combination of T functions, $h_t : X \to \{-1, 1\}$ where $t \leq T$, called weak learners.

Every weak learner h_t has a weight $w_{t,i}$ associated with each training example (x_i, y_i). This weight is initialized to $\frac{1}{n}$ and updated during the training process by the following rule:

$$w_{t+1,i} = \frac{w_{t,i} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases}}{Z}$$

Z is a normalisation factor used to ensure that $\sum w_{t,i}$ remains equal to one. An important aspect of the above formula is that $w_{t+1,i}$ is inversely proportional to how accurately h_t classified (x_i, y_i). This means that $w_{t+1,i}$ is larger if the previous $h_1, h_2, ..., h_t$ weak learners struggled to classify x_i. This technique helps specialise the weak candidates by providing annotated training data which has had the difficult examples emphasized with larger weights.

We define α_t, the confidence rating of h_t, as:

$$\alpha_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$$

ϵ_t is the error of h_t and is defined as:

$$\epsilon_t = \sum_{h(x_i)\neq y_i}^{i\leq n} w_{t,i}$$

This means that the error is a measure of how accurately difficult training examples are classified. It is also important to point out that α_t and ϵ_t are inversely proportional. Therefore, α_t can be seen as a measure of how valid and novel the classification of h_t is. This leads to a natural definition of h, which is:

$$h(x) = \text{sign}\left\{\sum_{i=0}^{i\leq T} \alpha_i h_i(x)\right\}$$

The reasons for the specific definitions of $w_{t,i}$, α_t and h_t, apart from their proportional relationships, are mathematical. A full explanation is beyond the scope of this paper. The mathematics reduces many of the computationally expensive procedures such as calculating Z.

Now that we have an idea of how boosting works let's consider how it can be used for particle identification. As mentioned earlier, artificial neural networks are the standard technique for particle identification for ALICE. However, boosting has been used for particle identification at the MiniBooNE experiment [Roe et al. 2005] [Yang et al. 2005] [Yang et al. 2007] and therefore provides us with a good framework to investigate how boosting may perform for ALICE. At the MiniBooNE experiment decision trees were used as the weak learners [Yang et al. 2005]. This construction is known as a boosted decision tree.

The tests between artificial neural networks and boosted decision trees at MiniBooNE were conclusive. Boosted decision trees improved performance on all tests and in some cases it improved classification accuracy by 80 percent [Roe et al. 2005]. Boosted decision trees could have made even greater gains had the experiments not required the artificial neural networks and boosted decision trees to use the same number of explanatory variables. This is an issue because boosted decision trees, unlike artificial neural networks, can efficiently utilise many explanatory variables even if they have weak discriminant power [Roe et al. 2005]. This is not true for artificial neural networks because many explanatory variables make it much harder to optimise the weights [Roe et al. 2005]. Artificial neural networks performed optimally with 30 explanatory variables while the boosted decision tree could have performed significantly better with more explanatory variables [Roe et al. 2005]. In the context of ALICE however, the computational requirements may restrain the number of explanatory variables and unfortunately none of the experiments test the performance differences between artificial neural networks and boosted decision trees for a small number of explanatory variables.

Another advantage boosted decision trees have over artificial neural networks is that they are more stable. To compare the stability of the two techniques experiments were run where the training data was altered. For example the data was smeared

using the following formula:

$$x_i = x_i \times (1 + sf \times r)$$

where sf is a smear factor which was varied between 0.01 and 0.1 and r was a random number with Gaussian distribution [Yang et al. 2007]. Both artificial neural networks and boosted decision trees turned out to be robust [Yang et al. 2007]. However, boosted decision trees performed slightly better than artificial neural networks. This difference in classification accuracy becomes clearer for larger smear factors [Yang et al. 2007].

Boosted decision trees also require less optimisation than artificial neural networks because there are fewer changeable parameters [Roe et al. 2005].

Naturally the next consideration is what implementation of boosting is optimal. Fortunately, various boosted decision tree implementations have been tested at MiniBooNE. The practical experiments compared AdaBoost, ϵ-Boost ,ϵ-LogitBoost and ϵ-HingeBoost.

