Boosted decision trees as an alternative to artificial neural networks for particle identification

Byron P. Roe[a], Hai-Jun Yang[a,*], Ji Zhu[b], Yong Liu[c], Ion Stancu[c], Gordon McGregor[d]

[a]Department of Physics, University of Michigan, Ann Arbor, MI 48109, USA
[b]Department of Statistics, University of Michigan, Ann Arbor, MI 48109, USA
[c]Department of Physics and Astronomy, University of Alabama, Tuscaloosa, AL 35487, USA
[d]Los Alamos National Laboratory, Los Alamos, NM 87545, USA

Abstract

The efficacy of particle identification is compared using artificial neutral networks and boosted decision trees. The comparison is performed in the context of the MiniBooNE, an experiment at Fermilab searching for neutrino oscillations. Based on studies of Monte Carlo samples of simulated data, particle identification with boosting algorithms has better performance than that with artificial neural networks for the MiniBooNE experiment. Although the tests in this paper were for one experiment, it is expected that boosting algorithms will find wide application in physics.
© 2005 Elsevier B.V. All rights reserved.

1. Introduction

The artificial neural network (ANN) technique has been widely used in data analysis of High Energy Physics experiments in the last decade. The use of the ANN technique usually gives better results than the traditional simple-cut techniques. In this paper, another data classification technique, *boosting*, is introduced for data analysis in the MiniBooNE experiment [1] at Fermi National Accelerator Laboratory. The MiniBooNE experiment is designed to confirm or refute the evidence for $\nu_\mu \to \nu_e$ oscillations at $\Delta m^2 \simeq 1\,\mathrm{eV}^2/c^4$ found by the LSND experiment [2]. It is a crucial experiment which will imply new physics beyond

*Corresponding author.
  E-mail address: yhj@umich.edu (Hai-Jun Yang).

the standard model if the LSND signal is confirmed. Based on our studies, particle identification (PID) with the boosting algorithm is 20–80% better than that with our standard ANN PID technique. The boosting performance relative to that of ANN depends on the Monte Carlo samples and PID variables. Although the boosting algorithm was tested in only one experiment, it's anticipated to have wide application in physics, especially in data analysis of particle physics experiments for signal and background events separation.

The boosting algorithm is one of the most powerful learning techniques introduced during the past decade. The boosting algorithm is a procedure that combines many "weak" classifiers to achieve a final powerful classifier. Boosting can be applied to any classification method. In this paper, it is applied to decision trees. Two boosting algorithms, AdaBoost [3] and ε-Boost [4], are considered. A brief description of boosting algorithms is given in the next section. Our results are presented in Section 3, while we summarize our conclusions in Section 4.

2. Brief description of boosting

2.1. Decision tree

Suppose one is trying to divide events into signal and background and suppose Monte Carlo samples of each are available. Divide each Monte Carlo sample into two parts. The first part, the training sample, will be used to train the decision tree, and the second part, the test sample, to test the final classifier after training.

For each event, suppose there are a number of PID variables useful for distinguishing between signal and background. Firstly, for each PID variable, order the events by the value of the variable. Then pick variable one and for each event value see what happens if the training sample is split into two parts, left and right, depending on the value of that variable. Pick the splitting value which gives the best separation into one side having mostly signal and the other mostly background. Then repeat this for each variable in turn. Select the variable and splitting value which gives

the best separation. Initially there was a sample of events at a "node". Now there are two samples called "branches". For each branch, repeat the process, i.e., again try each value of each variable for the events within that branch to find the best variable and splitting point for that branch. One keeps splitting until a given number of final branches, called leaves, are obtained, or until each leaf is pure signal or pure background, or has too few events to continue. This description is a little oversimplified. In fact at each stage one picks as the next branch to split, the branch which will give the best increase in the quality of the separation. A schematic of a decision tree is shown in Fig. 1, in which 3 variables are used for signal/background separation: event hit multiplicity, energy, and reconstructed radial position.

