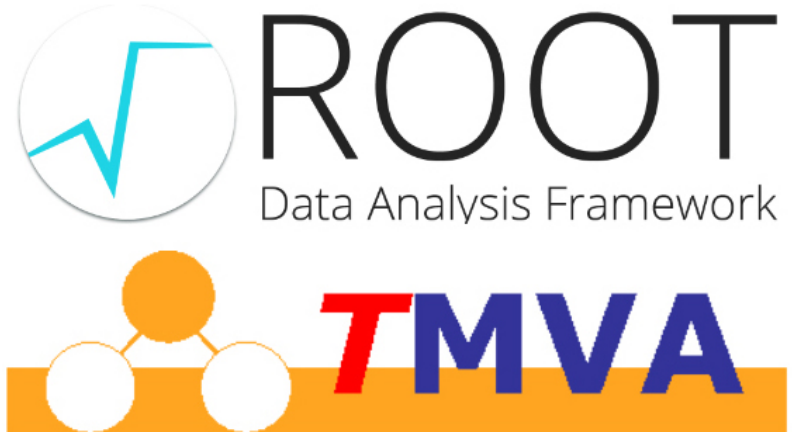
PresentationInfo

Username	Andrew Schick
Event Name	TMVA Hands On Tutorial
Date	February 19, 2025

A Practical Tutorial for Using TMVA in Hall D



Goal: Use ROOTs TMVA to train a Boosted Decision Tree (BDT) to do a basic binary classification (in our case e^-/π^- separation)

Prerequisites

- Access to the ifarm
- a speedy way to display graphics made by ROOT, preferably a VDI connection using VMWare, or to rsync the output ROOT files locally

Overview of Procedure for a TMVA analysis

- Produce ROOT trees for your reaction of interest using the reaction filter plugin (in our case, $\gamma p \rightarrow e^+e^-(p)$)
- Use a DSelector to produce a flat ROOT tree containing the same information + branches to contain the model inference
- Load the ROOT trees into TMVAClassification.C and train the model
- Classify real data with TMVAClassificationApplication.C and add the inference as a branch to your ROOT tree
- Analyze the flat ROOT tree with the inferences added in a TSelector or MakeClass.C plugin (or interactively!)

Copy the Files to a Directory You Own

- cp /work/halld/home/acschick/AIworkshop2025_TMVA/*.* .

```
ifarm2402.jlab.org> ls -1d $PWD/*.*
/work/halld/home/acschick/AIworkshop2025_TMVA/DSelector_2eMissingProton.C
/work/halld/home/acschick/AIworkshop2025_TMVA/DSelector_2eMissingProton.h
/work/halld/home/acschick/AIworkshop2025_TMVA/TMVAClassificationApplicationMINUS.C
/work/halld/home/acschick/AIworkshop2025_TMVA/TMVAClassificationMINUS.C
ifarm2402.jlab.org>
```

- TMVAClassification.C → Training of your models
- TMVAClassificationApplication → Inference on your data
- DSelector → Produce flat trees to classify or be used in training (done for you)

TMVAClassification.C

1. Select which models to train

```
Use["HMatrix"]         = 0;
//
// Function Discriminant Analysis
Use["FDA_GA"]          = 0; // minimisation of user-defined function using Genetics Algorithm
Use["FDA_SA"]          = 0;
Use["FDA_MC"]          = 0;
Use["FDA_MT"]          = 0;
Use["FDA_GAMT"]        = 0;
Use["FDA_MCMT"]        = 0;
//
// Neural Networks (all are feed-forward Multilayer Perceptrons)
Use["MLP"]             = 1; // Recommended ANN
Use["MLPBFGS"]         = 0; // Recommended ANN with optional training method
Use["MLPBNN"]          = 0; // Recommended ANN with BFGS training method and bayesian regulator
Use["CFMlpANN"]        = 0; // Depreciated ANN from ALEPH
Use["TMlpANN"]         = 0; // ROOT's own ANN
Use["DNN"]             = 0; // Deep Neural Network
Use["DNN_GPU"]         = 0; // CUDA-accelerated DNN training.
Use["DNN_CPU"]         = 0; // Multi-core accelerated DNN.
//
// Support Vector Machine
Use["SVM"]             = 0;
//
// Boosted Decision Trees
Use["BDT"]             = 1; // uses Adaptive Boost
Use["BDTG"]            = 0; // uses Gradient Boost
Use["BDTB"]            = 0; // uses Bagging
Use["BDTD"]            = 0; // decorrelation + Adaptive Boost
Use["BDTF"]            = 0; // allow usage of fisher discriminant for node splitting
//
// Friedman's RuleFit method, ie, an optimised series of cuts ("rules")
Use["RuleFit"]         = 0;
// ---------------------------------------------------------------

std::cout << std::endl;
std::cout << "==> Start TMVAClassification" << std::endl;
```