The structure of ϵ-Boost is identical to AdaBoost but it has a different weight update and weak learner aggregation formula.

The tests showed that ϵ-LogitBoost and ϵ-HingeBoost were inferior [Yang et al. 2005]. The major difference in performance between AdaBoost and ϵ-Boost, which is worth emphasising in our context, is the fact that for small numbers of weak learners - around 100 - AdaBoost is more accurate [Roe et al. 2005]. For $5,000$ or more weak learners ϵ-Boost starts to perform better than AdaBoost [Roe et al. 2005]. It would appear that for ALICE, where fewer weak learners would be better, AdaBoost would outperform ϵ-Boost.

The results from the MiniBooNE experiment are clear. Using various different metrics the literature has shown that boosted decision trees perform well compared to artificial neural networks at identifying particles. However, it is important to point out that the MiniBooNE experiment is in some respects quite different to ALICE. The particles being identified are not pions or electrons and the receptor at MiniBooNE is different to ALICE's TRD layer.

For the rest of this review we look at techniques for increasing the speed of classifications. This is generally done by trying to reduce the number of weak learners. This technique is called pruning. Given that the upgrades to ALICE will increase the number of particle collisions detected and that on-line classification is desired, being able to quickly classify particles is essential.

A natural question to ask is whether all the weak learners are improving the classification. If not, could some be pruned? In general, increasing the number of weak learners improves classification accuracy [Roe et al. 2005] but this is in aggregate. It is possible that cherry picking a subset of the weak learners could improve classification accuracy and reduce classification speed. An evolutionary approach, called evolutionary pruning, is used to answer this question. It is based upon the assumption that there is redundancy in weak learners resulting from dependencies[Jang and Kim. 2008].

Evolutionary pruning involves two stages. The first stage attempts to calculate

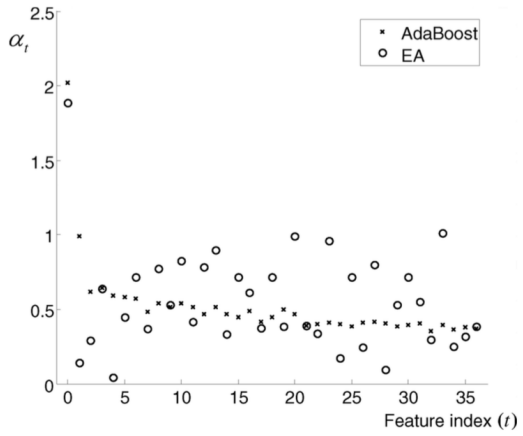

Fig. 1. The distribution of AdaBoost weights and the results of evolutionary pruning [Jang and Kim. 2008]
.

the importance of each weak learner in the strong learner. In AdaBoost the weights $\alpha_0, \alpha_2, ..., \alpha_T$ are meant to be indicators of a weak learner's importance. However, due to AdaBoost's structure it is unable to search the entire weight space. Greater importance is put on weak learners in early stages, with smaller t [Jang and Kim. 2008]. Intuitively this is because they classify the 'low hanging fruit' and leave the more difficult classifications for weak learners at a later stage. This means that the algorithm doesn't return the ideal weight distribution [Jang and Kim. 2008]. The first stage of the evolutionary pruning aims to redistribute the weights more efficiently.

The algorithm works as follows. Pick p and q which are acceptable detection and false positive rates. Get a sequence $\alpha_0, \alpha_2, ..., \alpha_T$ of weights from AdaBoost which achieve p and q. Use this sequence to construct a population of chromosomes. Each chromosome is a vector of T weights $\hat{\alpha_0}, \hat{\alpha_1}, ..., \hat{\alpha_T}$ chosen randomly with respect to the condition that $\hat{\alpha_i} \leq \max(\alpha_0, \alpha_2, ..., \alpha_T)$ for all $i \leq T$. The original sequence $\alpha_0, \alpha_2, ..., \alpha_T$ is not part of this population because it restricts the exploration of evolution [Jang and Kim. 2008].

