Statistical Methods
and Analysis Techniques
in Experimental Physics
ETHZ/UNIZH, FS12

Introduction to ROOT

Pierluigi Bortignon HPK/E-23
pierluigi.bortignon@phys.ethz.ch
Update of Andrea Rizzi 2011 introduction

Outline

Intro

- What is ROOT
- ROOT interactive console
- Important C++ remarks

ROOT

- Reading/writing data with ROOT
- Ntuples, 1D/2D Histograms

Advanced

- Fitting
- Style, options, legend, canvas

Information

- This introduction
  - http://ihp-lx.ethz.ch/CompMethPP/lectureNotes/exercises/rootintro.pdf

  ROOT installation from binaries
  - Check the version of your distribution:

```
cmpp@ihp-lx:~$ cat /proc/version
Linux version 2.6.32-5-xen-amd64 (Debian 2.6.32-38) (ben@decadent.org.uk) (gcc version 4.3.5 (Debian 4.3.5-4) )
```

  - Download the right binaries from http://root.cern.ch/drupal/content/downloading-root
  - source root/bin/thisisroot.(c)sh
  - Root can be found in official repositories for some linux releases
    - Just check it out using your package manager
- Some ROOT & C++ examples
  - http://ihp-lx.ethz.ch/CompMethPP/lectureNotes/exercises/RootExamples.tar.gz
  - To unpack the archive (.tar.gz)

```
wget http://ihplx.ethz.ch/CompMethPP/lectureNotes/exercises/RootExamples.tar.gz
tar -xzvf RootExamples.tar.gz
```

Some Linux Shell commands

- *ls* -> List files in a directory
- *mkdir myDir* -> create a directory “myDir”
- *cd mkDir/mySubDir* -> change current directory
- *mv a.txt b.txt* -> rename a file from a.txt to b.txt
- *rm b.txt* -> remove b.txt
- *cat b.txt* or *less b.txt* -> print the content of b.txt
- *pwd* -> print your current directory
- Editors (as manu as you want):
  - *emacs nw, vi* (console editors)
  - *emacs, xemacs, kate, gedit* etc… (with graphics)

What is ROOT?

ROOT is an object oriented framework for data analysis

- Read data from different sources
- Write data (persistent object)
- Select data with some criteria
- Produce results as plots, fits, etc…

Support "interactive" (C/C++ , Python) as well as "compiled" usage (C++)

ROOT integrates several tools:

- Random number generators
- Fit methods (Minuit)
- Neural Network framework (TMVA)

Developed and supported by HEP community

- Homepage with documentation and tutorials: **root.cern.ch**

ROOT interactive console

Prepare your shell environment

```
source root_v5.30/bin/thisroot.sh
```

Launch ROOT interactive console (CINT interpreter)

```
root_
```

```
   ************************************************
   *                                              *
   *        W E L C O M E  to  R O O T            *
   *                                              *
   *   Version   5.30/02 21 September 2011        *
   *                                              *
   *  You are welcome to visit our Web site       *
   *          http://root.cern.ch                 *
   *                                              *
   ************************************************

ROOT 5.30/02 (tags/v5-30-02@40973, Sep 22 2011, 10:55:04 on macosx64)

CINT/ROOT C/C++ Interpreter version 5.18.00, July 2, 2010
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0] _
```

ROOT interactive console

First, how to quit ROOT! ☺ Type .q

Some useful commands

Load code from external file
.L fileName.C
Load code and *execute* myfunction()
.x myfunction.C
you can append "**++**" to the filename to have code compiled

Some names

CINT is the C/C++ interpreter of ROOT. C++ is not meant to be an interpreter language, so CINT has some limitation!
Aclic is the C/C++ compiler invoked by ROOT when you ask ROOT to compile something.

