Oskari Saarimäki, University of Jyväskylä, Finland
Supervisor: Prof. Yoshitaka Kuno, Kuno Laboratory, Osaka University, Japan
August 5, 2016

Using Boosted Decision Trees in the COMET Experiment

Abstract

In my work, I have implemented a machine learning algorithm called Boosted Decision Trees (BDT) for the COMET experiment. The BDT reduces background hits in the cylindrical drift detector, which is used to detect the signal electron with a kinetic energy of 105 MeV. The algorithm is divided into a program which trains the BDT and a program which uses the trees that were trained. The BDT gives each of the hits a score which can be then used to discriminate signal hits from background hits. The efficiency curve is looking as expected and for example 90% signal retention gives 88% background rejection. The results were better than a single cut, but there is room for improvement also. For example adding a shape recognition would most likely give even better results.

1 Introduction

The objective of the COMET experiment is to discover the charged lepton flavour violation. The charged lepton flavour violation means that for example a coherent neutrinoless conversion of muons to electrons can happen. The idea of the method used in the COMET experiment is to create muonic atoms, and within a lifetime of a muonic atom the muon converts into an electron in the orbit

$$\mu^- + N(A,Z) \to e^- + N(A,Z).$$

The atom emits the electron with a kinetic energy of 105 MeV and this is used as the signal [1]. The detection will be made by a cylindrical detector, which consists of a Cylindrical Drift Chamber (CDC) and trigger hodoscopes. The CDC is used to reconstruct tracks of charged particles in a magnetic field. The momentum of a particle can be calculated from the curvature of the particle track, which can be seen from figure 1. The signal electron has somewhat specific properties, which differ from the background. But because the separation of signal hit and background hit properties is not perfect, a single cut for example from energy deposition alone, won't be good enough.

2 Methods

In my work, the objective is to remove background hits and retain signal hits in the CDC as well as possible. The method chosen is a machine learning algorithm called Boosted Decision Trees (BDT) [2]. The whole program and the algorithm is completely implemented by me, using C++. Histograms and data storing have been done using ROOT [3]. The idea is to create a single strong learner from a multiple of weak learners, namely decision trees. An example of a single decision tree is drawn in figure 2. As one can see from the figure 2, the tree commits a series of cuts for the group of hits. In the case of the example, there are three cuts made, namely energy deposition, time from trigger and layer ID. In my code, in addition of those mentioned in the example, I also use the average energy deposition of the closest neighbours in the same layer as the hit. The distributions for signal and background of these four properties can be seen in the figure 3. From each of the distributions it can be seen that signal hits behave differently compared to the background hits.

Speaking about single hits in the CDC might be first confusing as there are thousands of events forming from groups of hits. In my code all of the hits from the selected events get listed with the four properties described before. This single list of hits with different properties is then used in the BDT.