The evolutionary mechanism uses a q-tournament selection method and the following fitness function:

$$\text{Fitness} = \sum_{i=0}^{i \leq n} c(x_i) - \begin{cases} p - \hat{p} & \text{if } \hat{p} \leq p \\ 0 & \text{otherwise} \end{cases} - \begin{cases} \hat{q} - q & \text{if } \hat{q} \geq q \\ 0 & \text{otherwise} \end{cases}$$

The variables $\hat{p}$ and $\hat{q}$ are the obtained detection and false positive rate from the chromosome. The function $c(x_i)$ equals 1 if x_i was correctly classified and 0 otherwise. Basically the fitness is increased by 1 for every correct classification and reduced by 1 if the detection rate decreases or false positive rate increases [Jang and Kim. 2008].

The resulting weights of this evolutionary process can be seen in figure 1 which displays the weights after the first stage of evolutionary pruning compared to the original AdaBoost weights. Notice how the AdaBoost weights are more uniform and in a downward pattern. The purpose of this process was to get a better indication of which weak learners were important so that in the second stage weak learners could

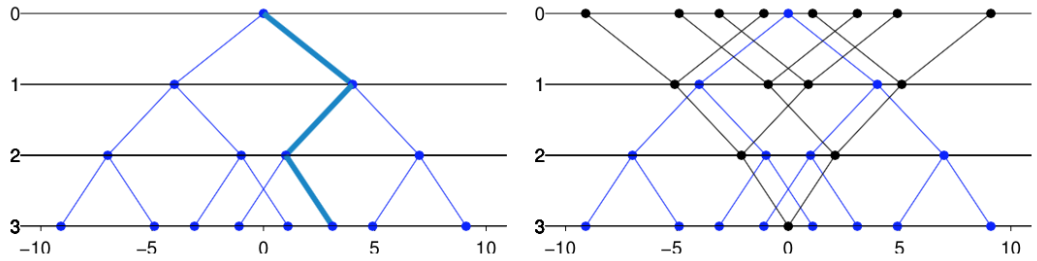

Fig. 2. A boosted strong learner represented as a tree on the left and the same tree flipped and mapped on top of itself on the right [Sun and Zhou. 2013].

be pruned [Jang and Kim. 2008].

The second stage is an iterative process which prunes the weak learner with smallest $\hat{\alpha}_i$ until the detection rate drops below p or the false positive rate exceeds q [Jang and Kim. 2008]. This wouldn't have worked with the original weights because it would have an inaccurate bias towards weak learners with large t values. It would essentially just trim the weak learners off the end.

Evolutionary pruning has two major advantages. Firstly it can reduce the number of weak learners and therefore increase classification speeds [Jang and Kim. 2008]. However, this is not guaranteed. It is possible after the first stage of re-weighting the weak learners that none get pruned in the second stage. The second major advantage is that the accuracy of the strong learner can be increased. Both [Jang and Kim. 2008] and [Zhou et al. 2002] have found that evolutionary pruning can improve accuracy.

Evolutionary pruning is useful but boosting lends itself to another kind of pruning. Dynamic pruning can change the number of weak learners depending on the complexity of the instance being classified.

For the next few paragraphs we will discuss different forms of dynamic pruning. For simplicity let each weak learner be a pseudo binary function $h_t : X \rightarrow \{\alpha_t, \beta_t\}$ and without loss of generality assume $\alpha_t < \beta_t$. Given that each weak learner has two possible values it is useful to consider the strong learner as a binary tree. Each level t of the tree represents a weak learner h_t and every node has an associated value. The root has value 0. Every non-leaf node on level t with value v has exactly 2 children. The left child has value $v + \alpha_t$ and the right child has value $v + \beta_t$. The tree is complete and every possible value for h is represented by a leaf. The left tree of figure 2 is an example of such a tree. The bold line is an example of a path an input x may take through the graph to a leaf which has value $h(x)$. Given this representation of a tree one can consider that improving the classification speed of h is analogous to pruning the tree.

