\( \nu_e \) and \( \pi^0 \) Separation in the MiniBooNE Experiment by using the Boosting Algorithm

Ji Zhu\(^ \dagger \), Hai-Jun Yang, Byron P. Roe

Department of Physics, University of Michigan, Ann Arbor, MI 48109, USA
\hspace{1cm} \dagger \hspace{0.1cm} Department of Statistics, University of Michigan, Ann Arbor, MI 48109, USA

Abstract

We report tests on a new analysis technique, the boosting algorithm, for \( \nu_e \) and \( \pi^0 \) separation in the MiniBooNE experiment at the Fermi National Accelerator Laboratory (FNAL). 0.46\% \( \pi^0 \) contamination ratio with 50\% \( \nu_e \) selection efficiency was achieved for a test sample. This is a significant improvement over the results obtained with an artificial neural network (ANN) performance. For the ANN, we had obtained 1.07\% \( \pi^0 \) contamination ratio with 44\% \( \nu_e \) efficiency.

1 Introduction

The Artificial Neural Network (ANN) technique has been widely used in data analysis of High Energy Physics (HEP) experiments in the last ten years. The use of the ANN technique usually gives better results than the traditional simple-cut techniques. In this report we introduce another data mining technique, so called boosting, for data analysis in the MiniBooNE experiment.

The boosting algorithm is one of the most powerful learning techniques introduced in the past decade. It is a procedure that combines many classifiers to achieve a final powerful classifier. We consider two versions of the boosting algorithm in this paper, AdaBoost[3] and \( \epsilon \)-Boost[4].

Before describing the boosting algorithm, we introduce some notations: we have a set of training data \((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\), where \( x_i \) represents variables containing information for the \( i \)th event, \( y_i \in \{-1, 1\} \) represents \{background, signal\} for the \( i \)th event. Then the AdaBoost algorithm and the \( \epsilon \)-Boost algorithm go as the following:

- **AdaBoost Algorithm**
  1. Initialize the event weights \( w_i = 1/n \), \( i = 1, 2, \ldots, n \)
  2. For \( m = 1 \) to \( M \):
     2.a Fit a classifier \( T_m(x) \) to the training data using weights \( w_i \)
2. b Compute
\[ err_m = \frac{\sum_{i=1}^{n} w_i I(y_i \neq T_m(x_i))}{\sum_{i=1}^{n} w_i} \]

2. c Compute \( \alpha_m = 0.5 \times log((1 - err_m)/err_m) \)

2. d Set \( w_i \leftarrow w_i \times exp(\alpha_m I(y_i \neq T_m(x_i))) \), \( i=1, 2, ..., n \)

2. e Re-normalize \( w_i = w_i/\sum_{i=1}^{n} w_i \)

3. Output \( T(x) = \sum_{m=1}^{M} \alpha_m T_m(x) \)

\* \( \epsilon \)-Boost Algorithm:
1. Initialize the event weights \( w_i = 1/n, i = 1, 2, ..., n \)
2. For \( m = 1 \) to \( M \):
   2.a Fit a classifier \( T_m(x) \) to the training data using weights \( w_i \)
   2.b Set \( w_i \leftarrow w_i \times exp(2\epsilon I(y_i \neq T_m(x_i))) \), \( i=1, 2, ..., n \)
   2.c Re-normalize \( w_i = w_i/\sum_{i=1}^{n} w_i \)
3. Output \( T(x) = \sum_{m=1}^{M} \epsilon T_m(x) \)

Probability estimate:
\[ P(Y = 1|x) = \frac{1}{1 + e^{-2T(x)}} \]

Final Selection:
\[ G(x) = 1 \text{ (i.e., signal) if } P(Y = 1|x) \geq c. \]

where \( c \) is a constant that we can adjust to achieve different signal efficiencies.

In the above algorithms, \( T_m(x) \) is a general classifier. In our experiments, we choose \( T_m(x) \) to be a binary classification tree. Every node of the classification tree \( T_m(x) \) has two daughters, unless it is a terminal node (leaf). For a selected splitting variable, all events with values of the splitting variable less than a threshold value are put in the left daughter node, all other events with values of the splitting variable equal to or greater than the threshold value are put in the right daughter node. To select the splitting variable and the threshold value, we use a criterion based on the Gini index. The Gini index of a node is defined as:
\[ \text{Gini} = P \times (1 - P) \times N_{\text{tot}} \times \sum_{i=1}^{N_{\text{tot}}} w_i \]

where
\[ P = N_{\text{signal}}/N_{\text{tot}} \]
representing the purity of the node. \( N_{\text{signal}} \) and \( N_{\text{tot}} \) are the number of signals and the total number of events in the node. \( w_i \) is the weight of event \( i \) in the node.