TMVAClassification.C

1. Select which models to train
2. Specify your signal and background (the two classes for classification) ROOT files. Open them, and then get the ROOT trees.

```
// ------------------------------------------------------------------------------------
// Read training and test data
// (it is also possible to use ASCII format as input -> see TMVA Users Guide)
TString fname = "/work/halld/home/acschick/AIworkshop2025_TMVA/trainingSamples/Flat_eeRBHG_train_RADON_Anaprop_corAVGALL.root";
TString background_fname = "/work/halld/home/acschick/AIworkshop2025_TMVA/trainingSamples/Flat_rho0p_2018-01_BASE_TrMCSIZE.root";

//this is just a fall back thing.
//if (gSystem->AccessPathName( fname ))  // file does not exist in local directory
 //  gSystem->Exec("curl -O http://root.cern.ch/files/tmva_class_example.root");

TFile *input = TFile::Open( fname );
TFile *backgroundInput = TFile::Open( background_fname );

std::cout << "--- TMVAClassification       : Using input file: " << input->GetName() << std::endl;

// Register the training and test trees
TTree *background = (TTree*)backgroundInput->Get("FLAT_pip_TMVA");
TTree *signalTree = (TTree*)input->Get("Flat2018_MC_Electrons");
```

Whatever the names of your trees are in your signal and background files

TMVAClassification.C

1. Select which models to train
2. Specify your signal and background (the two classes for classification) ROOT files.
   Open them, and then get the ROOT trees.
3. Specify your training features and
   “spectator” variables.

AddVariable for features used in training

AddSpectator for any variables you need to do your physics analysis after model inference, but don’t need in your model.

```
// Define the input variables that shall be used for the MVA training
// note that you may also use variable expressions, such as: "3*var1/var2*abs(var3)"
// [all types of expressions that can also be parsed by TTree::Draw( "expression" )]

dataloader->AddVariable( "EoverP_minus", "Electron FCAL Energy/KinFit momentum magnitude", 'F');
dataloader->AddVariable( "FCAL_DOCA_em", "Electron FCAL DOCA", 'F');
dataloader->AddVariable( "FCAL_E9E25_em", "Electron E9/E25 shower ratio" , 'F' );

// You can add so-called "Spectator variables", which are not used in the MVA training,
// but will appear in the final "TestTree" produced by TMVA. This TestTree will contain the
// input variables, the response values of all trained MVAs, and the spectator variables

//dataloader->AddSpectator( "spec1 := var1*2",  "Spectator 1", "units", 'F' );
//dataloader->AddSpectator( "spec2 := var1*3",  "Spectator 2", "units", 'F' );

dataloader->AddSpectator("RFTime_kin","RFTime_kin");
dataloader->AddSpectator("RFTime_meas","RFTime_meas");
dataloader->AddSpectator("BeamX4_kin_T","BeamX4_kin_T");
dataloader->AddSpectator("BeamX4_kin_X","BeamX4_kin_X");
dataloader->AddSpectator("BeamX4_kin_Y","BeamX4_kin_Y");
dataloader->AddSpectator("BeamX4_kin_Z","BeamX4_kin_Z");
dataloader->AddSpectator("BeamX4_meas_T","BeamX4_meas_T");
dataloader->AddSpectator("BeamX4_meas_X","BeamX4_meas_X");
dataloader->AddSpectator("BeamX4_meas_Y","BeamX4_meas_Y");
dataloader->AddSpectator("BeamX4_meas_Z","BeamX4_meas_Z");
dataloader->AddSpectator("BeamP4_kin_E","BeamP4_kin_E");
dataloader->AddSpectator("BeamP4_kin_X","BeamP4_kin_X");
dataloader->AddSpectator("BeamP4_kin_Y","BeamP4_kin_Y");
dataloader->AddSpectator("BeamP4_kin_Z","BeamP4_kin_Z");
dataloader->AddSpectator("BeamP4_meas_E","BeamP4_meas_E");
dataloader->AddSpectator("BeamP4_meas_X","BeamP4_meas_X");
dataloader->AddSpectator("BeamP4_meas_Y","BeamP4_meas_Y");
dataloader->AddSpectator("BeamP4_meas_Z","BeamP4_meas_Z");
dataloader->AddSpectator("TargetCenter_Z","TargetCenter_Z");
dataloader->AddSpectator("TargetCenter_X","TargetCenter_X");
dataloader->AddSpectator("TargetCenter_Y","TargetCenter_Y");
dataloader->AddSpectator("kinfit_chisq","kinfit_chisq");
dataloader->AddSpectator("FCAL_Energy_ep","FCAL_Energy_ep");
dataloader->AddSpectator("TOF_dEdx_ep","TOF_dEdx_ep");
dataloader->AddSpectator("FDC_dEdx_ep","FDC_dEdx_ep");
dataloader->AddSpectator("ep_p_kin_E","ep_p_kin_E");
```

What are Spectator Variables?

