Decision Trees in the Single Top Quark Search

Steve Herrin
Rice University
University of Washington REU 2006
Advisor: Gordon Watts

25 August 2006

1 Background

1.1 Single Top Quark Production[2]

In the Standard Model of particle physics, the top quark is one of six quarks and the heaviest elementary particle that has been discovered. This discovery took place in 1995 at the Fermilab Tevatron Collider. The top quark was found by looking for top-antitop pair production, which occurs through the strong interaction.

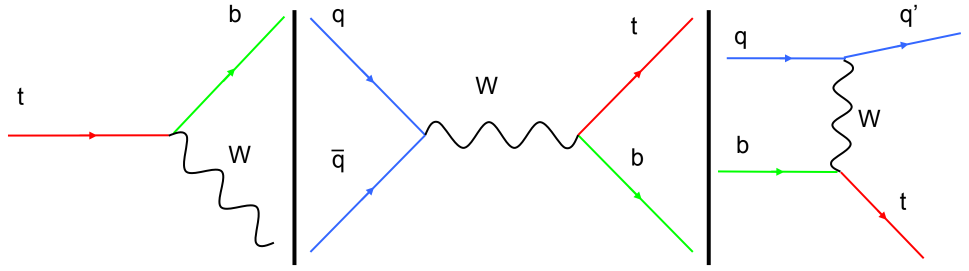

Figure 1: Left: A top quark decays via the weak interaction. Center: s-channel production of a single top quark. Right: t-channel production of a single top quark.

The top quark is known to decay electroweakly into a W boson and a bottom quark. However, as shown in Figure 1, the same interaction vertex can occur to produce a single top quark. In the s-channel (sometimes called the tb channel), the collision produces a t and a b quark. In the t-channel (sometimes called the tqb channel), a t, b, and one other quark are produced. Studying the electroweak interaction of the top quark provides several tests of the Standard Model.

In particle physics, the cross section of an interaction when colliding two particles is a way of expressing how probable that interaction is. The Standard Model predicts cross sections of about 0.88 pb[†] for the s-channel and 1.98 pb for the t-channel, at the 1.96 TeV center-of-mass energy obtained at the Tevatron Collider. The experimental goal is to measure these cross sections. If the measured cross-section differs from the predictions, it may indicate some new physical process at work.

To date, the DØ experiment has collected over 1 fb^{-1} of data from the Tevatron Collider. Therefore, there is expected to be on the order of 1000

[†]picobarn, where a barn is 1×10^{-28} m^2. 1 pb corresponds to roughly 1 in every 75 billion proton-antiproton collisions.

single top production events in this data. However, searching for these events proves difficult. The heavier quarks and bosons are too short lived to ever reach the detector, so we are limited to observing their decay products, such as leptons and jets of hadrons. However, there are other processes that produce similar decay products in similar distributions to those of a single top quark. The most significant of these is the production of a W boson and jets of hadrons (referred to as W+jets). This process has a cross section of over 1000 pb. The next most significant background is top-antitop production (referred to as $t\bar{t}$). The $t\bar{t}$ background has a cross section of roughly 10 pb. Though a $t\bar{t}$ event produces more jets than a single t event, the detector can imperfectly reconstruct jets and lose some, leading to an event that looks similar to single t quark. Finding single t quark events, therefore, requires a powerful means of separating signal from background.

1.2 Classifiers

1.2.1 General

In the field of machine learning, a classifier is an algorithm that distinguishes between classes of events based on a set of input variables. Classifiers, therefore, are ideally suited for distinguishing between signal and background in the search for single t quark production.

Most classifiers work by first being trained on a set of events, which consist of a vector $\vec{x}$ of variables containing information about the event, as well as a number y, which indicates the class of the event. The variable vector for an unknown event is then tested by the classifier, which outputs a predicted value for y, based on the model built by the classifier during training.

1.2.2 Decision Trees

One of the more powerful types of classifiers are decision trees. The structure of a basic, binary decision tree used to separate two classes ($y = 0$ and $y = 1$, for simplicity), as shown in Figure 2, is very simple. At a given node, a certain variable is specified, as well as a value to split on. If, in a given event, the value of this variable is greater than the split value, the algorithm follows the left branch. Otherwise, it follows the right branch. This continues until the algorithm arrives at a 'leaf'. The leaf is assigned a purity score, which indicates how likely an event on that leaf is to belong to a certain class.