We then consider the following criterion:
\[ \text{Criterion} = \text{Gini}_{\text{father}} - \text{Gini}_{\text{left daughter}} - \text{Gini}_{\text{right daughter}} \] (1)
and to grow the tree we select the splitting variable and the threshold value that simultaneously maximizes the criterion. For a given event \( x \), \( T_m(x) \) depends on the purity of the leaf node (or the terminal node) the event falls in. If the purity of the leaf node is less than one half, \( T_m(x) = -1 \), otherwise \( T_m(x) = 1 \).

The tree size and the iteration number \( M \) are tuning parameters we need to select. In the \( \epsilon \)-Boost algorithm, \( \epsilon \) is also a tuning parameter.

Three independent boosting programs were constructed for cross checking. One was written by Ji Zhu in the R language\(^5\). R is a widely used statistical package in the statistics community. The other two were written in Fortran by Byron Roe and Hai-Jun Yang, respectively. In Zhu and Roe’s programs, all training events are sorted in ascending values for all variables. The criterion (1) is calculated event by event for all variables. In Yang’s program, training events are filled into many bins (let \( N_{\text{bin}} \) denote the number of bins) with a uniform step for each variable and the criterion (1) is calculated bin by bin. The bin number is also a tuning parameter in Yang’s program. \( N_{\text{bin}} = 100 \) was used for the \( \nu_e \) and \( \pi^0 \) separation. Detailed results are shown in section 2.

2 Analysis and Results

For the \( \nu_\mu \to \nu_e \) oscillation search in the MiniBooNE experiment, the main background come from intrinsic \( \nu_e \) contamination in the beam, mis-identified \( \nu_\mu \) quasi-elastic scattering and mis-identified neutral current \( \pi^0 \) production. Good sensitivity for the \( \nu_e \) appearance search requires low background contamination. Here, we use the boosting algorithm to obtain the \( \nu_e \) and the \( \pi^0 \) separation. The \( \pi^0 \) contamination ratio is defined as,

\[
\text{contamination ratio} = \frac{\text{fraction of } \pi^0 \text{ kept}}{\text{fraction of } \nu_e \text{ kept}}
\]

We used a sample of \( \pi^0 \) and \( \nu_e \) events, not the NUANCE events, that had the following cuts:

- Reconstructed radius less than 498.7 cm.
- More than 200 signal PMT’s hit
- Less than 6 veto hits

All fractions quoted are for the sample after these cuts have been made. After the cuts, 34457 \( \nu_e \) Monte Carlo events and 33914 \( \pi^0 \) Monte Carlo events were reconstructed by ‘RoeFitters’ package\(^6\) in the MiniBooNE Analysis Framework. The signature of each event is given by 49 variables shown in Table 1. All variables are used in the boosting algorithm for training and testing. 16500 \( \nu_e \) and 16500 \( \pi^0 \) events were selected randomly as the training sample, and the rest of the events served as the test sample.

As a comparison, we achieved 1.07% \( \pi^0 \) contamination ratio with 44% \( \nu_e \) efficiency using an ANN technique for this Monte Carlo sample.

For the AdaBoost algorithm, we chose the tree size, or the number of leaves (i.e., terminal nodes) of each binary tree \( T_m(x) \) to be \( N_{\text{leaves}} = 35 \). The \( \pi^0 \) contamination ratio
versus number of tree iterations for 50% $\nu_e$ selection efficiency is shown in Figure 1. The $\pi^0$ contamination ratio is approximately 0.6% for tree iterations ranging from 300 to 1000. The $\pi^0$ contamination ratio as a function of $\nu_e$ selection efficiency is shown in Figure 2.

For $\epsilon$-Boosting, the parameter $\epsilon = 0.01$ and the number of leaves $N_{leaves} = 45$ were used. The $\pi^0$ contamination ratio versus number of tree iterations for 50% $\nu_e$ selection efficiency is shown in Figure 3. The $\pi^0$ contamination ratio is approximately 0.46% for 1000 tree iterations. The $\pi^0$ contamination ratio as a function of $\nu_e$ selection efficiency is shown in Figure 4.