- The ROOT tree that used in training must have all the same branches as the data ROOT tree you will classify
- The **spectator variables** are every variable you need to perform your analysis after any cuts you apply based on the model inference.
- Simple approach: write all the original branches in the ROOT tree produced by the reaction filter plugin into your flat ROOT tree

Making Flat ROOT Trees with the DSelector

```
//USERS: SET OUTPUT FILE NAME //can be overriden by user in PROOF
dOutputFileName = "Hist_2018-01_0pol.root"; //"" for none
dOutputTreeFileName = "";//"ee_MVA_7-20-2020_ComboTrees.root"; //"" for none
dFlatTreeFileName = "Flat_2018-01_0pol.root"; //output flat tree (one combo per tree entry), "" for none
dFlatTreeName = "Flat2018_0pol_Electrons"; //if blank, default name will be chosen
```

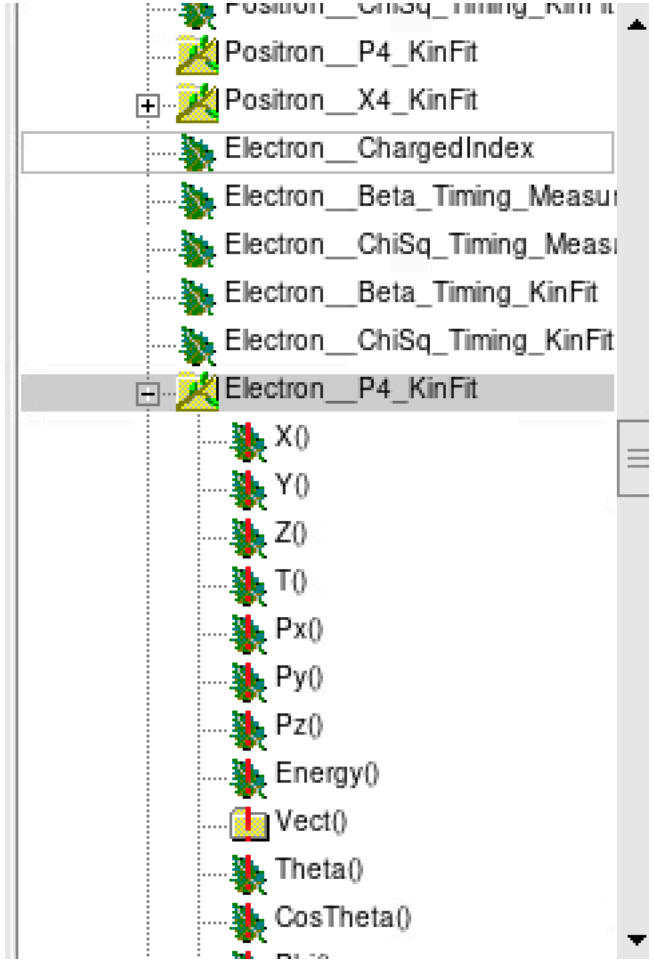
```
dFlatTreeInterface->Create_Branch_Fundamental<Float_t>("BeamP4_meas_Z");

dFlatTreeInterface->Create_Branch_Fundamental<Float_t>("TargetCenter_Z");
dFlatTreeInterface->Create_Branch_Fundamental<Float_t>("TargetCenter_X");
dFlatTreeInterface->Create_Branch_Fundamental<Float_t>("TargetCenter_Y");
dFlatTreeInterface->Create_Branch_Fundamental<Float_t>("kinfit_chisq");

dFlatTreeInterface->Create_Branch_Fundamental<Float_t>("FCAL_Energy_ep");
dFlatTreeInterface->Create_Branch_Fundamental<Float_t>("TOF_dEdx_ep");
dFlatTreeInterface->Create_Branch_Fundamental<Float_t>("FDC_dEdx_ep");

dFlatTreeInterface->Create_Branch_Fundamental<Float_t>("ep_p_kin_E");
dFlatTreeInterface->Create_Branch_Fundamental<Float_t>("ep_p_kin_X");
dFlatTreeInterface->Create_Branch_Fundamental<Float_t>("ep_p_kin_Y");
dFlatTreeInterface->Create_Branch_Fundamental<Float_t>("ep_p_kin_Z");
dFlatTreeInterface->Create_Branch_Fundamental<Float_t>("ep_p_meas_E");
dFlatTreeInterface->Create_Branch_Fundamental<Float_t>("ep_p_meas_X");
dFlatTreeInterface->Create_Branch_Fundamental<Float_t>("ep_p_meas_Y");
dFlatTreeInterface->Create_Branch_Fundamental<Float_t>("ep_p_meas_Z");

dFlatTreeInterface->Create_Branch_Fundamental<Float_t>("FCAL_Energy_em");
dFlatTreeInterface->Create_Branch_Fundamental<Float_t>("TOF_dEdx_em");
dFlatTreeInterface->Create_Branch_Fundamental<Float_t>("FDC_dEdx_em");
```



TMVA doesn't know how to handle the "combo" structure of the original ROOT file

The data type of the data must be floats.