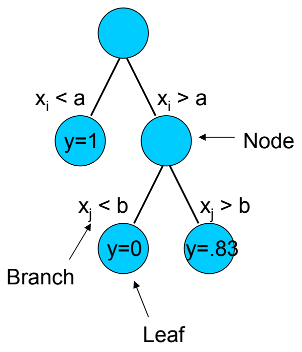

Figure 2: The node, branch, and leaf structure of a decision tree.

Training a decision tree is slightly more complicated than its structure implies. The most basic algorithms are recursive. At each node, for each variable, the classifier sorts the training events based on the value of the variable. The algorithm then considers all possible split value and determines which produces the greatest discrimination between the two classes. The amount of discrimination is usually quantified in one of two main ways: improvement in 'Gini', which represents how pure the resulting split subsets are, or the reduction in entropy of the resulting subsets. The tree then splits based the best choice of variable and split value, and repeats this process on the resulting nodes, until the improvements in discrimination fall below a certain cutoff value, and so a leaf is formed. The purity score of each leaf is assigned by taking the number of $y = 1$ events on that leaf divided by the total number of events on that leaf.

Decision trees have several of advantages over other types of classifiers. First, they are relatively quick to train. For comparison, consider another type of powerful classifier called a neural net. The neural net starts with all of its (rather complicated) structure in place, and then trains by adjusting the parameters of this structure to best fit its output the training data. This process can be very slow. On the other hand, the tree's structure is both grown and optimized from the bottom up, which proves to be much faster. Secondly, the structure of a decision tree can be easily parsed and understood by the human mind, and also somewhat mirrors human thinking.

However, decision trees have drawbacks, as well. The largest is that decision trees are not 'stable'. That is, a small change in the training data can sometimes lead to large changes in the resulting tree and its predictions. Additionally, all but the most sophisticated decision tree algorithms cannot take advantage of correlations between the input variables. If two variables are linearly correlated for $y = 1$, but not at all correlated for $y = 0$, the tree cannot take advantage of this unless another variable representing this correlation is added. Most decision tree algorithms, furthermore, are 'greedy' in their selection of variables. They always select the best variable at each individual node and will never choose a less powerful variable, even if it will allow even greater discrimination at later nodes. Finally, owing to its branch and leaf structure, the output of a decision tree is discrete. Ideally, the output of the decision tree would fall in a continuum between 0 and 1, reflecting that an event can resemble $y = 0$ or $y = 1$ to any possible degree, when it actually only outputs a limited number of values.

1.2.3 Ensemble Methods[3]

Fortunately, a number of ensemble methods allow the use of the quick speed of decision tree growth to overcome some of the algorithm's disadvantages. Because decision trees can be created quickly, it makes sense to combine and average multiple trees to get more stable and smoothy-distributed results. There are three main types of ensemble methods relevant to this paper: boosting, bagging, and random forest.

Boosting Boosting begins by creating a single tree with the basic algorithm. However, next, the boosting algorithm looks at the results of the classifier on the training data. Misclassified events are given an increased weight (so that they count for more than one event, now) and a new tree is created. This process continues a set number of times, or until boosting no longer provides any more improvement. The weighted average of all the resulting trees determines the algorithm's output. An example of how boosting might proceed is shown in Figure 3.

Bagging Bagging also generates a number of trees, but uses a different approach to do so. Instead of using all events, a bagged tree only considers a random subset of the training events. By training many trees this way, the averaged result of the trees tends to reenforce the result on those events that clearly belong in one class. An example of bagging is shown in Figure 3.