### Table 1: 49 Variables from RoeFitters

<table>
<thead>
<tr>
<th>Roe_chir</th>
<th>Roe_chia</th>
<th>Roe_chire</th>
<th>Roe_chiae</th>
<th>Roe_abar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roe_asigma</td>
<td>Roe_askew</td>
<td>Roe_askewmu</td>
<td>Roe_avgtine</td>
<td>Roe_aksigntime</td>
</tr>
<tr>
<td>Roe_chiphi</td>
<td>Roe_xintmax</td>
<td>Roe_bigang</td>
<td>Roe_avgtime</td>
<td>Roe_earlylit</td>
</tr>
<tr>
<td>Roe_timekurt</td>
<td>Roe_s2mis</td>
<td>Roe_gfrac9</td>
<td>Roe_leftright</td>
<td>Roe_costhc</td>
</tr>
<tr>
<td>Roe_latelit</td>
<td>Roe_scratio</td>
<td>Roe_ffrac9</td>
<td>Roe_rkurt</td>
<td>Roe_chira5</td>
</tr>
<tr>
<td>Roe_qfrac6</td>
<td>Roe_qfrac8</td>
<td>Roe_ffrac9</td>
<td>Roe_rkurtmu</td>
<td>Roe_tkurt</td>
</tr>
<tr>
<td>Roe_chisqmich</td>
<td>Roe_tbar</td>
<td>Roe_ffrac5</td>
<td>Roe_chisqst</td>
<td>Roe_chisqst</td>
</tr>
<tr>
<td>Roe_sbar</td>
<td>Roe_ssigma</td>
<td>Roe_ffrac6</td>
<td>Roe_pi0mass</td>
<td>Roe_cosg</td>
</tr>
<tr>
<td>Roe_tave</td>
<td>Roe_triple</td>
<td>Roe_ffrac8</td>
<td>Roe_tave</td>
<td>Roe_sratio</td>
</tr>
<tr>
<td>Roe_cosg</td>
<td>Roe_geratio</td>
<td>Roe_ffrac9</td>
<td>Roe_trpksig</td>
<td>Roe_earlylit</td>
</tr>
</tbody>
</table>

3 Preliminary Results with a NUANCE Sample

For the NUANCE sample, we divide $\nu_\mu$ events into “pi-like” and “mu-like”. As a (very) preliminary division, we defined event types 1, 2, 3, 5, 7, 9, 10, 12, 14, 16, 17, 20, 21, 28, 31, 32, 39, 40, 41, 46, 47, 48, 53, 54, 55, 60, 61, 62, 67, 70, 73, 76, 79, 80, 83, 85, 86, 89, 91, 95, 97, 99 as mu-like and the rest as pi-like.

For the $\nu_e$, only the quasi-elastic events (event type 1) were kept. After this selection, and after the cuts listed in the preceding section, the size of the available sample of events was quite small, i.e., 2187 signal and 3199 background events. The training sample only had 986 signal and 1414 background events. (We are currently generating a larger sample.) We used 30 leaves per tree and ran for 1000 trees. $\epsilon$-Boost was used with $\epsilon = 0.005$. For 50% signal efficiency, the pi-like contamination ratio was reduced to 1.67%.

4 Conclusion

Based on our current studies, both boosting algorithms, AdaBoost and $\epsilon$-Boost, improved the $\nu_e$ and the $\pi^0$ separation significantly compared with the neural network performance. 0.46% $\pi^0$ contamination ratio with 50% $\nu_e$ selection efficiency was achieved by using the $\epsilon$-Boost algorithm. In addition, the $\epsilon$-Boost algorithm seems to perform better than the AdaBoost
algorithm for this particular sample. We plan to apply the boosting algorithm to separate $\nu_e$ and other major backgrounds shortly.

References


Figure 1: The $\pi^0$ contamination ratio versus number of tree iterations for 50% $\nu_e$ selection efficiency.
Figure 2: The $\pi^0$ contamination ratio as a function of $\nu_e$ selection efficiency.
nue and pi0 separation (eboost, 45 leaves, epsilon=0.01)

Figure 3: The $\pi^0$ contamination ratio versus number of tree iterations for 50% $\nu_e$ selection efficiency.
Figure 4: The $\pi^0$ contamination ratio as a function of $\nu_e$ selection efficiency.
Figure 5: The pi-like contamination ratio for NUANCE events versus number of tree iterations for 50% $\nu_e$ selection efficiency.