Customizing Your Models

Specify number of events to use in training

select which events are used in training and in validation randomly

```
dataloader->PrepareTrainingAndTestTree( mycut,
                                        "nTrain_Signal=20000:nTrain_Background=20000:SplitMode=Random:NormMode=NumEvents:!V" );
```

use all remaining events are validation

Specify model architecture

```
   // TMVA ANN: MLP (recommended ANN) -- all ANNs in TMVA are Multilayer Perceptrons
   if (Use["MLP"])
      factory->BookMethod( dataloader, TMVA::Types::kMLP, "MLP", "H:!V:NeuronType=tanh:VarTransform=N:NCycles=600:HiddenLayers=N+5:TestRate=5:!UseRegulator" );


if (Use["BDT"])  // Adaptive Boost
   factory->BookMethod( dataloader, TMVA::Types::kBDT, "BDT",
                        "!H:!V:NTrees=850:MinNodeSize=2.5%:MaxDepth=3:BoostType=AdaBoost:AdaBoostBeta=0.5:UseBaggedBoost:BaggedSampleFraction=0.5:SeparationType=G
Index:nCuts=20" );

if (Use["BDTB"]) // Bagging
```

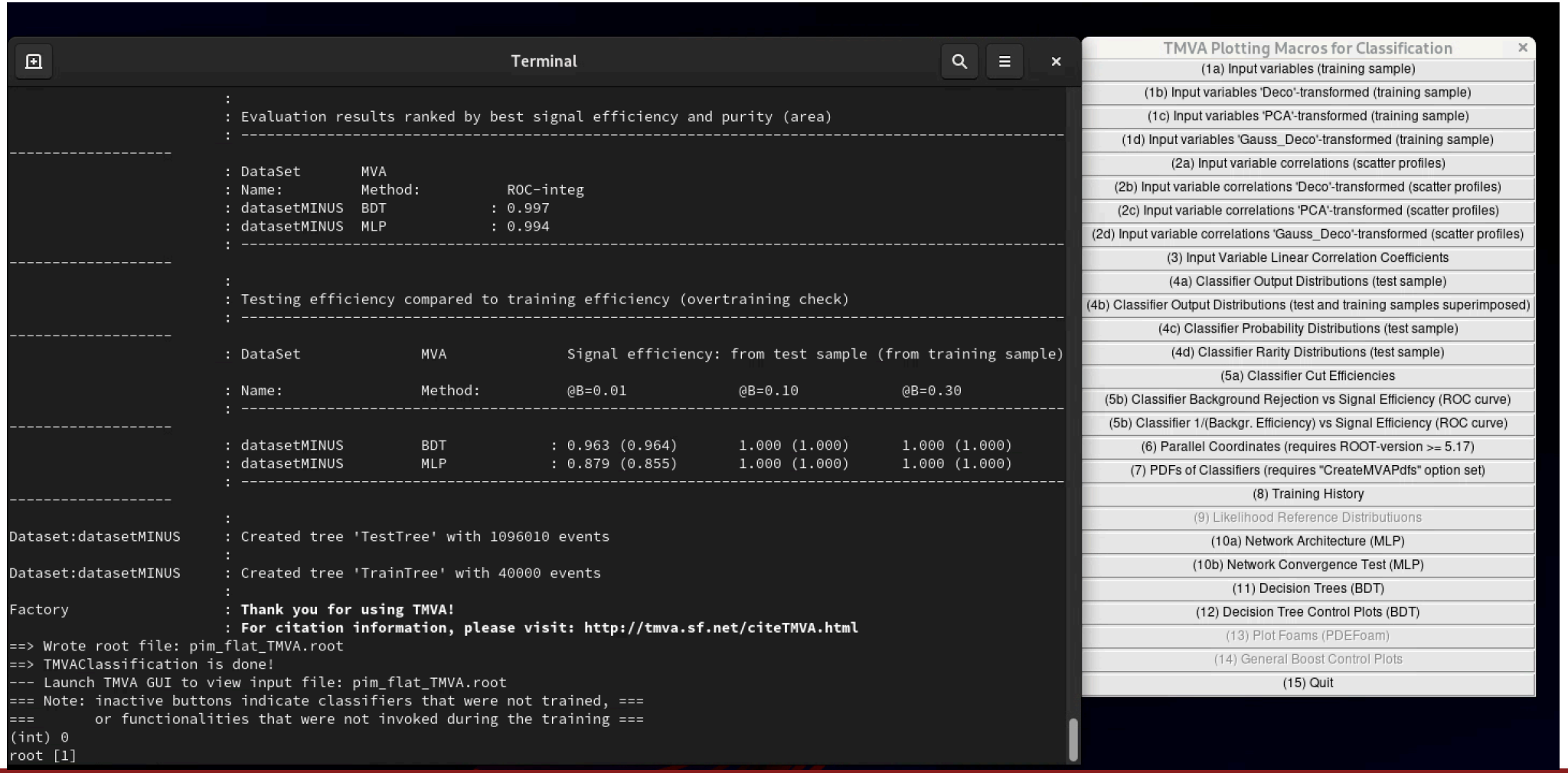
Train your model by running your TMVAClassification.C macro

```
ifarm2402.jlab.org> root -l TMVAClassificationMINUS.C
```