Random Forest[1] As with Bagging, the Random Forest algorithm trains many trees on random subsets of the data. However, it also injects random-

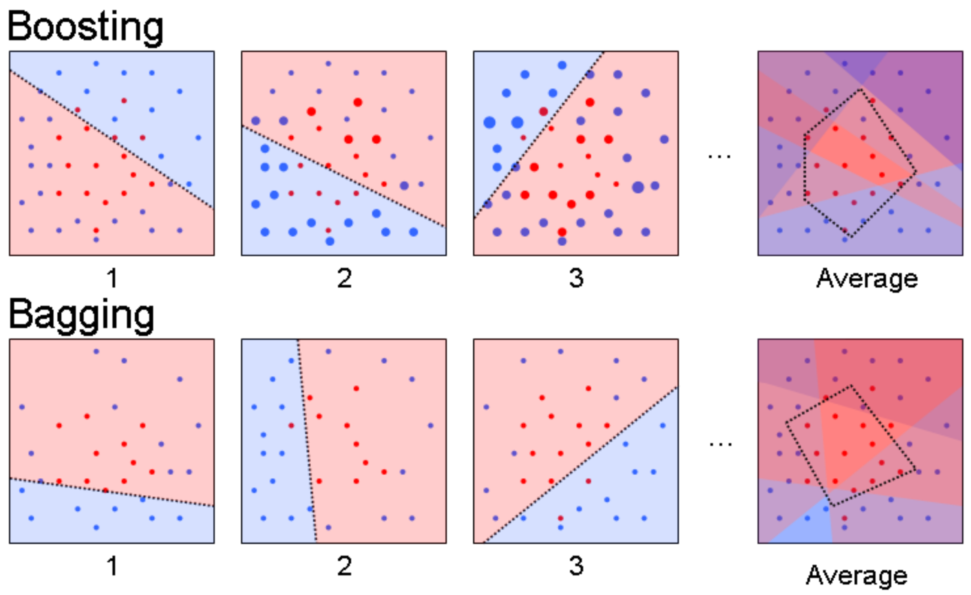

Figure 3: Boosting and bagging for 2 input variables. Thicker dots represent heavier weights. The dashed line represents an iso-output of the tree in variable space. Notice how while no individual tree correctly discriminates signal and background, the average tends to separate the inner red events from the outer blue events.

ness into the trees themselves. At each node of the tree, the tree randomly restricts which variables the node can branch on. This method, like other ensemble methods, generates trees that agree strongly on events that are firmly in one class, while disagreeing on more borderline events.

2 The Project

2.1 Coding

The basic tree algorithm and an algorithm for boosting were both functional when this project began. Bagging and random forest were implemented by expanding on this code, with the random forest being an additional feature within the bagging code, since both work using the same principle for generating subsets of data.

Additionally, tools were prepared to help test, analyze, and compare the performance and stability of these algorithms. One such tool, and perhaps the most useful, is one that allows '10-fold' evaluations. In a 10-fold evaluation, the training data is spit into 10 stratified[†] subsets, S_1, S_2, ..., S_{10}. Iterating i from 1 to 10, subset S_i is reserved for testing, while the remaining 9 subsets are used to train the classifier. By looking at the results of testing on each of the 10 subsets, it is possible to measure the stability of the algorithm. Additionally, summing or averaging the results (depending on the quantity to measure) allows analysis of the performance of the algorithm itself, rather than any individual tree.

2.2 Analysis

2.2.1 Performance

To test and compare the performance of the new ensemble methods, a 10-fold analysis was performed using simulated t-channel data for the signal, and simulated $t\bar{t}$ data as the background. Histograms showing the output of the trees on the test data are shown in Figures 4 and 5. Because the ensemble methods take the average result of many trees, the curves are much smoother than for the single tree. For 1 b-tagged events, the peaks of the bagged trees and random forest seem the least central. However, all the classifiers are producing a great deal of separation. In the 2 b-tagged events, the signal is roughly uniformly distributed, while the background still peaks. This suggests that the 1 b-tag signal is overwhelming the 2 b-tag signal when training the trees, because 1 b-tag events are much more common than 2 b-tag events in the t-channel.

The efficiency curves in Figure 6 were generated by integrating the individual output histograms for each of the 10-fold generated trees to find the 'background fraction' as a function of 'signal efficiency'. Background fraction is the fraction of background events that are incorrectly counted as signal after choosing a value of the decision tree's output as the cutoff between signal and background classifications. Signal efficiency is the fraction of signal events correctly classified as signal by this same cutoff. The optimal situation is a high signal efficiency while maintaining a low background fraction. From the figure, then, boosting appears to be the best performing classifier.

