Brief Manual for Using U. Michigan Boost programs

Byron P. Roe and Hai-jun Yang
University of Michigan, Ann Arbor, Michigan 48109, U.S.A.,

Abstract
This note gives a brief description of the boosting method and instructions for using the U. Michigan Boost programs.

1. Description of Boosted Decision Trees

1.1 Decision Tree

We start by describing a decision tree. Suppose you are trying to divide events into signal and background. You have Monte Carlo samples of each. Each Monte Carlo sample is divided 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 tree after it is trained.

For each event, suppose there are a number of particle id variables useful for distinguishing between signal and background. Firstly, for each particle id 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. We started with 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.

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

$$P = \frac{\sum_s W_s}{\sum_s W_s + \sum_b W_b},$$

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

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

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

$$Gini_{left\ daughter} + Gini_{right\ daughter}.$$

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

$$Criterion = Gini_{father} - Gini_{left\ daughter} - Gini_{right\ daughter}.$$

At the end, if a leaf has purity greater than 1/2 (or whatever is set), then it is called a signal leaf and if the purity is less than 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.

Decision trees have been available for some time[1]. They are known to be pwerful but unstable. That is, a small change in the training sample can give a large change in the tree and the results.

1.2 Boosting

Within the last few years a great improvement has been made[2, 3, 4]. 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 the 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 to cut on, one can select a desired fraction of the signal or a desired ratio of signal to background. For those familiar with neural nets, the use of this score is the same as the use of the neural net value for a given event.

Statisticians have found that this method of classification is very efficient and robust. Furthermore, as we will see the amount of tuning needed is rather modest compared with neural nets. It works well with many particle id 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.

For our mini-BooNE experiment with the event reconstruction program rfitter, we have 49 particle id variables (plus energy, radius, and distance_to_wall). A direct comparison of this method with the best we were able to do with neural nets found that with this method we more than doubled the sensitivity of the experiment. That is, to get the same sensitivity with a neural net method required more than double the number of events.

1.3 Some methods for boosting the weights

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

- $x_i =$ the set of particle id 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 $y_i = T_m(x_i)$.

We have used at least two methods for boosting the weights of the misclassified events in the training sample.

The first boosting method is called “Ada boost”[5]. Define for the mth tree:

$$err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq T_m(x_i))}{\sum_{i=1}^{N} w_i}.$$

Calculate:

$$\alpha_m = 0.5 \times \ln((1 - err_m)/err_m).$$

Change the weight of each event i, $i = 1, ..., N$.

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

Renormalize the weights.

$$w_i \rightarrow \frac{w_i}{\sum_{i=1}^N w_i}.$$

The score for a given event is

$$T(x) = \sum_{m=1}^{M} \alpha_m T_m(x),$$

which is just the weighted sum of the scores over the individual trees.

The second boosting method is called "epsilon boost"[6], or sometimes "shrinkage". After the mth tree, change the weights of each event i, $i = 1, ..., N$.

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

where ϵ is a constant of the order of 0.01. Renormalize the weights.

$$w_i \rightarrow \frac{w_i}{\sum_{i=1}^N w_i}.$$

The score for a given event is

$$T(x) = \sum_{m=1}^{M} \epsilon T_m(x),$$

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

2. Parameters in the programs

There are three separate boost programs built by the U. Michigan mini-BooNE group for the purpose of building trees. One was built by Hai-jun Yang and the other two, with only a minor difference between them, were built by Byron Roe. The Yang program and one of the Roe programs, for each variable, divides the events into bins, grouping events with nearly the same value of that variable together. There are minor differences in the way this grouping is done between the two progams. The second Roe program does not group the events into bins, but is otherwise the same as the first Roe program. In our tests, the results of all three programs are quite similar.

One might wish to build separate trees against various different backgrounds. For example, in mini-BooNE, one might run ν_e vs ν_μ and ν_e vs neutral current π^0 events. The second program combines these separate runs and determines what events pass all sets of trees. For data events it will also write out a file with the event number, particle id variables, and scores for each of the tree sets for each event. For the Roe set of programs this is done in combine_boost.

We start by discussing the options in the programs to build trees. All of these programs have a number of options in common. These options are given below.

- Choose whether to use Ada boost or epsilon boost. If ADA boost is chosen then the standard α can be multiplied by a constant if desired. If epsilon boost is chosen, epsilon can be specified.
- Choose the size of the training sample, n_{train}, independently for signal and background.
- The training sample can be randomized. That is, rather than pick the first n_{train} events as training events, one can take a random sample of the events. If the MC events start out random themselves, this would appear to be unnecessary. However, it is very useful if one wishes to examine the sensitivity of the results to the specific training sample chosen. By specifying a different initial random number seed, one can vary the training sample.