Text output from training will rank usefulness of training variables, and rank models by their performance

```
Factory                  : Training finished
                         :
                         : Ranking input variables (method specific)...
MLP                      : Ranking result (top variable is best ranked)
                         : ----------------------------------------
                         : Rank : Variable      : Importance
                         : ----------------------------------------
                         :    1 : EoverP_minus  : 1.082e+03
                         :    2 : FCAL_DOCA_em  : 2.705e+02
                         :    3 : FCAL_E9E25_em : 1.333e+01
                         : ----------------------------------------
BDT                      : Ranking result (top variable is best ranked)
                         : ------------------------------------------------
                         : Rank : Variable      : Variable Importance
                         : ------------------------------------------------
                         :    1 : FCAL_E9E25_em : 3.506e-01
                         :    2 : EoverP_minus  : 3.286e-01
                         :    3 : FCAL_DOCA_em  : 3.208e-01
                         : ------------------------------------------------
                       :
                       : Evaluation results ranked by best signal efficiency and purity (area)
                       : ------------------------------------------------------------------------------------
------------------
                       : DataSet       MVA
                       : Name:         Method:          ROC-integ
                       : datasetMINUS  BDT            : 0.997
                       : datasetMINUS  MLP            : 0.994
                       : ------------------------------------------------------------------------------------
------------------
```

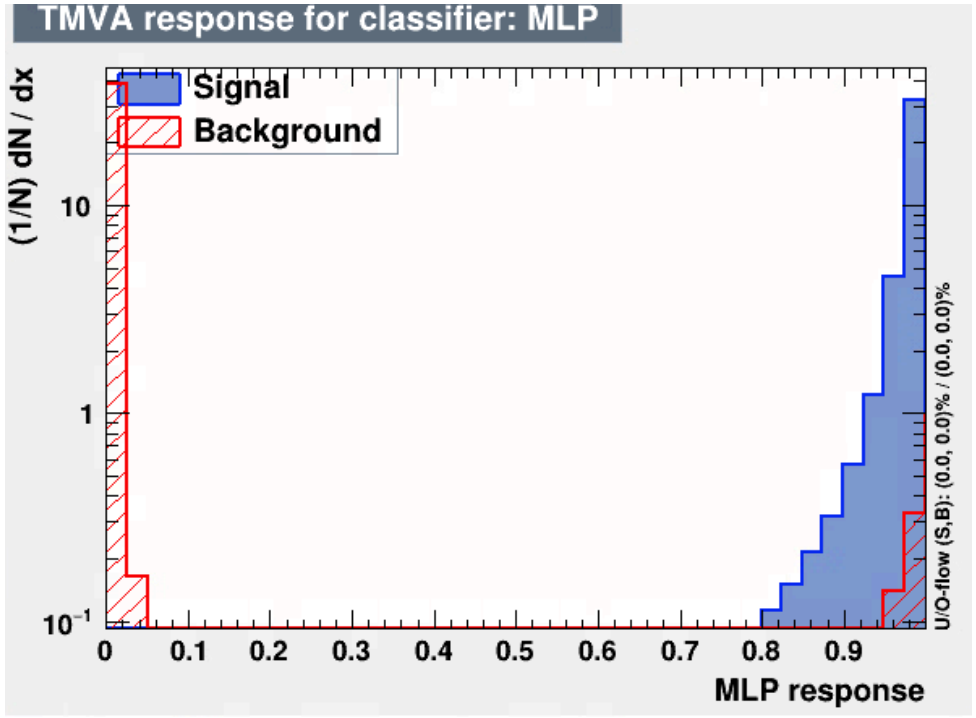
After training completes, a GUI will appear allowing you to explore the results from training and validation.
Almost all the plots here are interesting, but it's worth pointing out specifically:
Classifier Output Distributions, Classifier Cut Efficiencies, ROC curve