Some useful tips

•You can use "TAB" key to complete names in ROOT or to get help about the argument of a funcion

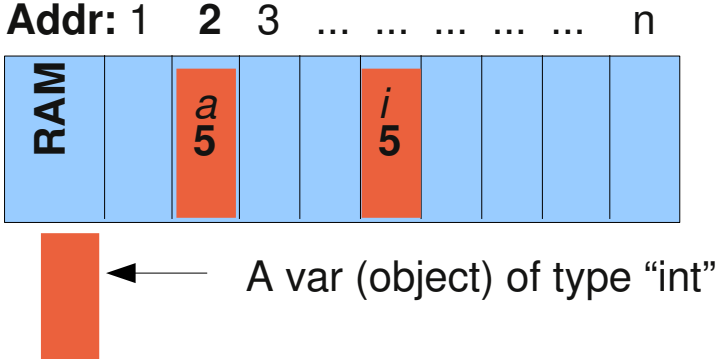
```
root[0] TH1F histo(
TH1F TH1F()
TH1F TH1F(const char* name,...
TH1F TH1F(const char* name,...
TH1F TH1F(const char* name,...
TH1F TH1F(const TVectorF& v)
TH1F TH1F(const TH1F& h1f)
```

TAB

•The history of your recent commands is kept in a file ~/.root_hist
#sh cat ~/.root_hist

Some C++ (pointers, references)

A pointer is a variables that knows where another variable is stored in RAM




```
int a = 5;
int i = a;   // create another object and copy "a"
int * b = &a; //give to b the address of a
int & r = a;  // r is a reference to a
const & c = a;  // c is a const reference to a
```

```
cout << "a value is : " << a << endl;      //  b value is : 5
cout << "b value is : " << b << endl;      //  b  value is : 2
cout << "value pointed by b : " << *b << endl; // value pointed by b : 5

//references are not copies, they refer to the same address
r=6;
cout << a << endl;    // 6
cout << i << endl;    // 5
cout << &r << endl;   // 2

//constant reference cannot be modified (are often used to pass objects
// to functions that should not modify the passed object
cout << c << endl;    // 6
c=7; //this doesn't compile!
```

- the operator * return the value pointed (of a ptr)
- the operator **&** return the pointer of a variable

Some C++ (allocation, scope of objects)

New objects can be created in two ways

- Object created by the users with "new" should be deleted by the users with delete

```
Object * myobj = new Object;
...
delete myobj;
```

- Object declared in a block are deleted automatically when they go out of scope

```
{     My2DPoint P_a(2.1,5.4);
      My2DPoint * P_b = new My2DPoint(2.1,5.4);
} // here P_a is deleted, while P_b is not!
```

- Two common problems
  - Memory leaks when P_b is not deleted
  - Invalid pointers when the address of P_a is taken

```
My2DPoint *P_c = &P_a;//cannot be used after P_a is deleted
```

ROOT memory management

- ROOT objects (Histograms, Canvas, etc) are managed in memory (and disk) by root using "names"
- In the same directory you cannot have two object with the same name (ROOT will complain about memory leaks)
  - ROOT does not like the following

Same name!

```
TH1F * histos[10];
for(int i = 0; i < 10 ; i++) histos[i]= new TH1F("hist","hist",1,2,3);
```

- Objects member functions can be accessed with "." (for instance and reference) or "->" (for pointers) root ""understand" both
  - Interactive ROOT fixes for you wrong usage of pointer vs reference, but when you compile you MUST use the correct syntax.

Standard Template Library

- ROOT supports STD containers:
  - std::vector<double>, std::vector<MyObject>
  - std::pair<std::string, double>
  - std::map<std::string, double>
  - ...
- ROOT works with “const char *”. If you want to use std::string make sure to convert it and to pass the “C string” to ROOT
  - The conversion comes with the member “c_str()”

```
std::string histogramName;
histogramName = prefix+"_EnergyHistogram";
TH1F his(histogramName.c_str(),"Title",10,1,10);
```