- The maximum number of leaves and nodes can be chosen. The number of leaves is equal to 1/2(number of nodes + 1). Originally, the statisticians suggested small trees with perhaps only 11 nodes. Our experience with mini-BooNE has been that large trees, with perhaps 89 nodes worked better.
- The number of trees that will be built can be chosen.
- The number of particle id variables can be chosen
- The maximum number of events (used in dimension statements) can be chosen. This is the maximum number of the total number of signal plus background events in the training plus testing samples.
- The Yang program and one of the Roe programs, for each variable, divides the events into bins, grouping events together with nearly the same value of that variable. The number of bins can be chosen. The minimum number of events in a bin can be chosen. If a bin has less than this number it is combined with an adjacent bin. The minimum number of events in a leaf is set at twice the minimum bin size.

In addition to these common options, the Roe and Yang programs have some options not in common. There are two Roe programs. The first looks at Monte Carlo training and test samples to find the sensitivity of the separation. This first program has two versions, one binning events (Boost_bins) and one not binning events (Boost_evts).

For the first Roe program, options not described above are given below.

- If the unbinned version of the Roe program is used, one can select a minimum weight limit for the sum of the weights in a given node. If the sum of the weights of the events within that node is less than this value the node is declared an ending node, i.e., a leaf.
- J. Friedman has suggested that it might be useful to use only a fraction of the training events, randomly chosen, from among the training sample, for each tree. That fraction can be selected. If it is greater than 1, then all of the training events are used for each tree.
- J. Friedman has also suggested that it might be useful to use only a fraction of the variables, randomly chosen, for each node within a tree. That fraction can be selected. If it is greater than 1, then all of the nodes select among all of the variables.
- Purity cut. The usual purity cut is to define a leaf as a signal leaf if the purity is greater than 0.5 and a background leaf otherwise. However, the cut point can be changed if desired. It was sometimes found that 0.6 worked slightly better than 0.5 for the mini-BooNE situation. Whatever is chosen, the cut point is, in fact, the cut point specified only if the numbers of events in the signal and background training samples are equal. If this is not true, then the chosen purity cut is renormalized to reflect the inequality.
- The results are printed out for every kth tree. k is currently set at 25.
- At each printout, the program iteratively finds the score for which a given fraction of the signal is kept. This fraction and the number of iterations to find this fraction can be set. Currently the fraction is 0.5 and the number of iteractions is 15.

For the Roe program to combine tree sets, combine_boost, the parameters to set are given below.

- The number of tree sets to combine is specified.
- If one of the sets was signal vs all backgrounds, set all_bkrd_set to that set.
- If the input events are data, not MC events set $data\_set$ to true
- There may a different distribution of training and signal events in the various tree sets. The default is to take the minimum testing set. To do this set $num\_train\_sig\_fix = -1$. (Note that if boost picked a scrambled set for testing, combine will not work properly.) If you want to fix the number of events that will be considered training events set $num\_train\_sig\_fix$ to that number.
- The number of particle id variables in the events must be set.

- The program will run through for a set of scores (sum_T_m) and then query you online to set a new set. The initial set will be the best values found for the individual boost runs by default. To use this default set $sum\_T\_m\_fix = 0$ for each tree set. This variable has dimension equal to the number of tree sets. If it is set non-zero, then those values will be used for the initial run.

3. How to set the parameters

3.1 The Roe programs

The two first-part Roe programs are boost_evts.f, which is the unbinned version and boost_bins.f, which is the binned version. Both use the include file boost.inc. The programs are linked to the Roe web page: http://www.mhp-physics.lsa.umich.edu/∼roe/

The parameters are all set in the include file. subroutine readin_data will have to be rebuilt using the given version as a model, to input data in whatever is your favorite format. You may wish to change the name of the output files which are written to unit 50 in subroutine check_purity.

There is output to the online window and to unit 40, nominally every 25th tree, giving the tree number, the fraction of signal kept, the fraction of background kept, the ratio of the two fractions, the number of background and signal events kept, and the score which keeps the desired fraction, nominally .5 of the signal events.

An ntuple, boost.hst, is made up which has the number of training signal and background events, and then for every 25th tree, the tree number, followed by the number of signal events, the number of background events and the score for 21 selected values of score with the center value being the score keeping the desired fraction of the signal events. In addition a number of histograms are made up. Histogram 501 is the fraction of signal kept versus tree number for the nominal desired fraction. (Because of the iteration, this will not be exactly the nominal fraction for the first trees, but rapidly approaches it as a function of tree number.) Histogram 502 is the fraction of background vs tree number for the nominal desired fraction, i.e., for score11. Histograms 503 and 504 are the ratios of the fraction of background kept divided by the fraction of signal kept plotted on two different scales. The histograms 1–49 are the plots of the particle id variables for the signal and 101–149 are the plots of the particle id variables for the background. You will need to modify these plots to match your variables.