What criterion is used to define the quality of separation between signal and background in the split? Imagine the events are weighted with each event having weight W_i. Define the purity of the sample in a branch by
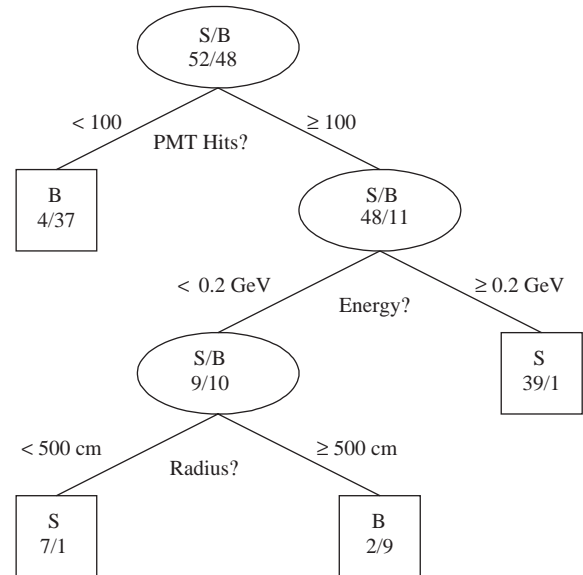
$$P = \frac{\sum_s W_s}{\sum_s W_s + \sum_b W_b}$$

Fig. 1. Schematic of a decision tree. S for signal, B for background. Terminal nodes (called leaves) are shown in boxes. If signal events are dominant in one leaf, then this leaf is signal leaf; otherwise, background leaf.

where $\sum_s$ is the sum over signal events and $\sum_b$ is the sum over background events. Note that $P(1-P)$ is 0 if the sample is pure signal or pure background. For a given branch let

$$\text{Gini} = \left(\sum_{i=1}^{n} W_i\right) P(1-P)$$

where n is the number of events on that branch. The criterion chosen is to minimize

$$\text{Gini}_{\text{left son}} + \text{Gini}_{\text{right son}}.$$

To determine the increase in quality when a node is split into two branches, one maximizes

$$\text{Criterion} = \text{Gini}_{\text{father}} - \text{Gini}_{\text{left son}} - \text{Gini}_{\text{right son}}.$$

At the end, if a leaf has purity greater than $\frac{1}{2}$ (or whatever is set), then it is called a signal leaf and if the purity is less than $\frac{1}{2}$, it is a background leaf. Events are classified signal if they land on a signal leaf and background if they land on a background leaf. The resulting tree is a *decision tree*.

Decision trees have been available for some time [5]. They are known to be powerful but unstable, i.e., a small change in the training sample can give a large change in the tree and the results.

There are three major measures of node impurity used in practice: misclassification error, the gini index and the cross-entropy. If we define p as the proportion of the signal in a node, then the three measures are: $1 - \max(p, 1-p)$ for the misclassification error, $2p(1-p)$ for the gini index and $-p\log(p) - (1-p)\log(1-p)$ for the cross-entropy. The three measures are similar, but the gini index and the cross-entropy are differentiable, and hence more amenable to numerical optimization. In addition, the gini index and the cross-entropy are more sensitive to change in the node probabilities than the misclassification error. The gini index and the cross-entropy are similar.

2.2. Boosting

Within the last few years a great improvement has been made [6–8]. Start with unweighted events and build a tree as above. If a training event is misclassified, i.e., a signal event lands on a background leaf or a background event lands on a signal leaf, then the weight of that event is increased (boosted).

A second tree is built using the new weights, no longer equal. Again misclassified events have their weights boosted and the procedure is repeated. Typically, one may build 1000 or 2000 trees this way.

A score is now assigned to an event as follows. The event is followed through each tree in turn. If it lands on a signal leaf it is given a score of 1 and if it lands on a background leaf it is given a score of -1. The renormalized sum of all the scores, possibly weighted, is the final score of the event. High scores mean the event is most likely signal and low scores that it is most likely background. By choosing a particular value of the score on which to cut, one can select a desired fraction of the signal or a desired ratio of signal to background. For those familiar with ANNs, the use of this score is the same as the use of the ANN value for a given event. For the MiniBooNE experiment, boosting has been found to be superior to ANNs. Statisticians and computer scientists have found that this method of classification is very efficient and robust. Furthermore, the amount of tuning needed is rather modest compared with ANNs. It works well with many PID variables. If one makes a monotonic transformation of a variable, so that if $x_1 > x_2$ then $f(x_1) > f(x_2)$, the boosting method gives exactly the same results. It depends only on the ordering according to the variable, not on the value of the variable.