Reading data

- ROOT can read data from different sources such as file, network, databases
- In ROOT framework the data is usually stored in **TNuple** (or **TTree**)
  - Trees/Ntuples are like “tables”
    - Each raw represent usually one “event”
    - Each column is a given quantity (energy, mass, angle, etc…)
- Ntuples and Trees
  - can be read from “ROOT files” in which they are stored
  - can be created and filled from ASCII file
  - can be saved by the user

Reading from ASCII file

- Ex: test file 3 columns space separeted
- We can create an "Ntuple" with three columns and read it

```
sh# head -n4 calls.txt
#cost   time     type
1.46    127      2
2.25    124      11
0.82    71       1
```

```
root [0] TNtuple calls("calls","calls","cost:time:type")
root [1] calls.ReadFile("calls.txt")
(Long64_t)192
```

number of row read

name and title

Declaration of columns

```
root [2] calls->Scan()
************************************
*    Row   *       cost *       time *       type *
************************************************
*        0 * 1.4600000 *        127 *          2 *
*        1 *      2.25 *        124 *         11 *
*        2 * 0.8199999 *         71 *          1 *
```

The list of variables to print can be specified in the parenthesis as "var1:var2:var3..."

Saving/reading ROOT files

- We can save TNuple in a file

```
root[2] TFile f("rootfile.root","CREATE")
root[3] f.cd()
root[4] calls.Write()
root[5] f.Close()
```

- And read it back form a new ROOT console

```
root[0] TFile f("rootfile.root")
root[1] TNtuple * calls = f.Get("calls")
```

- When you read back, the pointer to the NTuple is owned by root, you should not delete it
- The "Get" method identify the objects with their name
- You can list the names and the types of the objects in a ROOT file

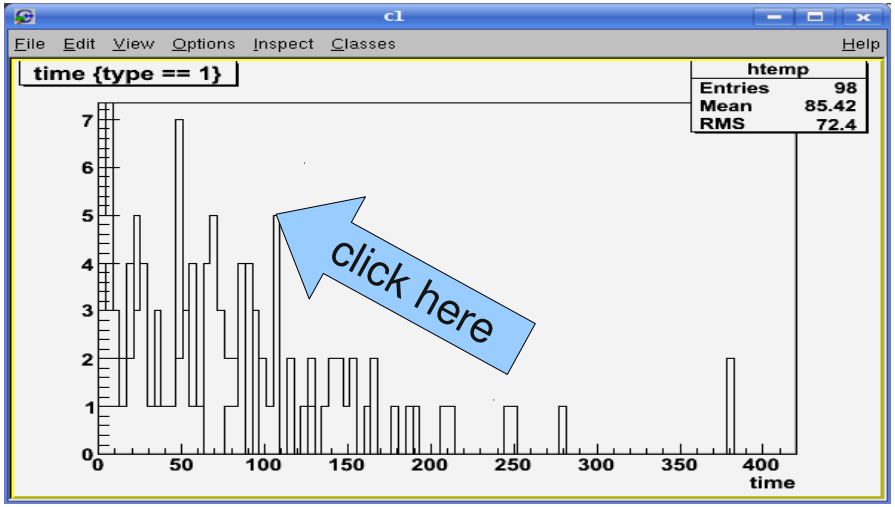
```
root [2] f.ls()
TFile**         rootfile.root
 TFile*         rootfile.root
  KEY: TNtuple  calls;1 calls
```