However, there are several parameters than can be adjusted when training bagged trees and random forest, such as the size of the training data subset

[†]'Stratified' in this sense means that the different classes of data are all found in approximately the same ratios as in the original training data

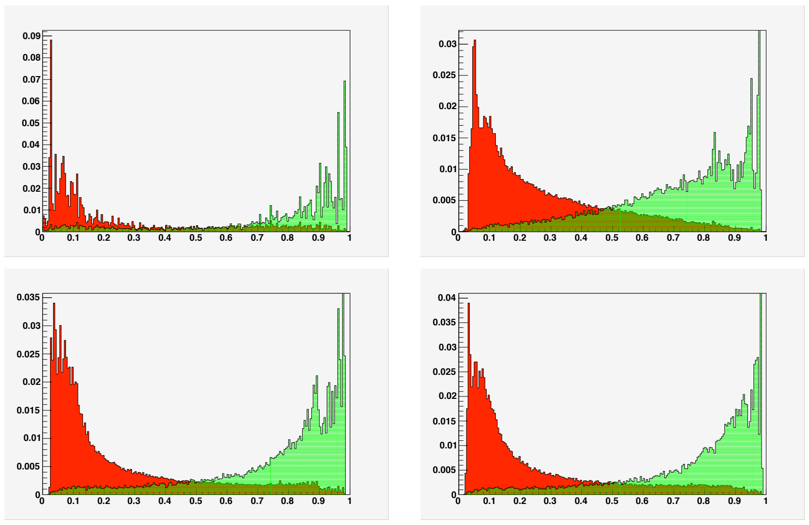

Figure 4: Decision tree results for t-channel (green) and $t\bar{t}$ (red) for single tree (upper right), boosted trees (upper left), bagged trees (bottom right), random forest (bottom left), in the case with 1 b-tag. Note the histograms are normalized. There is in reality much less signal than background.

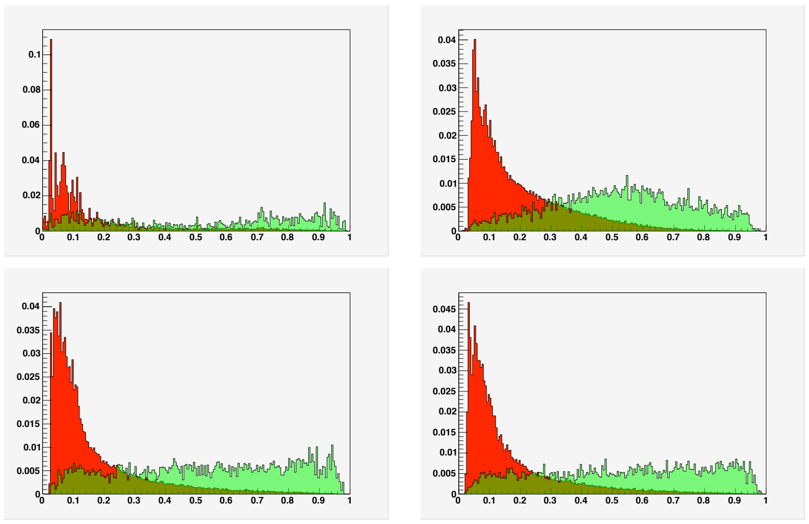

Figure 5: Decision tree results for t-channel (green) and $t\bar{t}$ (red) for single tree (upper right), boosted trees (upper left), bagged trees (bottom right), random forest (bottom left), in the case with 2 b-tags. Note the histograms are normalized. There is in reality much less signal than background.