The second program is combine_boost. The parameters are again mostly set in the include file, combine_boost.inc. There is again a program readin and you will wish to change the name of the files to be read in. The format of the event files will likely be different and you will need to modify this routine accordingly. The value of $sum\_T\_m\_fix$ is set as a data statement in subroutine find_passed_events since Fortran didn't allow a dimensioned parameter statement.

For data events, combine_boost outputs a file with the particle id information and the scores of the events vs each of the sets of trees. You may wish to change the name or format of this file. It is written to unit 60 in subroutine find_passed_events.

3.2 The Yang programs

The Yang boosting programs are available at URL, http://tenaya.physics.lsa.umich.edu/simhyang/boosting/boosting.tar.gz. All files needed to run boosting programs are listed in the following,

- boost-train.f - main program to train and test boosting
- boost.inc - include file, need to modify input variables, node number, tree iterations, training events etc.
- hbookboost.inc - include file for ntuples information
- boost-test.f - main program to test boosting performance only
- boost-test.inc - include file for testing boosting performance

- data file - input data file for training and test, the first line indicates number of signal and backgrounds events in the file, the rest lines show signal information followed by backgrounds.
- makefile - make executable file, please make sure you set correct link to CERN library
- run - makefile and run the executable file

All input parameters are set in the “boost.inc” file listed in the following,

- nevt - number of total events(maximum) include signal and background, e.g. 100000
- nvar - number of variables plus 2, one for event number index and the other for signal and background identification, e.g. 52
- ntr - number of classification trees plan to be constructed, e.g. 1000
- node - maxmimum nodes of binary tree, the leaves for the tree is node/2, e.g. 90
- nsig_train - number of trainning events for signal, e.g. 20000
- nbkgd_train - number of training events for background, e.g. 20000
- nc - number of bins to separate events in each node, e.g. 100
- iepsilon - 1 for epsilon-boosting algorithm, otherwise for adaboost, e.g. 1
- epsilon - parameter to increase weight of mis-classified events, e.g. 0.01
- beta - tunning parameter for adaboosting algorithm, e.g. 0.5
- psig - if purity of signal in one leaf is greater than psig, then all events in this leaf are classified as ’signal’, e.g. 0.5
- irand - 0 for obtaining MC sequentially for training, 1 for selecting MC randomly, e.g. 0
- nmin - the minimal number of events in one node
- ngini_index - 1 means using gini_index criterion to select next splitter, otherwise, using the following two criterions to find the tree leaves, e.g. 1
  c1 nmax - if number of events in one node is less than nmax, then it’s a terminal node, e.g. 2000
  c2 psig0 - if purity*(1.-purity) is less than psig0, then it’s a terminal node, e.g. 0.05

The output files are listed below,

- out-debug: for debugging
- test.dat: usually, part of the MC samples will be selected for boosting training and the rest will be used for testing saved in this file, this is input file for “boost-test.f”.
- treeindex.dat: all classification trees information are recorded in this file, it will be used to test other samples, this is input file for “boost-test.f”.
- boost.ntp (boost-test.ntp): some useful information are saved in this file, it’s good to make some plots about the boosting performance. Ntuple ID=10, there are 4 variables listed below,
  1. ntree - number of tree iterations
  2. eff - efficiency of signal
  3. purity - purity of signal
  4. ratio - contamination ratio which is defined as fraction of background kept divided by fraction of signal kept
  Ntuple ID=12, it includes the boosting output for test sample,
  1. ntree - number of tree iterations
  2. boost - boosting output
  3. signal - 1 for signal, otherwise, backgrounds
  For instance, if you want to look at the contamination ratio versus signal efficiency at tree iteration of 1000, then use “nt/pl 10.ratio%eff ntree=1000” to make the plot in Physics Analysis Workstation(PAW, a widely used data analysis software package in High Energy Physics Community); “nt/pl 10.ratio%ntree abs(eff-50)¡0.1” will show the contamination ratio as a function of tree iteration for signal efficiency of 50%.

4. REFERENCES

References

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

[2] Robert E. Schapire, *The boosting approach to machine learning: An overview*, MSRI Workshop on Nonlinear Estimation and Classification, (2002).

[3] Yoav Freund and Robert E. Schapire, *A short introduction to boosting*, Journal of Japanese Society for Artificial Intelligence, **14(5)**, 771-780, (September, 1999). (Appearing in Japanese, translation by Naoki Abe.)

[4] J. Friedman, *Recent Advances in Predictive (Machine) Learning*, Proceedings of Phystat2003, Stanford U., (Sept. 2003).

[5] Y. Freund and R.E. Schapire,*Experiments with a new boosting algorithm*, Proc COLT, 209–217. ACM Press, New York (1996).

[6] J. Zhu, Private Communication (2003).