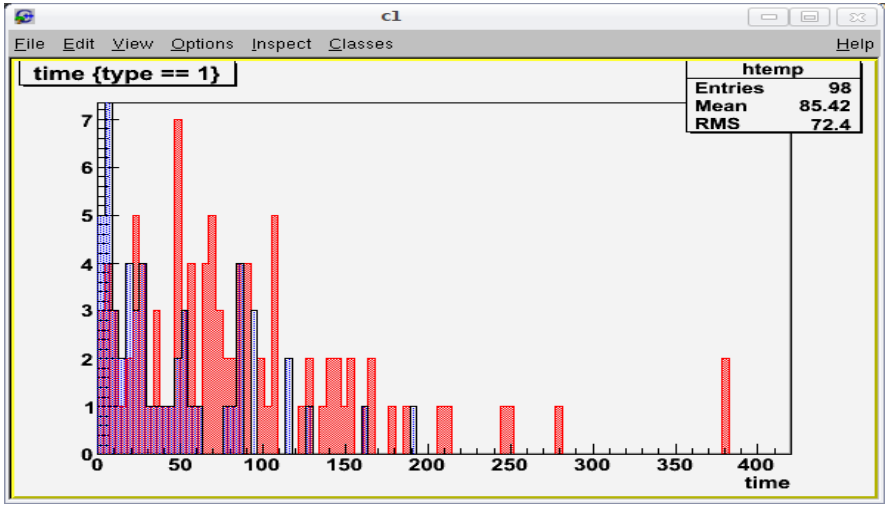
TTree/TNtuple drawing

- You can make an histogram of the distribution of a variable in a Tree

```
root[5] calls->Draw("time")
root[6] calls->Draw("time","type == 1")
root[7] calls->Draw("time","type == 2","same")
root[8] calls->Draw( TAB
```

- Variable to plot
- Cut to apply
- Options for drawing
- To know more...

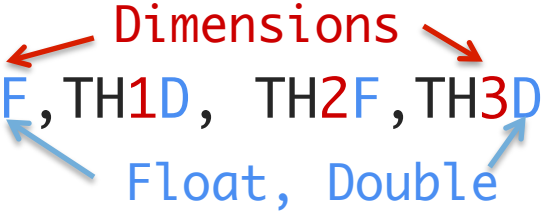




- Properties of draw objects can be changed using the editor (View->Editor). Click on the top of a bin of an histogram to activate the relative option.

Booking histograms

- It is possible (and it is better!) to (user)define histograms: dimension, axis range, number of bins, name and title
- Histogram objects are called TH1F,TH1D, TH2F,TH3D



- To create a new histograms with 20 bins, in range [0,400] you have to do:

```
root[2] TH1F hist("hist","hist",20,0,400);
root[3] calls->Draw("time>>hist")
```

- Now we can do a lot of things on the histograms
  - Changing the properties, fitting, asking integrals, value of bins, overflow, uderflow, scaling, drawing normalized, etc…
- If you want to have more info ask to the "big G" for TH1F

http://root.cern.ch/root/html512/TH1F.html

Manual filling of histograms

- We have already seen how to fill an histogram from TTree/TNtuple::Draw (using “>>histoname”)
- An histogram can be also filled by calling TH1F::Fill function

```
root [0] TH1F::Fill(

Int_t Fill(Double_t x)
Int_t Fill(Double_t x, Double_t w)
Int_t Fill(const char* name, Double_t w)
```

- Fill() function can be useful if in your program/macro you do by hand the loop on the events:

```
TFile f("rootfile.root")
TNtuple* calls = f->Get("calls");
TH1F hist("hist","hist",20,0,10);
.L loop.C
loop(calls, &hist)
hist->Draw()
```