FastExit is one such algorithm. It calculates if the remaining weak learners have sufficient sway to change the sign of the boosted algorithm's output and if not it terminates [Kim at al. 2012]. For example, consider a strong learner h which consists of a sequence of weak learners $h_1, h_2, ...h_T$ where $h_t : X \rightarrow [\alpha_t = -1, \beta_t = 1]$ for all $t \leq T$. Then, if at step t the equation $|\sum_{i=0}^{i \leq t} h_i(x)|$ is greater than the threshold value

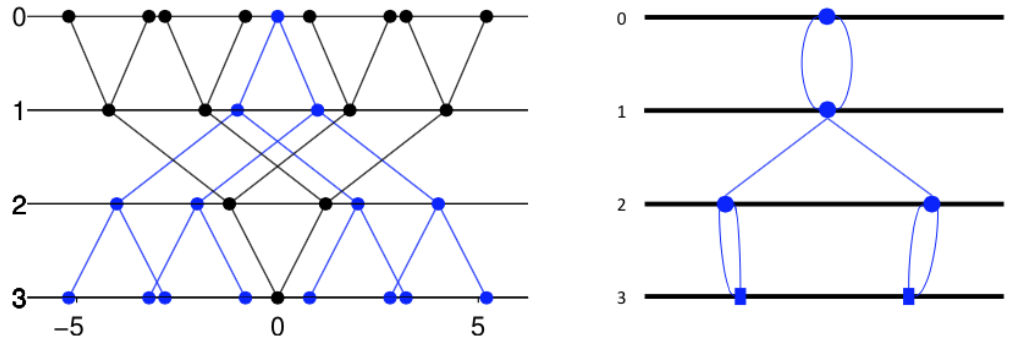

Fig. 3. A figure showing how a tree representation of boosted strong learner, such as on the left, can be converted to a Binary Decision Diagram, such as on the right [Sun and Zhou. 2013].

$n - t$ the remaining $n - t$ weak learners don't need to be calculated. This is because the sign of h cannot be changed. This algorithm can naturally be extended to weak learners with non-uniform ranges by altering the equation for the threshold value. In a binary tree representation such as 2 FastExit would return at any node whose entire sub tree falls into exclusively $\mathbb{R}^+$ or $\mathbb{R}^-$. FastExit has two major advantages. It doesn't increase classification error and the threshold values for each t can be calculated once and saved. This means that there is almost no computational cost and absolutely no accuracy cost, so implementing it is essentially free. However, its major downfall is that it's relatively unintelligent and therefore can only make major reductions to the number of weak learners executed for a very small subset of the input space.

FastExit is based upon the observation that at a certain point the remaining weak classifiers cannot affect the overall classification. Direct Backwards Pruning uses the additional observation that many of the possible paths in a tree are never traversed by a training example [Zhang and Viola. 2007]. Therefore, Direct Backwards Pruning terminates at a node if all paths in its sub tree which have been traversed by a training example fall into exclusively $\mathbb{R}^+$ or $\mathbb{R}^-$ [Zhang and Viola. 2007]. Given that the number of training examples is generally much less than 2^T - the size of the tree - Direct Backwards Pruning can massively improve classification speeds. However, unlike FastExit it can increase the classification error. The magnitude of this negative effect is dependent on the training data. It won't work if the training set is too small, has incorrectly labelled instances or isn't representative for some other reason. The increase in classification speed needs to be weighed against the loss in accuracy.

Another more advanced pruning technique called a Threshold Binary Decision Diagram uses a tree $\top$ of a strong learner to construct a Binary Decision Diagram [Sun and Zhou. 2013]. The technique involves flipping $\top$ to a tree we shall label $\bot$ and mapping $\top$ on top of $\bot$ [Sun and Zhou. 2013]. An example can be seen on the right of figure 2. Notice that the nodes of $\bot$ creates a set of intervals I on each level. If two nodes of $\top$ fall into the same interval of I then we merge them into one node [Sun and Zhou. 2013]. The resulting tree may have nodes whose left and right children have become the same node. In this case the node can be removed and the resulting tree is a binary decision diagram [Sun and Zhou. 2013]. Consider figure 3 as an example. The original tree which had three layers of classification has been reduced to a single weak classifier - the only node with more than one child. Using Threshold Binary

Diagram will always prune more nodes than FastExit and one would expect less than Direct Backwards Pruning.