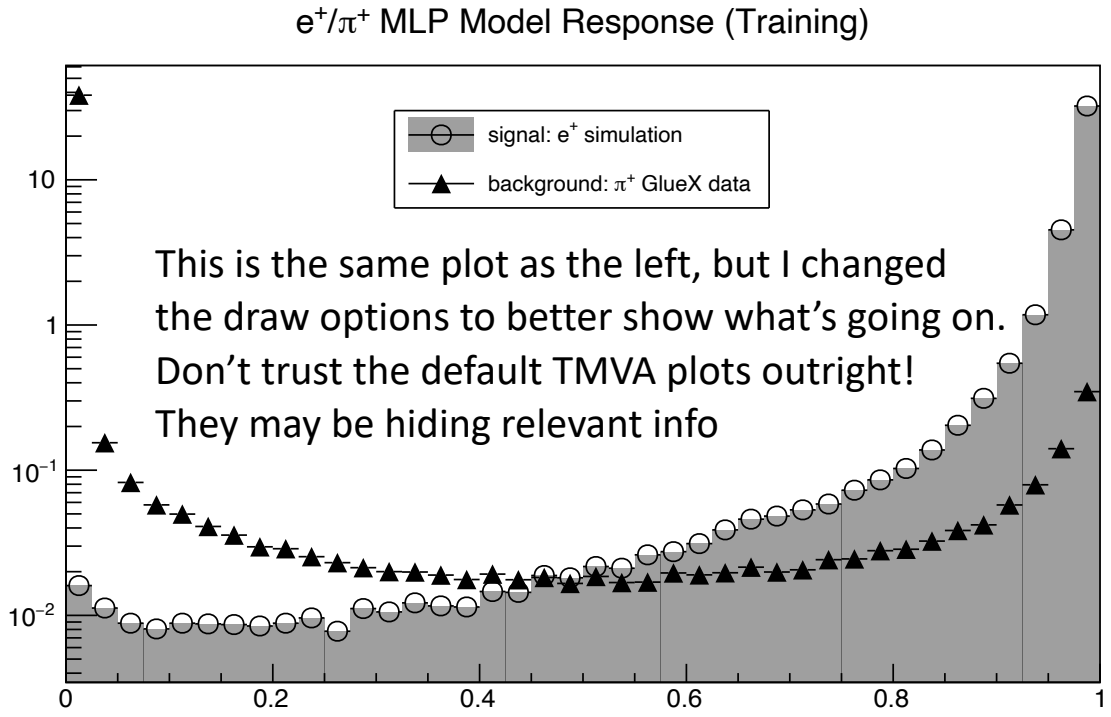


You can reopen the training file and see all these plots with

```
TMVA::TMVAGui( outfileName );
```

You can also explore all the results from training with a TBrowser(), and customize your own histograms.
Some of the default plots TMVA makes lack the ability to manipulate, e.g. the Y axis and hides important info.





TMVAClassificationApplication.C

Set what inferences you will perform

```
//
// Linear Discriminant Analysis
Use["LD"]              = 0; // Linear Discriminant identical to Fisher
Use["Fisher"]          = 0;
Use["FisherG"]         = 0;
Use["BoostedFisher"]   = 0; // uses generalised MVA method boosting
Use["HMatrix"]         = 0;
//
// Function Discriminant analysis
Use["FDA_GA"]          = 0; // minimisation of user-defined function using Genetics Algorithm
Use["FDA_SA"]          = 0;
Use["FDA_MC"]          = 0;
Use["FDA_MT"]          = 0;
Use["FDA_GAMT"]        = 0;
Use["FDA_MCMT"]        = 0;
//
// Neural Networks (all are feed-forward Multilayer Perceptrons)
Use["MLP"]             = 1; // Recommended ANN
Use["MLPBFGS"]         = 0; // Recommended ANN with optional training method
Use["MLPBNN"]          = 0; // Recommended ANN with BFGS training method and bayesian regulator
Use["CFMlpANN"]        = 0; // Depreciated ANN from ALEPH
Use["TMlpANN"]         = 0; // ROOT's own ANN
Use["DNN"]             = 0; // improved implementation of a NN
//
// Support Vector Machine
Use["SVM"]             = 0;
//
// Boosted Decision Trees
Use["BDT"]             = 1; // uses Adaptive Boost
Use["BDTG"]            = 0; // uses Gradient Boost
Use["BDTB"]            = 0; // uses Bagging
Use["BDTD"]            = 0; // decorrelation + Adaptive Boost
Use["BDTF"]            = 0; // allow usage of fisher discriminant for node splitting
//
```