load the file loop.C

loop.C

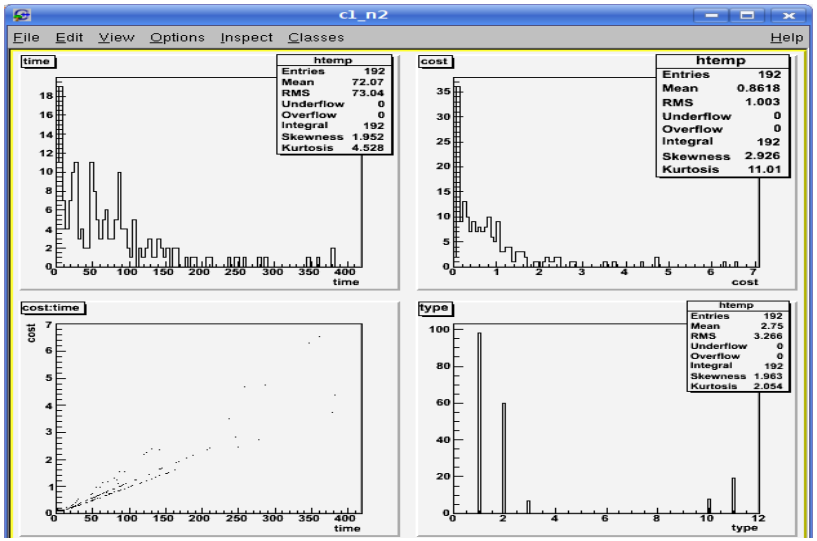
```
loop(TNtuple * nt,TH1F * histo) {
 Float_t time,cost,type;
 nt->SetBranchAddress("time", &time);
 nt->SetBranchAddress("cost", &cost);
 nt->SetBranchAddress("type", &type);

Int_t nevent = nt->GetEntries();

 for (Int_t i=0;i<nevent;i++) {
  nt->GetEntry(i);
  if(type == 1)
     histo->Fill(cost,2.); //weight 2
    else
     histo->Fill(cost,1.); //weight 1
 }
}
```

Canvas, style, options

- If no Canvas is available, ROOT create one when you "Draw"
- Canvas can be created with: `root[0] c1 = new TCanvas`
- Canvas can be splitted: `root[1] c1->Divide(2,2); c1->cd(3);`
- Using canvas you can set log scale or draw a grid

```
root[1] c1->SetGridx(); c1->SetGridy();
root[2] c1->SetLogy();
```

- The information shown in the top right box in a plot can be customized with gStyle->SetOptStat(1111111); (before drawing the histogram)