In the previous section we considered the techniques for reducing the number of weak learners involved in a classification. We shall now discuss the ordering of the weak learners. Classification speeds can be very sensitive to ordering [Sun and Zhou. 2013] and therefore it should be considered given the computational requirements of particle identification for ALICE. Consider the following simple explanatory example. Let h be a strong learner consisting of $h_1 : X \rightarrow \{-3, 3\}$, $h_2 : X \rightarrow \{-1, 1\}$ and $h_3 : X \rightarrow \{-1, 1\}$. Clearly if we use FastExit the ordering h_1, h_2, h_3 will always return after one execution of a weak learner while the ordering h_3, h_2, h_1 will always involve all three weak learners.

The first sorting technique we are going to discuss follows naturally from the explanatory example in the previous paragraph. Simply annotate each weak learner h_t with a weight equal to $|\alpha_t| + |\beta_t|$ and then order the set of weak learners in decreasing order [Kim at al. 2012]. This technique is simple and as the example in the previous paragraph shows it can make major improvements. However, it is limited by the fact that for more complex pruning techniques such as Direct Backwards Pruning and Threshold Binary Decision Diagrams there is no guarantee that classification speeds will be improved.

An obvious algorithm to find the optimal ordering would be an exhaustive search. However, for a strong learner h consisting of T weak learners this would involve checking $T!$ options. This is clearly not feasible for anything other than very small T and therefore an algorithm called sifting is suggested as a compromise. Initialise the sequence of weak learners into a random order. For every weak learner swap it with the learner to its left until it has risen to the top. Do the same with its right neighbours until it has sunk to the bottom. After each swap measure the efficiency of the current ordering and record which permutation was best. At the end of this iteration place the weak learner in the position where it performed best. Repeat this process for each weak learner m times [Sun and Zhou. 2013]. The time complexity of this algorithm is $O(mT^2)$ [Sun and Zhou. 2013]. The difference in efficiency between the returned permutation and the actual best permutation is reduced by increasing m.

The primary issue with this algorithm is its computational cost. $O(mT^2)$ is reasonable but the constant factors are large primarily due to the fact that a permutation's efficiency must be calculated on every iteration of the inner loop. The exact method for determining the efficiency of a permutation is not strictly defined. However, common techniques involve calculating the expected path length. The exact details of this calculation can be found in [Sun and Zhou. 2013]. The technique involves classifying all training instances and therefore is costly. Despite the computational cost it is still a massive improvement from the brute-force method and does obtain better solutions than simpler sorting implementations such as the one mentioned above. Also, it only needs to be run once and the ordering can be saved for future classifications.

Throughout this paper we have focused on the speed of classification rather than training speeds. This is due to the desired on-line classification for ALICE. However, there is literature on increasing training speeds which reveals two classes of optimisation generally used. The first is to use subsets of the training data and the second is parallelism. Despite the apparent sequential nature of training a boosting algorithm,

parallel implementations have been suggested [Lozano and Rangel. 2005][Merler et al. 2006].

This review introduced the concept of boosting to the reader. Boosting as a possible particle identification technique was compared with artificial neural networks. It was shown that boosted decision trees performed very well compared to artificial neural networks. Secondly, we reviewed how boosted algorithms can be pruned to improve performance and accuracy - two important requirements for ALICE. The review has made it clear that boosting is an effective technique for particle identification and appropriate for ALICE. Therefore, boosting should be considered as a possible solution to the particle identification problem at ALICE.

REFERENCES

Ron Appel, Thomas Fuchs, Piotr Dollar and Pietro Perona. Quickly Boosting Decision Trees - Pruning Underachieving Features Early. in *The Proceeding of the 30th International Conference on Machine Learning*, (Atlanta, 2013) JMLR.