TMVAClassificationApplication.C

List all your variables that you want in the output file for your analysis

Load in the variables that you will actually perform the inference with, and then the spectator variables

```
TMVA::Reader *reader = new TMVA::Reader( "!Color:!Silent" );

// Create a set of variables and declare them to the reader
// - the variable names MUST corresponds in name and type to those given in the weight file(s) used
Float_t RFTime_kin,RFTime_meas,BeamX4_kin_T,BeamX4_kin_X,BeamX4_kin_Y,BeamX4_kin_Z,BeamX4_meas_T,
  BeamX4_meas_X,BeamX4_meas_Y,BeamX4_meas_Z,BeamP4_kin_E,BeamP4_kin_X,BeamP4_kin_Y,BeamP4_kin_Z,
  BeamP4_meas_E,BeamP4_meas_X,BeamP4_meas_Y,BeamP4_meas_Z,TargetCenter_Z,TargetCenter_X,
  TargetCenter_Y,kinfit_chisq,FCAL_Energy_ep,TOF_dEdx_ep,FDC_dEdx_ep,ep_p_kin_E,ep_p_kin_X,ep_p_kin_Y,
  ep_p_kin_Z,ep_p_meas_E,ep_p_meas_X,ep_p_meas_Y,ep_p_meas_Z,FCAL_Energy_em,TOF_dEdx_em,FDC_dEdx_em,em_p_kin_E,
  em_p_kin_X,em_p_kin_Y,em_p_kin_Z,em_p_meas_E,em_p_meas_X,em_p_meas_Y,em_p_meas_Z,g_p_kin_E,g_p_kin_X,g_p_kin_Y,g_p_kin_Z,
  g_p_meas_E,g_p_meas_X,g_p_meas_Y,g_p_meas_Z,DecayingPi0_p_kin_E,DecayingPi0_p_kin_X,DecayingPi0_p_kin_Y,DecayingPi0_p_kin_Z,
  DecayingPi0_p_meas_E,DecayingPi0_p_meas_X,DecayingPi0_p_meas_Y,DecayingPi0_p_meas_Z,Energy_UnusedShowers,EoverP_plus,FCAL_DOCA_ep,FCAL_E9E25_ep,ep_phi,EoverP_mi\
nus,
  FCAL_DOCA_em,FCAL_E9E25_em,em_phi;

reader->AddVariable( "EoverP_minus", &EoverP_minus );
reader->AddVariable( "FCAL_DOCA_em", &FCAL_DOCA_em );
reader->AddVariable( "FCAL_E9E25_em", &FCAL_E9E25_em );
```

```
reader->AddSpectator("RFTime_kin",&RFTime_kin);
reader->AddSpectator("RFTime_meas",&RFTime_meas);
reader->AddSpectator("BeamX4_kin_T",&BeamX4_kin_T);
reader->AddSpectator("BeamX4_kin_X",&BeamX4_kin_X);
reader->AddSpectator("BeamX4_kin_Y",&BeamX4_kin_Y);
reader->AddSpectator("BeamX4_kin_Z",&BeamX4_kin_Z);
reader->AddSpectator("BeamX4_meas_T",&BeamX4_meas_T);
reader->AddSpectator("BeamX4_meas_X",&BeamX4_meas_X);
reader->AddSpectator("BeamX4_meas_Y",&BeamX4_meas_Y);
reader->AddSpectator("BeamX4_meas_Z",&BeamX4_meas_Z);
reader->AddSpectator("BeamP4_kin_E",&BeamP4_kin_E);
reader->AddSpectator("BeamP4_kin_X",&BeamP4_kin_X);
reader->AddSpectator("BeamP4_kin_Y",&BeamP4_kin_Y);
reader->AddSpectator("BeamP4_kin_Z",&BeamP4_kin_Z);
reader->AddSpectator("BeamP4_meas_E",&BeamP4_meas_E);
reader->AddSpectator("BeamP4_meas_X",&BeamP4_meas_X);
reader->AddSpectator("BeamP4_meas_Y",&BeamP4_meas_Y);
reader->AddSpectator("BeamP4_meas_Z",&BeamP4_meas_Z);
reader->AddSpectator("TargetCenter_Z",&TargetCenter_Z);
reader->AddSpectator("TargetCenter_X",&TargetCenter_X);
reader->AddSpectator("TargetCenter_Y",&TargetCenter_Y);
reader->AddSpectator("kinfit_chisq",&kinfit_chisq);
```

TMVAClassificationApplication.C

Point to the file containing signal and background you want to classify. The structure of this file must match exactly the structure of the files used in the training of the models

```
// PUT FILE YOU WANT TO CLASSIFY HERE
TString fname = "/volatile/halld/home/acschick/2018-01_Redo/DSelector/Flat_2018-01_Redo/0DEG/Flat_2018-01_Redo_042277.root";

if (!gSystem->AccessPathName( fname ))
   input = TFile::Open( fname ); // check if file in local directory exists
else
   input = TFile::Open( "http://root.cern.ch/files/tmva_class_example.root" ); // if not: download from ROOT server

if (!input) {
   std::cout << "ERROR: could not open data file" << std::endl;
   exit(1);
}
std::cout << "--- TMVAClassificationApp    : Using input file: " << input->GetName() << std::endl;

// Event loop

// Prepare the event tree
// - Here the variable names have to corresponds to your tree
// - You can use the same variables as above which is slightly faster,
//   but of course you can use different ones and copy the values inside the event loop
//
std::cout << "--- Select signal sample" << std::endl;


TTree* theTree = (TTree*)input->Get("Flat_2018_Redo");

// Create a file to write the ntuple in and the TTree
TFile *outTarget = new TFile("Fall2018_0Deg_PIM.root", "RECREATE" );
TTree *outTree = new TTree("TMVAOutTreePIM","TMVAOutTree");
```

Define the output file = input file + model inferences

TMVAClassificationApplication.C

If you add more models, you have to add to this list.

```
// get the MLPBNN response and fill the tree with all of its attached variables
MLP_response = reader->EvaluateMVA( "MLP method"        );
BDT_response = reader->EvaluateMVA( "BDT method"        );
outTree->Fill();
```

Perform the application

```
ifarm2402.jlab.org> root -l TMVAClassificationApplicationMINUS.C
```

You can then analyze the data by using MakeClass or MakeSelector on the ROOT tree that this macro outputs.
I've included some basic slides on using MakeClass after this slide.
Example Class: /work/halld/home/acschick/channels/epemmissprot/TMVA_RBHG/Temp_Class_2eTMVA.C
can copy a lot of the code into your own class.