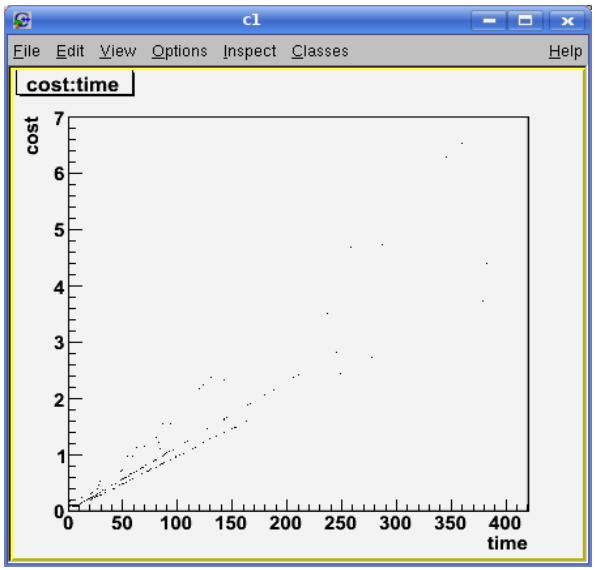


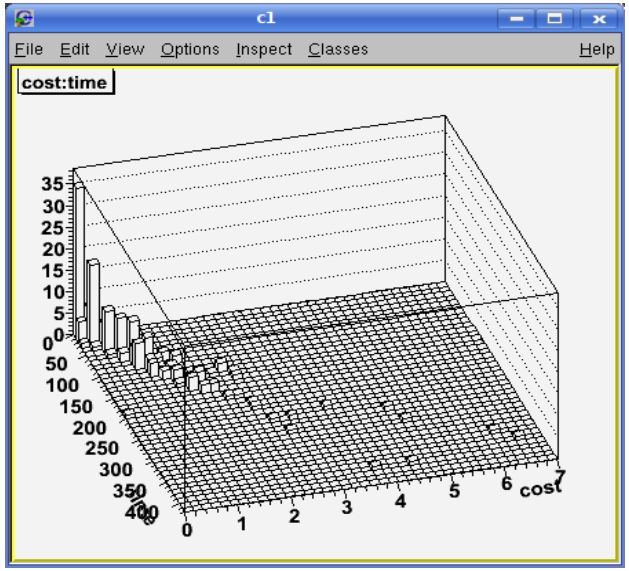
2D histograms

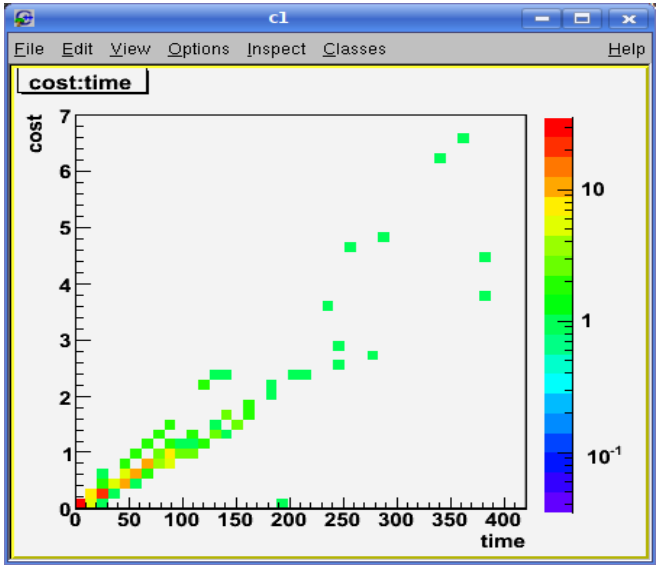
- 2D histograms can be drawn with many different styles

```
root[2] calls->Draw("cost:time")   //default:scatter plot
root[3] calls->Draw("cost:time","","lego")
root[4] gStyle->SetPalette(1)  //set nice palette colors
root[5] calls->Draw("cost:time","","COLZ")
```

- It is possible to rotate with the mouse 3D graphics (e.g. lego plot)
- SetLogz can be used to set log scale for the histogram bin







Fitting histograms

- ROOT provides predefined fittable functions for polynomials, exponentials, Gaussian, landau
- User defined functions can be defined

```
TF1 *f1 = new TF1("fun1","x*[0]*sin(x+[1])",-5,5);
```