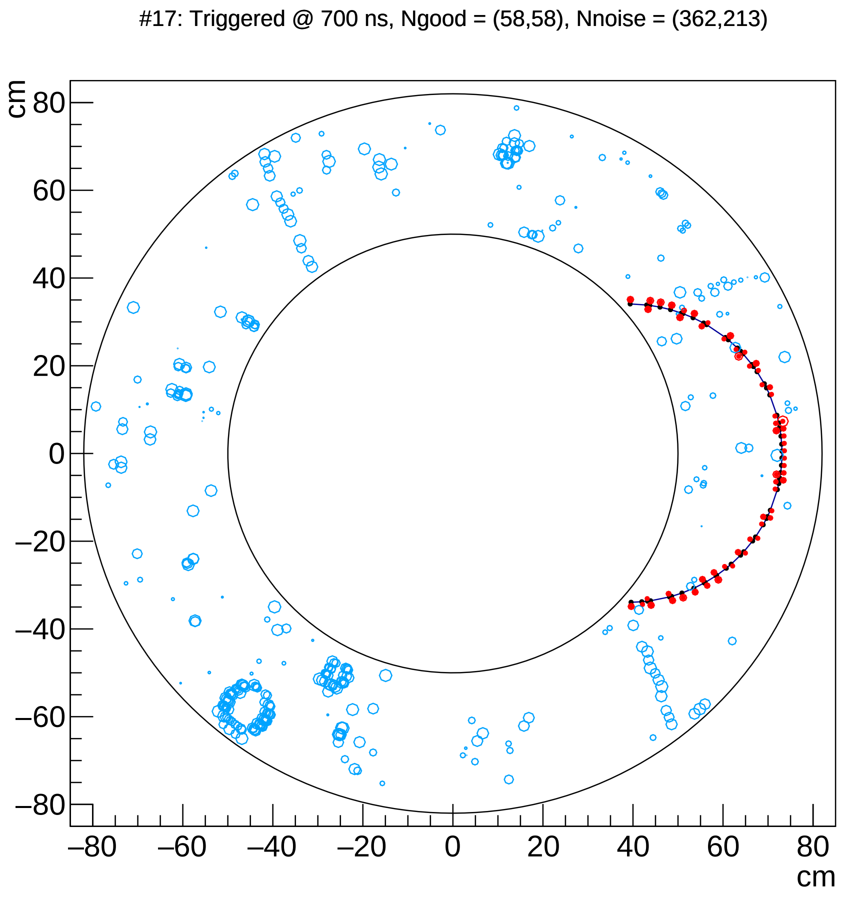


Figure 1: Example of a CDC event. Blue circles are background hits and red circles signal hits.

2.1 Creating a single Decision Tree

The trees are created using a control group of hits. This means that the program knows which control group hits are signal and which are background. The tree begins at a node, where all of the hits are included and in the beginning the weights of the hits are equal

$$W_i = \frac{1}{N_{\text{hits}}} \ \forall\, i.$$

This node will be split into two new branches. The program has to decide the optimal cut point for each of the comparisons and also choose the best comparison to use, as in my case there are four different choices. To decide

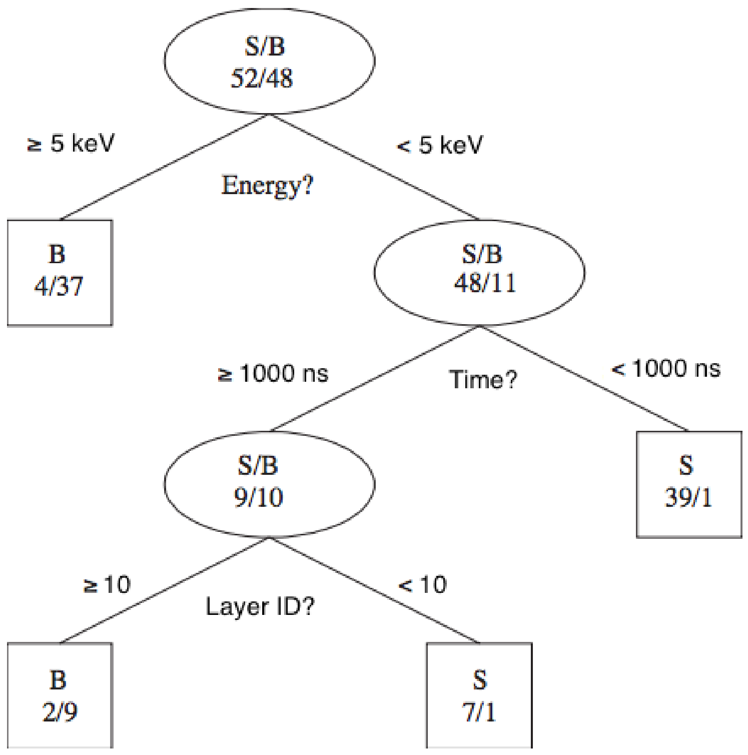


Figure 2: An example of a decision tree. Trees created by the algorithm can basically be of any shape and cut points also vary. S means signal and B background. The original figure is created by Byron P. Roe et al. [2] and edited by me.