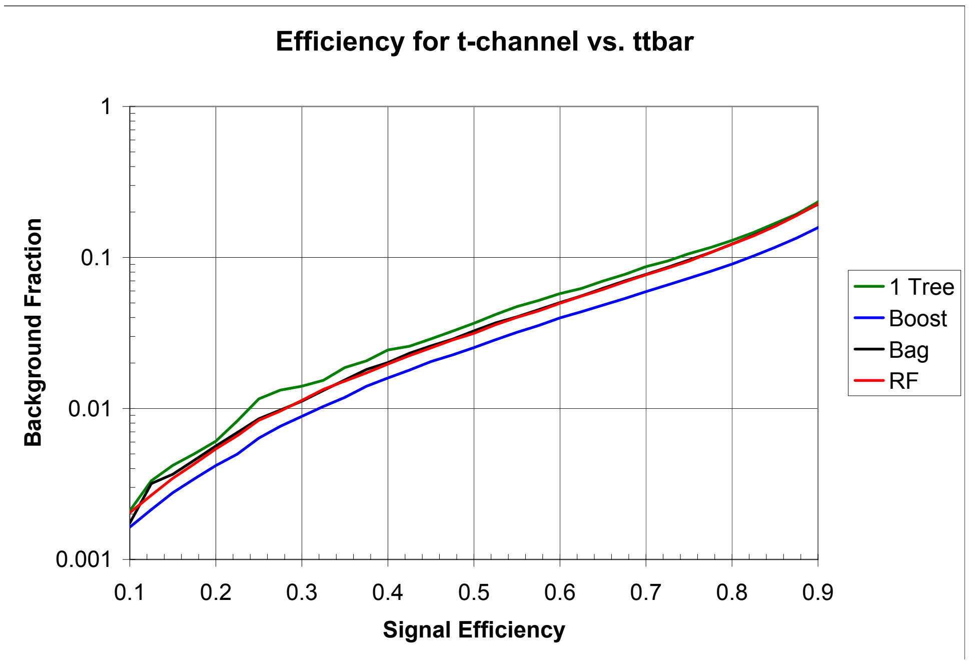

Figure 6: Mean of 10-fold generated trees' efficiency curves, for single trees (green), boosted trees (blue), bagged trees (black), and random forest (red) on t-channel versus $t\bar{t}$. The x-axis is the signal efficiency, while the y-axis is the log of the background fraction. Since we desire the greatest signal with as little background as possible, a classifier is better if its curve is to the lower right.

used, and the number of variables made available to the tree at each node of a random forest. These parameters were set to arbitrary values, and so further optimization might improve the performance of the other ensemble methods relative to boosting.

Bagging and random forest have almost identical performance. This occurs because there is not enough randomness being introduced into the variables available at each node of the random forest trees. If the number of variables available to the nodes in random forest were reduced, the performance would begin to diverge from that of bagged trees. It is also possible that some of the variables provided are correlated, so that removing one potential variable still leaves another that allows similar separation. Removing these would also allow the performance curves to diverge.

At first, it seems strange that although the output distributions for boosting are more central than for random forest and bagging, boosting has better performance. This is partially because the normalization of the histograms is misleading. While the peaks occur more central for boosted trees, the 'tails' of the decision tree output histogram drop off more abruptly than for the bagged trees or random forest. This is especially visible in Figure 4, where random forest and bagging have tails that are essentially flat, while boosting has tails that decrease linearly. So while the peaks are less central in bagging and random forest, they also occur where there is more of the tail present, resulting in an overall lower performance.

2.2.2 Stability

By looking at the signal efficiency as a function of purity cutoff, it is possible to gain an insight into the actual output of a tree. If an algorithm for generating the trees is stable, the signal efficiency for a given purity cutoff should not change when the input data is changed slightly. Figure 7 shows the difference from the mean of the signal efficiency for 10-fold trees grown using the different algorithms plotted versus purity cutoff. Looking at the plots, all algorithms are stable at low purity cutoffs. However, in the high purity cutoff region that we wish to use in our analysis, all algorithms begin to show instability due to the small number of events that score above this cutoff. The single tree is the least stable, while all of the ensemble methods show roughly the same magnitude of instability. However, random forest and, to a lesser extent, bagging, remain more stable than boosting to a higher purity cutoff.

3 Conclusions and Future Work

Based on the results summarized in the previous section, it seems as though boosting will be the most useful ensemble method for the single t quark search. While random forest shows promise for its stability, its overall performance is worse than boosting. There is hope, however, that tweaking and optimizing the parameters of random forest will bring its performance in line with boosting. This will require further investigation and testing.