In articles on boosting within the statistics and computer science communities, it is often recommended that short trees with eight leaves or so be used. For the MiniBooNE Monte Carlo samples it was found that large trees with 45 leaves worked significantly better.

2.3. Some boosting algorithms

If there are N total events in the sample, the weight of each event is initially taken as $1/N$. Suppose that there are N_{tree} trees and m is the index of an individual tree. Let

- x_i = the set of PID variables for the ith event.

- $y_i = 1$ if the ith event is a signal event and $y_i = -1$ if the event is a background event.
- $w_i =$ the weight of the ith event.
- $T_m(x_i) = 1$ if the set of variables for the ith event lands that event on a signal leaf and $T_m(x_i) = -1$ if the set of variables for that event lands it on a background leaf.
- $I(y_i \neq T_m(x_i)) = 1$ if $y_i \neq T_m(x_i)$ and 0 if $y_i = T_m(x_i)$.

There are at least two commonly used methods for boosting the weights of the misclassified events in the training sample.

The first boosting method is called AdaBoost [3]. Define for the mth tree:

$$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq T_m(x_i))}{\sum_{i=1}^{N} w_i},$$

$$\alpha_m = \beta \times \ln((1 - \text{err}_m)/\text{err}_m).$$

$\beta = 1$ is the value used in the standard AdaBoost method. For the MiniBooNE Monte Carlo samples, $\beta = 0.5$ has been found to give better results. Change the weight of each event i, $i = 1, \dots, N$:

$$w_i \rightarrow w_i \times e^{\alpha_m I(y_i \neq T_m(x_i))}.$$

Each classifier T_m is required to be better than random guessing with respect to the weighted distribution upon which the classifier is trained. Thus, err_m is required to be less than 0.5, since, otherwise, the weights would be updated in the wrong direction. Next, renormalize the weights, $w_i \rightarrow w_i/\sum_{i=1}^{N} w_i$. The score for a given event is

$$T(x) = \sum_{m=1}^{N_{\text{tree}}} \alpha_m T_m(x)$$

which is just the weighted sum of the scores over the individual trees, see Fig. 2.

The second boosting method is called ε-Boost [4], or sometimes "shrinkage". After the mth tree, change the weight of each event i, $i = 1, \dots, N$:

$$w_i \rightarrow w_i e^{2\varepsilon I(y_i \neq T_m(x_i))}$$

where ε is a constant of the order of 0.01. Renormalize the weights, $w_i \rightarrow w_i/\sum_{i=1}^{N} w_i$. The
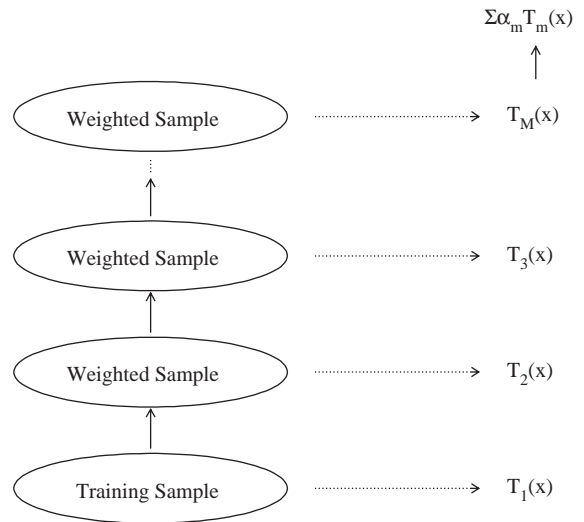

Fig. 2. Schematic of a boosting procedure.

score for a given event is

$$T(x) = \sum_{m=1}^{N_{\text{tree}}} \varepsilon T_m(x)$$

which is the renormalized, but unweighted, sum of the scores over individual trees.