which cut is the best, a purity of a node or a branch is defined as

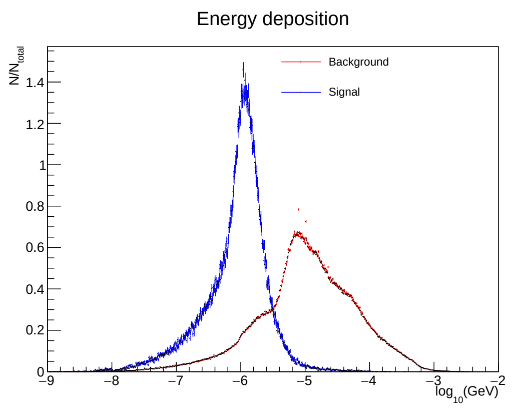
$$P = \frac{\sum_s W_s}{\sum_s W_s + \sum_b W_b}, \tag{1}$$

where $\sum_s W_s$ is the sum of weights for all the signal hits and $\sum_b W_b$ is the sum of weights for all the background hits. From the equation 1, Gini is defined as

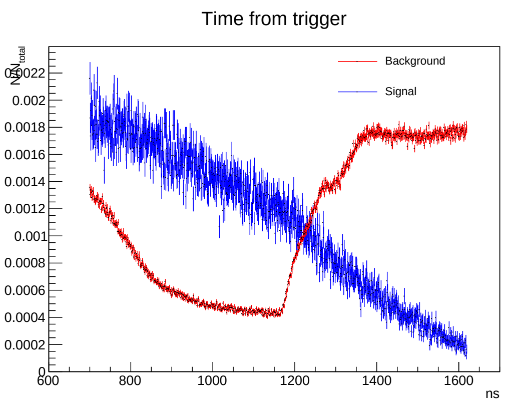
$$\text{Gini} = \left(\sum_{i=1}^{n} W_i\right) P(1-P). \tag{2}$$

Note that the n in this equation is the number of hits in that particular branch or node, as it changes depending on where in the tree is Gini being calculated. Then, for a cut is defined

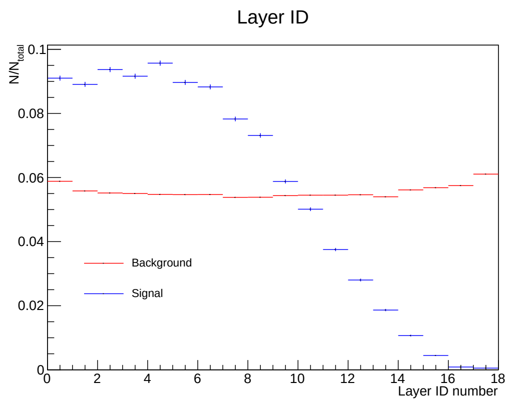
$$\text{Criterion} = \text{Gini}_{\text{father}} - \text{Gini}_{\text{left son}} - \text{Gini}_{\text{right son}}, \tag{3}$$



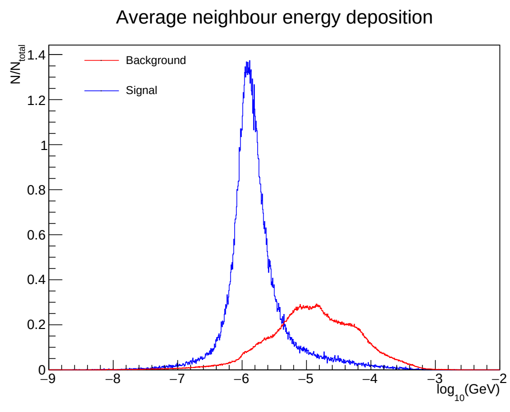
(a) Energy deposition distribution



(b) Distribution of time from trigger



(c) Layer ID distribution



(d) Avg. neighbour energy deposition

Figure 3: Histograms for signal and background. These four properties were used in my algorithm. It is well seen here that the signal behaves little bit differently than background.

where father is the branch which is being split and sons are the two remaining branches. When deciding between many cuts, maximising Criterion gives the best separating cut.

Before choosing the best comparison for the first node, the best cut point for each of the comparisons have to be decided. This is good to do, because the optimal cut point changes between different trees and branches. The Criterion will be calculated for each of the points in that histogram and the point with biggest Criterion will be chosen as the optimal cut point. This optimal cut point will be calculated for each of the comparisons. Then the best Criterions of each comparison will be compared and the comparison with the biggest Criterion will be chosen as the initial comparison.