MakeClass()

- Need to write a program that loops over all events.
- Since this is such a common task, ROOT provides a utility to generate a skeleton class to loop over the entries of a tree.

MyTree->MakeClass("NameOfClass");

```
root [1] .ls
TFile**         lepton_v18.dat.root
 TFile*         lepton_v18.dat.root
  KEY: TTree    ntuple;1        lepton_v18.dat
root [2] ntuple->MakeClass("LeptonAnalyzer");
Info in <TTreePlayer::MakeClass>: Files: LeptonAnalyzer.h and LeptonAnalyzer.C generated from TTree: ntuple
root [3]
```

Two new files generated

- Exit ROOT and type ls into your terminal.
- There should be two new files:

  The .h file automatically declares all your variables for you and the file it runs over.

  The .C file contains your loop where actual analysis will occur

  Let's open the .C file

```
#define LeptonAnalyzer_cxx
#include "LeptonAnalyzer.h"
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>

void LeptonAnalyzer::Loop()
{
//   In a ROOT session, you can do:
//      root> .L LeptonAnalyzer.C
//      root> LeptonAnalyzer t
//      root> t.GetEntry(12); // Fill t data members with entry number
//      root> t.Show();       // Show values of entry 12
//      root> t.Show(16);     // Read and show values of entry 16
//      root> t.Loop();       // Loop on all entries
//

//     This is the loop skeleton where:
//    jentry is the global entry number in the chain
//    ientry is the entry number in the current Tree
//  Note that the argument to GetEntry must be:
//    jentry for TChain::GetEntry
//    ientry for TTree::GetEntry and TBranch::GetEntry
//
//       To read only selected branches, Insert statements like:
// METHOD1:
//    fChain->SetBranchStatus("*",0);  // disable all branches
//    fChain->SetBranchStatus("branchname",1);  // activate branchname
// METHOD2: replace line
//    fChain->GetEntry(jentry);       //read all branches
//by  b_branchname->GetEntry(ientry); //read only this branch
   if (fChain == 0) return;

   Long64_t nentries = fChain->GetEntriesFast();

   Long64_t nbytes = 0, nb = 0;
   for (Long64_t jentry=0; jentry<nentries;jentry++) {
      Long64_t ientry = LoadTree(jentry);
      if (ientry < 0) break;
      nb = fChain->GetEntry(jentry);   nbytes += nb;
      // if (Cut(ientry) < 0) continue;
   }
-uu-:---F1  LeptonAnalyzer.C   Top L1     (C++/l Abbrev)----------
```

Include any libraries we need
Define functions
Define constants

```
const double ElectronMass = 5.109989461e-4;
```

```
TFile* oFile = TFile::Open("Histograms.root", “RECREATE");
```

Declare Out-file Histograms, etc.

Write the code that does math and fills histograms **inside** the loop

```
#define LeptonAnalyzer_cxx
#include "LeptonAnalyzer.h"
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>


void LeptonAnalyzer::Loop()
{
//   In a ROOT session, you can do:
//      root> .L LeptonAnalyzer.C
//      root> LeptonAnalyzer t
//      root> t.GetEntry(12); // Fill t data members with entry number 12
//      root> t.Show();       // Show values of entry 12
//      root> t.Show(16);     // Read and show values of entry 16
//      root> t.Loop();       // Loop on all entries
//

//     This is the loop skeleton where:
//    jentry is the global entry number in the chain
//    ientry is the entry number in the current Tree
//  Note that the argument to GetEntry must be:
//    jentry for TChain::GetEntry
//    ientry for TTree::GetEntry and TBranch::GetEntry
//
//       To read only selected branches, Insert statements like:
// METHOD1:
//    fChain->SetBranchStatus("*",0);  // disable all branches
//    fChain->SetBranchStatus("branchname",1);  // activate branchname
// METHOD2: replace line
//    fChain->GetEntry(jentry);       //read all branches
//by  b_branchname->GetEntry(ientry); //read only this branch



   if (fChain == 0) return;

   Long64_t nentries = fChain->GetEntriesFast();

   Long64_t nbytes = 0, nb = 0;
   for (Long64_t jentry=0; jentry<nentries;jentry++) {
      Long64_t ientry = LoadTree(jentry);
      if (ientry < 0) break;
      nb = fChain->GetEntry(jentry);   nbytes += nb;
      // if (Cut(ientry) < 0) continue;
   }
}
```

Write to outfile and close as the last thing

```
oFile->Write();
oFile->Close();
```

Analyze the Data

Because we are using flat trees, all combos in an event get turned into their own event. This means in your analysis you must either

1. set up your own uniqueness tracking in your macro
2. perform accidental subtraction on all histograms even if they don't directly involve the photon energy.

```
root -l
.L MyClassName.C
MyClassName l; l.Loop()
```