The AdaBoost and ε-Boost algorithms used in this paper try to minimize the expectation value: $E(e^{-yF(x)})$, where $y = 1$ for signal, $y = -1$ for background, $F(x) = \sum_{i=1}^{N_{\text{trees}}} f_i(x)$, where the classifier $f_i(x) = 1$ if an event lands on signal leaf, and $f_i(x) = -1$ if an event lands on background leaf. This minimization is closely related to minimizing the binomial log-likelihood [4]. It can be shown that $E(e^{-yF(x)})$ is minimized at

$$F(x) = \frac{1}{2} \ln \frac{P(y = 1|x)}{P(y = -1|x)} = \frac{1}{2} \ln \frac{p(x)}{1 - p(x)}.$$

Let $y^* = (y + 1)/2$. It is then easy to show that

$$e^{-yF(x)} = \frac{|y^* - p(x)|}{\sqrt{p(x)(1 - p(x))}}.$$

The right-hand side is known as the χ statistic. χ^2 is a quadrative approximation to the log-likelihood, so χ can be considered a gentler alternative. It turns out that fitting using χ is monotone and smooth; the criteria will continually drive the estimates towards purer solutions. An ANN tries

to minimize the squared-error $E(y - F(x))^2$, where $y = 1$ for signal events, $y = 0$ for background events, and $F(x)$ is the network prediction for training events.

3. Results

For the $\nu_\mu \to \nu_e$ oscillation search in the MiniBooNE experiment [1], the main backgrounds come from intrinsic ν_e contamination in the beam, mis-identified ν_μ quasi-elastic scattering and mis-identified neutral current π^0 production. Since intrinsic ν_e events are real ν_e events, the PID variables cannot distinguish them from oscillation ν_e events. This report concentrates on separating the non-ν_e events from the ν_e events. Good sensitivity for the ν_e appearance search requires low background contamination from all kinds of backgrounds. Here, the ANN and the two boosting algorithms are used to separate ν_e charged current quasi-elastic (CCQE) events from non-ν_e background events.

500 000 Monte Carlo ν_μ events distributed among the many possible final states and 200 000 intrinsic ν_e CCQE events were fed into the reconstruction package R-fitter [9]. Among these events, 88 233 intrinsic ν_e CCQE and 162 657 background events passed reconstruction and pre-selection cuts.

The signature of each event is given by 52 variables for the R-fitter. All variables are used in the boosting algorithms for training and testing. It is a challenge to have agreement between data and Monte Carlo for all of the PID variables and for the boosting outputs. The MiniBooNE Collaboration is devoting considerable effort to achieve it. Monte Carlo samples using 18 different parameter sets have been generated and run through the same reconstruction programs. The results for both the PID variables and the boosting outputs are consistent. When the present Monte Carlo is compared with the real data samples, the shapes of the various PID variables and the boosting outputs match well. Since the reconstruction and PID algorithms are still undergoing continuous modifications, relative results rather than absolute percentages are presented in the following plots.

For the AdaBoost algorithm, the parameter $\beta = 0.5$, the number of leaves $N_{leaves} = 45$ and the number of tree iterations $N_{tree} = 1000$ were used. The relative ratio (defined as the number of background events kept divided by the number kept for 50% intrinsic ν_e selection efficiency and $N_{tree} = 1000$) as a function of ν_e selection efficiency for various tree iterations is shown in the top plot of Fig. 3 and the AdaBoost output distributions are shown in the bottom plot. 20 000 intrinsic ν_e CCQE signal and 30 000 background events were used for training, 68 233 ν_e and 132 657 background events were used for testing. All results shown in the paper are for testing samples.