Eric Bauer and Ron Kohavi. 1999. An Empirical Comparison of Voting Classifications Algorithms: Bagging, Boosting, and Variants. *Machine Learning*, Vol. 36, 105-139.

R Brun, P Buncic, F Carminati, A Morsch, F Rademakers and K Safarik. 2003. Computing in ALICE. *Nuclear Instruments & Methods In Physics Research*. Vol. 502, 339-346.

Thomas Dietterich. 2000. An Experimental Comparison Of Three Methods for Constructing Ensemble Decision Trees: Bagging, Boosting, and Randomaization. *Machine Learning*, Vol. 40, 139-157.

Yoav Freund and Robert Schapire. 1999. A Short Introduction to Boosting. *Journal of Japanese Society for Artificial Intelligence*, Vol. 14, 771-780

Tae-Kyun Kim, Ignas Budvytis and Roberto Cipolla. 2012. Making a Shallow Network Deep: Conversion of a Boosting Classifier into a Decision Tree By Boolean Optimisation. *International Journal of Computer Vision*, Vol. 100, 203-215.

Fernando Lozano and Pedro Rangel. Algorithms For Parallel Boosting in *Proceedings of the Fourth International Conference on Machine Learning and Applications*, (2005), IEEE.

Stefano Merler, Bruno Caprile, and Cesare Furlanello. 2006. Parallelizing AdaBoost by Weights Dynamics. *Computational Statistics & Data Analysis*, Vol. 51, 2487-2498.

Byron Roe, Hai-Jun Yang, Ji Zhu, Yong Liu and Ion Stancu. 2005. Boosted Decision Trees as an Alternative to Artificial Neural Networks for Particle Identification. *Nuclear Instruments & Methods In Physics Research*, Vol. 543, 577-584.

Byron Roe, Hai-Jun Yang and Ji Zhu. 2014. Improved Boosting Algorithm Using Confidence-Rated Predictions. *European Physical Journal*.

Robert Schapire and Yoram Singer. 1999. Improved Boosing Algorithm Using COnfidence-Rated Predictions. *Machine Learning*, Vol. 37, 297-336.

Peng Sun and Jie Zhou. 2013. Saving Evaluation Time for the Decision Function in Boosting Representation and Reordering Base Learner in *Proceeding of the 30th International Conference on Machine Learning*, (Atlanta, 2013) JMLR.

Anne Trafton. 2010. Explained: Quark-Gluon Plasma *MIT News Office*, Retrieved April 28, 2015 from http://newsoffice.mit.edu/2010/exp-quark-gluon-0609

Hai-Jung Yang, Byron Roe and Ji Zhu. 2005. Studies of Boosted Decision Trees for MiniBooNE Particle Identification. *Nuclear Instruments & Methods In Physics Research*, Vol. 555, 370-385.

Hai-Jun Yang, Byron Roe and Ji Zhu. 2007. Studies of Stability and Robustness for Artificial Neural Networks and Boosted Decision Trees *Nuclear Instruments & Methods In Physics Research*, Vol. 574, 342-349.

Ju-Su Jang and Jong-Hwan Kim, 2008. Fast and Robust Face Detection Using Evolutionary Pruning. *IEEE Transactions on Evolutionary Computation*.

Liu Yong and Ion Stancu. 2007. Cascade Training Technique for Particle Identification. *Nuclear Instruments & Methods In Physics Research*, 315-320.

Alexander Wilk. 2010. Particle Identification Using Artificial Neural Networks with the Alice Transition Radiation Detector. Ph.D. Dissertation.

Cha Zhang and Paul Viola. 2007. Multiple-Instance Pruning For Learning Efficient Cascade Detectors. *Microsoft Research*.

Zhi-Hua Zhou, Jianxin Wu and Wei Tang. 2002. Ensemble Neural Networks: Many Could Be Better Than All. *Artificial Intelligence*, Vol. 137, 239-263.