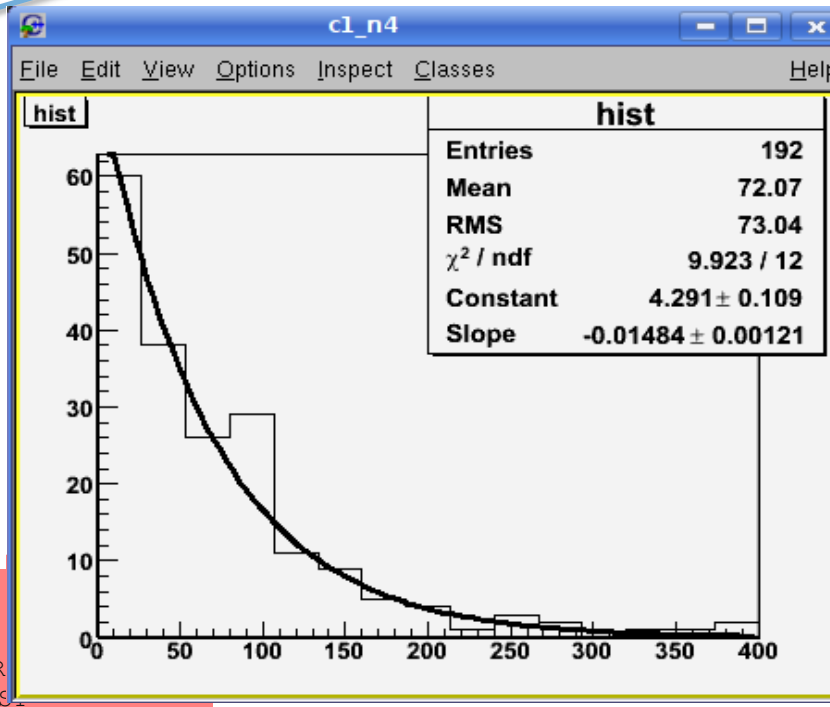
range

Parameters to be fitted

- Histograms can be fitted

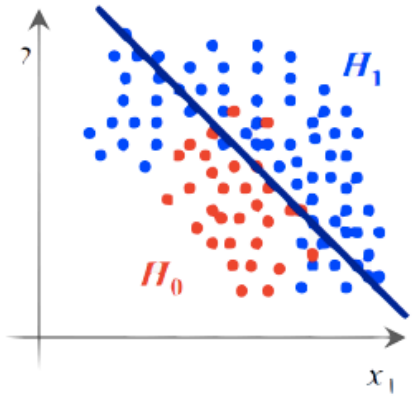
with TH1F::Fit(name of TF1)

ROOT uses Minuit package to fit




```
root [4] hist.Fit("expo")
 FCN=9.92324 FROM MIGRAD    STATUS=CONVERGED      62 CALLS
                   EDM=1.57791e-09    STRATEGY= 1      ERROR MATR
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant     4.29051e+00   1.09210e-01   1.19606e-04  -4.90897e-04
   2  Slope       -1.48356e-02   1.21109e-03   1.32615e-06   2.83027e-03
(Int_t)0
```

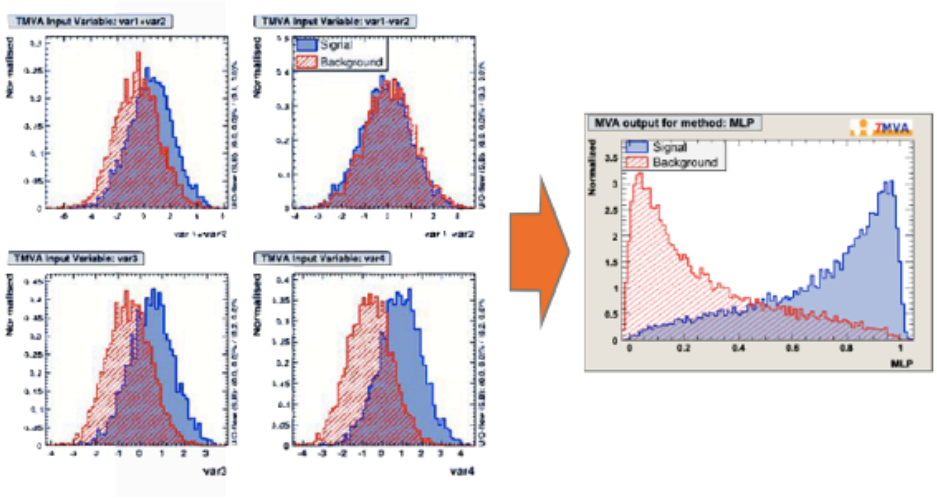
TMVA



- mainly used for Classification of Signal (H1) vs. Background (H0)

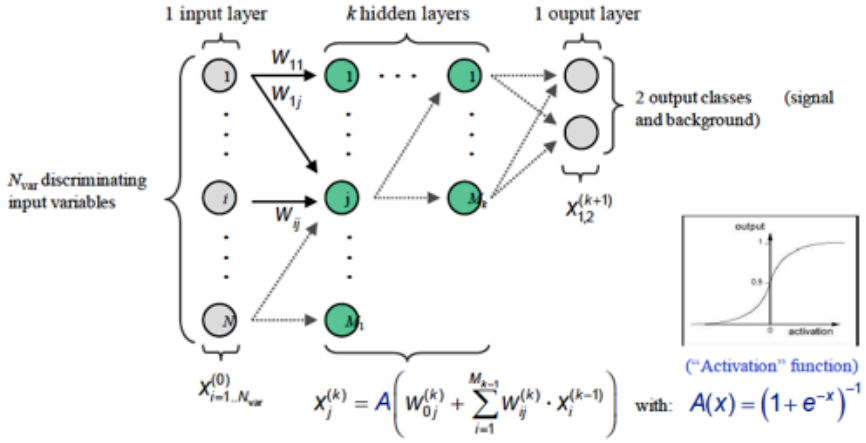
- based on supervised learning (usually on MC smaples)