In order to quantify the performance of the boosting algorithm, the AdaBoost results for a particular set of PID variables were compared with ANN results. The results, compared as a function of the intrinsic ν_e CCQE selection efficiency, are shown in Fig. 4. For the intrinsic ν_e signal efficiency ranging from 40% to 60%, the performances of AdaBoost were improved by a
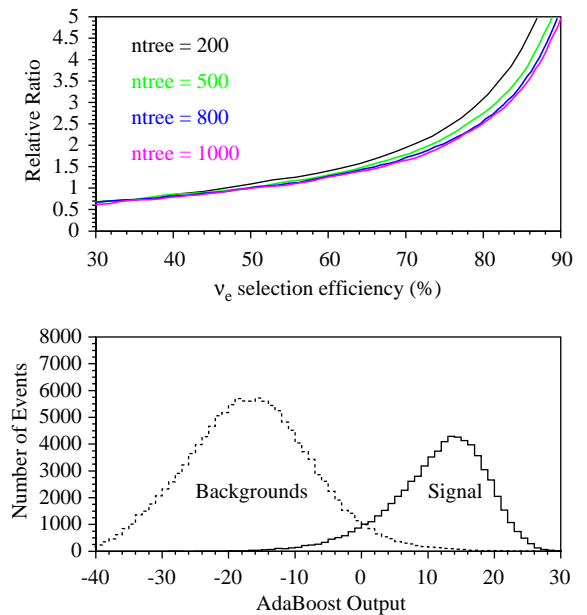

Fig. 3. Top: the number of background events kept divided by the number kept for 50% intrinsic ν_e selection efficiency and $N_{tree} = 1000$ versus the intrinsic ν_e CCQE selection efficiency. Bottom: AdaBoost output, all kinds of backgrounds are combined for the boosting training.
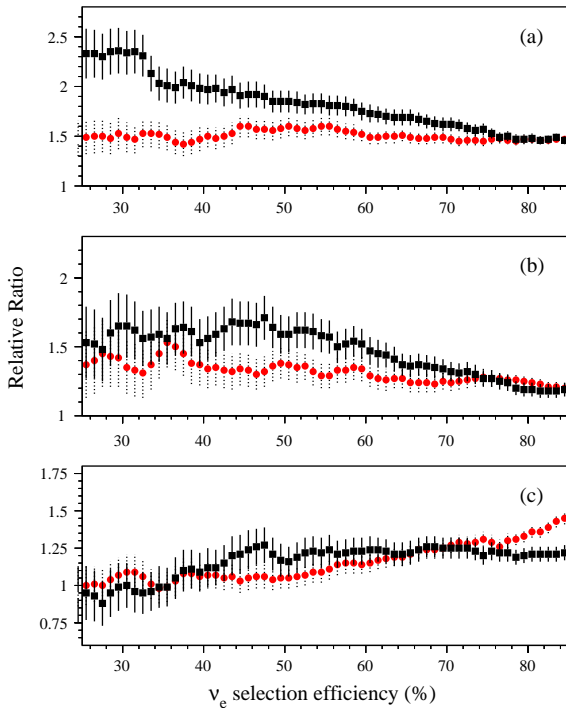
Fig. 4. Comparison of ANN and AdaBoost performance for test samples. Relative ratio (defined as the number of background events kept for ANN divided by the events kept for AdaBoost) versus the intrinsic ν_e CCQE selection efficiency. (a) All kinds of backgrounds are combined for the training against the signal. (b) Trained by signal and neutral current π^0 background. (c) Relative ratio is re-defined as the number of background events kept for AdaBoost with 21(red)/22(black) training variables divided by that for AdaBoost with 52 training variables. All error bars shown in the figures are for Monte Carlo statistical errors only.

factor of approximately 1.5 and 1.8 over the ANN if trained by the signal and all kinds of backgrounds with 21 (red dots) and 52 (black boxes) input variables respectively, shown in Fig. 4(a). If AdaBoost and ANN were trained by the signal and neutral current π^0 background, the performances of AdaBoost were improved by a factor of approximately 1.3 and 1.6 over the ANN for 22 (red dots) and 52 (black boxes) training variables respectively, shown in Fig. 4(b). The best results for the ANN were found with 22 variables, while the best results for boosting were found with 52 variables. Comparison of the best ANN results and the best boosting results indicates that, when trained by the signal and neutral current π^0

background, the ANN results kept approximately 1.5 times more background events than were kept by the boosting algorithms for about 50% ν_e CCQE efficiencies.