Now that the first cut has been made, two branches have been created. As Gini describes how well a group of hits are separated in a branch, I decided that the next comparison will be always asked from a branch with the biggest Gini. This means that the next cut will be made for a branch with the worst separation. After the next branch has been decided, all of the remaining comparisons will again be tested for the best cut point and out of those, the best comparison will be chosen in a similar manner as before. After the second cut there are three branches available. The next comparison will be then asked from the branch with the biggest Gini, in the same way as before. After all the comparisons have been asked once, the tree is ready. All the ending point branches are now leafs. Leafs with purity bigger than 0.5 will be labeled as signal leafs and leafs with purity less than 0.5 will be labeled as background leafs. Example of this can be seen in the figure 2. It is good to note that there will almost always be false positives and false negatives in these kinds of trees.

2.2 Boosting

After creating a decision tree comes the boosting part of the algorithm. I have used the Adaptive Boost, or AdaBoost for short. For each of the trees, there is defined a function

$$\mathrm{err}_m = \frac{\sum_{i=1}^{N} W_i I(y_i \neq T_m(x_i))}{\sum_{i=1}^{N} W_i}, \tag{4}$$

where N is the total amount of hits, x_i is ith hit and

$$y_i = \begin{cases} 1 & \text{if } i\text{th hit is signal} \\ -1 & \text{if } i\text{th hit is background} \end{cases} \tag{5}$$

$$T_m(x_i) = \begin{cases} 1 & \text{if } i\text{th hit is labeled as signal by the } m\text{th tree} \\ -1 & \text{if } i\text{th hit is labeled as noise by the } m\text{th tree} \end{cases} \tag{6}$$

$$I(y_i \neq h_m(x_i)) = \begin{cases} 1 & \text{if } y_i \neq h_m(x_i) \\ -1 & \text{if } y_i = h_m(x_i). \end{cases} \tag{7}$$

This means that if a tree labels almost all of the hits correctly, equation 4 will be near 0, and if a tree fails to label the hits correctly, equation 4 will get closer and closer to 1.

From equation 4 is defined

$$\alpha_m = \beta \times \ln\left(\frac{1 - \mathrm{err}_m}{\mathrm{err}_m}\right), \tag{8}$$

where $\beta = 1$ in my case. Some different methods use different values for the constant β. Equation 8 could be dubbed as 'the betterness equation' as it effectively describes how well the mth tree is separating the given group of hits. If the tree is good at separation, $\text{err}_m \to 0$ and α_m becomes a big number. If the tree is not separating the hits at all, $\text{err}_m = 0.5$ where $\alpha_m = 0$. This means that the tree did nothing useful. Also it could be possible that a tree could be labelling the signal as background and background as signal, and in this case $\text{err}_m \to 1$ and α_m becomes a big negative number. This is a harmful situation to the whole BDT, so usually the run is terminated if a case of $\text{err}_m > 0.5$ happens. This is also true for my algorithm.

Using equations 7 and 8, the weight of the hits are changed

$$W_i \to W_i \times e^{\alpha_m I(y_i \neq T_m(x_i))} \;\; \forall\, i. \tag{9}$$

What this basically means, is that the weight of the hits that were labeled wrong by the mth tree will be increased, so in the next tree they will get a bigger attention, and in the other hand correctly labeled hits get no change to their weight. It is good to notice that if the $\text{err}_m > 0.5$ then the weights would actually shrink, which is the opposite of what is meant to do. After this, the weights are again normalised