The results seem to indicate that the decision trees optimize to best separate 1 b-tagged signal events from background. It, therefore, may be beneficial to create different trees for differing numbers of b-tags. While the number of b-tags is supplied to the training algorithms as a variable, the algorithms

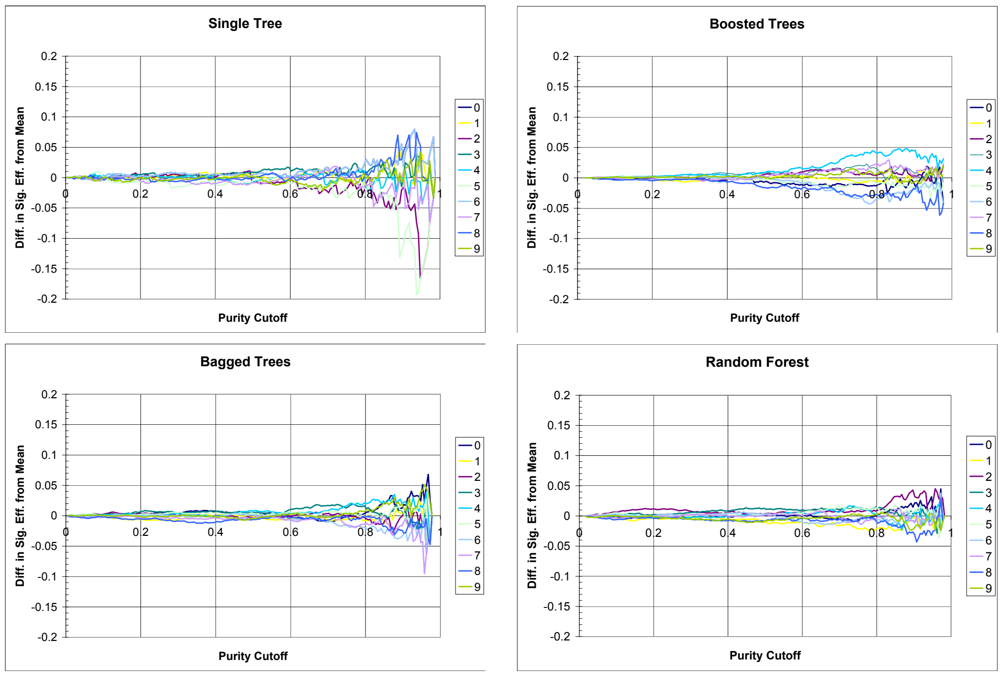

Figure 7: Difference from the mean of signal efficiency (% of signal events counted as signal) vs. purity cutoff (tree output value at which to separate signal from background) for 10-fold generated single trees (upper right), boosted trees (upper left), bagged trees (bottom right), random forest (bottom left). When the lines cluster close to 0, the algorithm is more stable.

do not choose to separate along it. This is a result of the 'greedy' nature of the training algorithms. Growing separate trees should be investigated. Additionally, it might be worthwhile to look for other variables that the tree is overlooking in a similar fashion..

There are other improvements to the training algorithms that can also be made. One such improvement, which could prove powerful, is 'pruning'. When a tree is grown, it does so in order to make predictions as well as possible on the training data. However, some sections of the tree may hold well for the training data, but are not useful for any other set of similar data, a condition known as 'overtraining'. Pruning attempts to remove these overtrained sections of the tree with a variety of different methods.

Finally, the signal/background separation is just one stage in the search for single t quark production. The next stage involves using the results of the decision trees, or of other classifiers, to try to estimate an upper limit on the cross section for single t quark production. Therefore, it would be worthwhile to investigate stability and performance in terms of the limits generated as a result of applying different algorithms.

4 Acknowledgments

The author wishes to acknowledge Gordon Watts for serving as his advisor, and for his advice and contributions to the decision tree code. Toby Burnett, Yann Coadou, and Dag Gillberg also contributed to much of the code used for this project.

The University of Washington in Seattle provided facilities and equipment for this research, which was supported by the National Science Foundation's Research Experience for Undergraduates program.

References

[1] Leo Breiman. "Random Forests". *Machine Learning*, 45:5–32, 2001.

[2] Abazov et. al. "Multivariate Searches for Single Top Quark Production with the DØ Detector". *Preprint*, 2006. [arXiv:hep-ex/0604020].

[3] David Opitz and Richard Maclin. "Popular Ensemble Methods: An Empirical Study". *Journal of Artificial Intelligence Research*, 11:169–198, 1999.