In Fig. 4(c), the ratio of the background kept for a 52 variable AdaBoost to that for a 21(red dots—results for AdaBoost trained by the signal and all kinds of backgrounds)/22(black boxes—results for AdaBoost trained by the signal and neutral current π^0 background) variables is shown as a function of ν_e efficiency. It can be seen that the AdaBoost performance is improved by the use of more training variables.

The above ANN and AdaBoost performance comparison with different input variables indicates that AdaBoost can improve the PID performance significantly by using more input variables, even though many of them have weak discriminant power; ANN, however, seems unlikely to make full use of all input variables because it is more difficult to optimize all the weights between ANN nodes, given more nodes in both the input and the hidden layers. For the MiniBooNE Monte Carlo samples, the ANN are optimum for approximately 20 PID variables. The authors have found a similar number to be true for several other applications. In general, the optimum number for ANN may vary depending on the strength of the PID variables and the correlations between them.

Further evidence of this effect comes from the S-fitter [10], a second reconstruction—PID program set for the MiniBooNE. A systematic attempt was made to find the optimum sets of variables for ANN and for boosting classifiers by using ν_e CCQE signal and π^0 background (which includes 25 reaction channels). It is found that, for the S-fitter, the optimum ANN result is achieved by a selected set of 22 variables, while for boosting, no obvious improvement is seen after a selected optimum set of 50 variables is used. Comparison of the best ANN results and the best boosting results indicates that, for a given fraction of ν_e CCQE events kept, the ANN results kept about 1.2 times more π^0 background events than were kept by the boosting algorithms within the target range of keeping close to 50% of the ν_e CCQE events.

As noted in the introduction, two boosting algorithms are considered in the present paper.
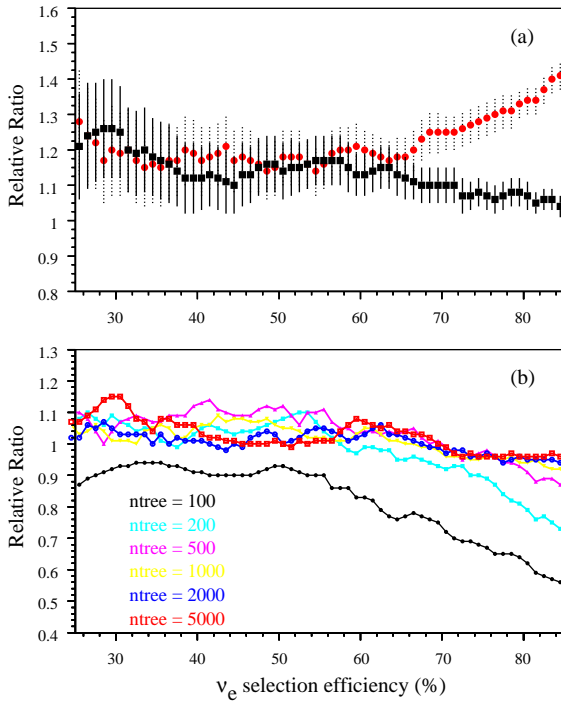
Fig. 5. Comparison of AdaBoost and ε-Boost performance with different decision tree sizes (8 and 45 leaves per decision tree) versus the intrinsic ν_e CCQE selection efficiency. (a) Relative ratio is defined as the number of background events kept for a decision tree of 8 leaves divided by that for a decision tree of 45 leaves, red dots with error bars represent results from AdaBoost and black boxes with error bars for ε-Boost. The tree iterations were 10 000 for 8 leaves/tree and 1800 for 45 leaves/tree, respectively. (b) Relative ratio here is the number of background kept for AdaBoost divided by that for ε-Boost with $N_{leaves} = 45$. The performance comparisons of AdaBoost and ε-Boost with different tree iterations are shown in different colors, $N_{tree} = 100$ (black), 200 (cyan), 500 (magenta), 1000 (yellow), 2000 (blue), 5000 (red).