$$W_i \to \frac{W_i}{\sum_{i=1}^{N} W_i} \;\; \forall\, i.$$

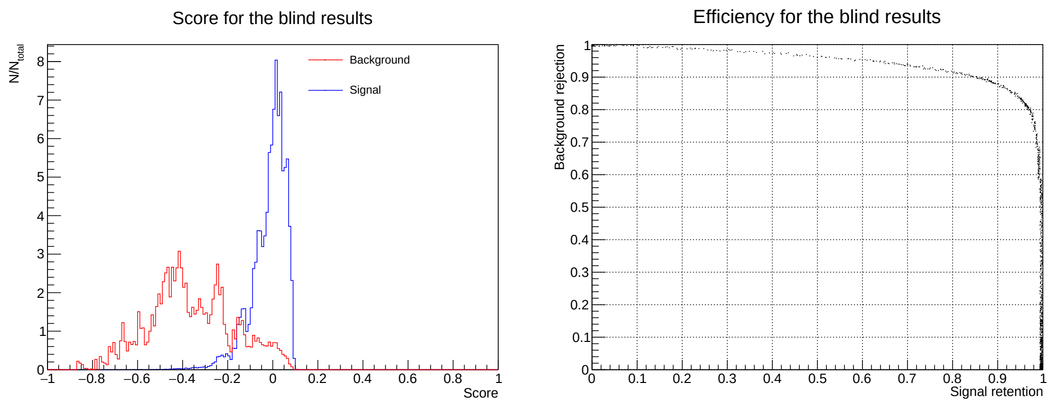
As the weights are different in the next round, the tree will also most probably be different and yield different results. The weights of the hits are quite important because they are being used in all stages of this algorithm. Next the second tree gives new weights and a third one is created and so on. This way hundreds or thousands of trees are created.

2.3 Score

When all of the hundreds of trees have been created by the control group of known hits, the created trees can be used also for unknown hits. All of the hits go through all of the trees and each hit gets a score

$$T(x_i) = \frac{\sum_{m=1}^{N_{\text{trees}}} \alpha_m T_m(x_i)}{\sum_{m=1}^{N_{\text{trees}}} \alpha_m}. \tag{10}$$

This score $T(x_i) \in [-1, 1]$. This basically means that a hit that has been labeled mostly as signal gets a bigger score and vice versa.



(a) Score distribution

(b) Efficiency curve

Figure 4: Score and efficiency for 5251 events with 301 trees.

3 Results

The score histogram for unknown hits can be seen in the figure 4a. The picture shows that the signal and background has separated, but not perfectly. This is expected. The reason why the signal hits are gathered around 0 and not near 1 might come from the fact that signal is really underrepresented in the data compared to the background.

The efficiency curve in figure 4b has been calculated from each point of the score histogram. From this curve it can be decided what kind of efficiency is the best in this case. Figure 4b shows that for example a 90% signal retention would result in a 88% background rejection, and vice versa, but many other choices can be also made. When handling real data, it cannot be discriminated which hit is background and which is signal in the same way as in figure 4a. In a real situation the best cut point for the score can be estimated according to the efficiency curve.

4 Summary

From the results of this algorithm, it is fairly safe to say that the BDT works for experiments like COMET. The results give a definite improvement over a simple single cut or a single decision tree. Still, from figure 4a it can be seen, that there still is room for improvement. Also the speed of the tree training algorithm is relatively slow, but it was never my intention to create the fastest algorithm in the first place. On the other hand, using the created trees is much faster.

I strongly believe that if added, a shape recognition for the signal track would probably increase efficiency. From figure 1 it can be seen that the signal electron has a distinct curvature which could be used to remove additional noise. I also want to point out for those interested that there are a couple of very capable machine learning libraries which can be used to create BDT for many purposes, for example scikit-learn for python and TMVA for ROOT.

References

[1] The COMET Collaboration,*COMET Phase-1 Technical Design Report*, July 2016

[2] Byron P. Roe and Hai-Jun Yang and Ji Zhu and Yong Liu and Ion Stancu and Gordon McGregor, *Boosted decision trees as an alternative to artificial neural networks for particle identification* Nuclear Instruments and Methods in Physics Research, vol. 543 2-3, doi:10.1016/j.nima.2004.12.018, 2005

[3] I. Antcheva, M. Ballintijn, B. Bellenot, M. Biskup, R. Brun, N. Buncic, Ph. Canal, D. Casadei, O. Couet, V. Fine et al., *ROOT: A C++ framework for petabyte data storage, statistical analysis and visualization*, Comput.Phys.Commun. 180 2499-2512 doi:10.1016/j.cpc.2009.08.005 arXiv:1508.07749, 2009