The comparison of AdaBoost and ε-Boost performance is shown in Fig. 5, where parameters $\beta = 0.5$ and $\varepsilon = 0.01$ were selected for AdaBoost and ε-Boost training, respectively. The comparison between small tree size (8 leaves) and large tree size (45 leaves) with a comparable overall number of decision leaves, indicates that large tree size with 45 leaves yields $10 \sim 20\%$ better performance for the MiniBooNE Monte Carlo samples shown in Fig. 5(a). Increasing the tree size past 45 leaves did not produce appreciable improvement.

Comparison of AdaBoost and ε-Boost performance for the background contamination versus the intrinsic ν_e CCQE selection efficiency as a function of the number of decision tree iterations is shown in Fig. 5(b). A smaller relative ratio implies a better performance for AdaBoost. The performance of AdaBoost is better than that of ε-Boost if the relative ratio is less than 1. Boosting performance in the high signal efficiency region is continuously improved for more tree iterations. AdaBoost has better performance than ε-Boost for less than about 200 tree iterations, but becomes slightly worse than ε-Boost for a large number of tree iterations, especially for ν_e signal efficiency below $\sim 60\%$. For higher ν_e signal efficiency ($> 70\%$), AdaBoost works slightly better than ε-Boost.

4. Conclusions

PID variables obtained using the R-fitter and the S-fitter event reconstruction programs for the MiniBooNE experiment were used to separate signal events from background events. The ANN and the boosting algorithms were compared for PID. Based on these studies with the MiniBooNE Monte Carlo samples, the boosting algorithms, AdaBoost and ε-Boost, improved PID performance significantly compared with the artificial neural network technique. This improvement manifested itself when a large number of PID variables was used. For a small number of variables, the ANN classification was competitive, but as the number of variables was increased, the boosting results proved more efficient and superior to the ANN technique. If more variables are needed, boosting will use them as necessary.

It was also found that boosting with a large tree size of 45 leaves worked significantly better than boosting with a small tree size, 8 leaves, as recommended in some statistics literature.

The boosting technique proved to be quite robust. If a transformation of variables from x to $y = f(x)$ is made, then as long as the ordering is preserved, that is if $x_2 > x_1$, then $y_2 > y_1$, the boosting results are unchanged. ANNs must be tuned for temperature, learning rate and other variables, while for boosting, there is much less to vary and it is quite straightforward.

B.P. Roe et al. / Nuclear Instruments and Methods in Physics Research A 543 (2005) 577–584

There are certainly applications where ANNs prove better than boosting. However, for this application boosting appears superior and seems to be exceptionally robust and simple to use. It is anticipated that boosting techniques will have wide application in physics.

Acknowledgements

References

[1] E. Church, et al., BooNE Proposal, FERMILAB-P-0898, 1997.

[2] A. Aguilar, et al., Phys. Rev. D 64 (2001) 112007.

[3] Y. Freund, R.E. Schapire, Experiments with a new boosting algorithm, in: Proceedings of COLT, ACM Press, New York, 1996, pp. 209–217.

[4] J. Friedman, Greedy function approximation: a gradient boosting machine, Ann. Statist. 29 (5) (2001) 1189; J. Friedman, T. Hastie, R. Tibshirani, Additive logistic regression: a statistical view of boosting, Ann. Statist. 28 (2) (2000) 337.

[5] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, Classification and Regression Trees, Wadsworth International Group, Belmont, California, 1984.

[6] R.E. Schapire, The boosting approach to machine learning: an overview, MSRI Workshop on Nonlinear Estimation and Classification, 2002.

[7] Y. Freund, R.E. Schapire, A short introduction to boosting, J. Jpn. Soc. Artif. Intell. 14 (5) (1999) 771 (Appearing in Japanese, translation by Naoki Abe).

[8] J. Friedman, Recent advances in predictive (machine) learning, Proceedings of Phystat, Stanford University, September 2003.

[9] B.P. Roe, et al., BooNE-TN-117, March 18, 2004. (Journal paper under preparation.)

[10] I. Stancu, et al., BooNE-TN-36, September 15, 2001; BooNE-TN-50, February 18, 2002; BooNE-TN-100, September 19, 2003. (Journal paper under preparation.)