-providing a common interface to train and use different Classifiers

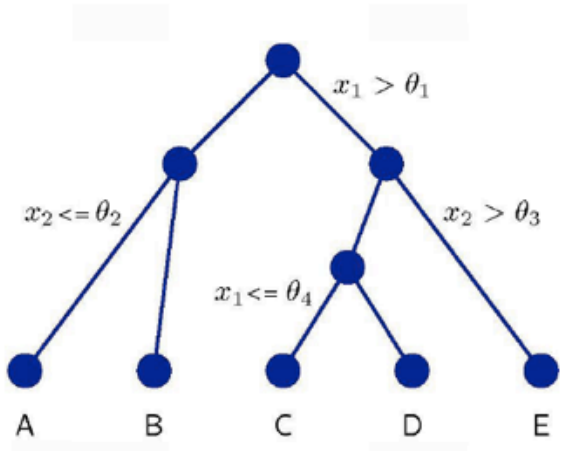


TMVA - Classifiers Examples

- Artificial Neural Networks:



- Decision Trees



...and many more (Cuts, Fisher, Likelihood, FunctionalDiscriminant, kNN, SupportVectorMachine, RuleFit…)

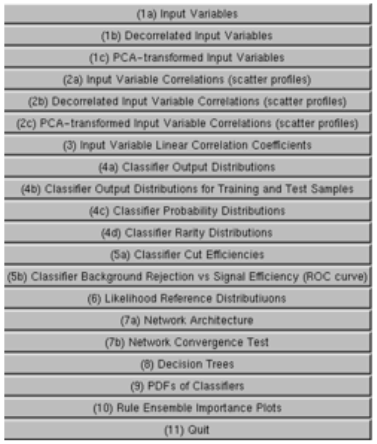
TMVA - Technicalities

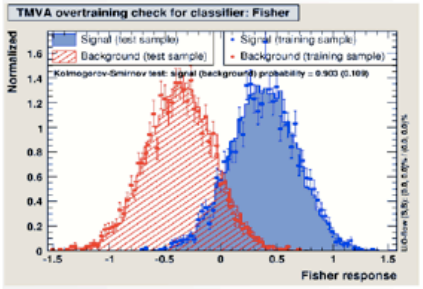
- installed with standard ROOT setup
(or more versions and very good reference available under
http://tmva.sourceforge.net/)

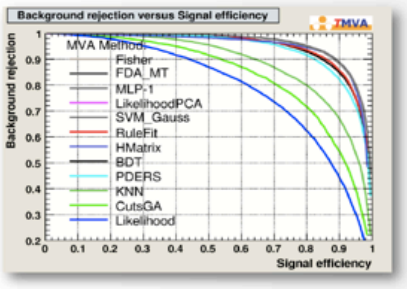
- C and python Version available
- in $ROOTSYS/tmva/test/ are some examples how to run TMVA
- try TMVAClassification.C
(or TMVAClassification.py)

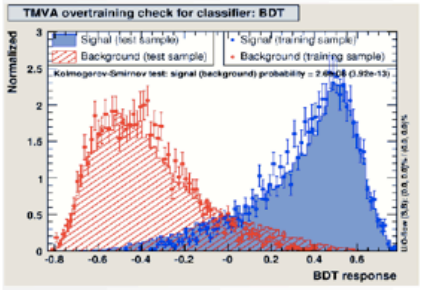
you can run an example by (for example):
root -l TMVAClassification.C\(\"Fisher,Likelihood\"\)

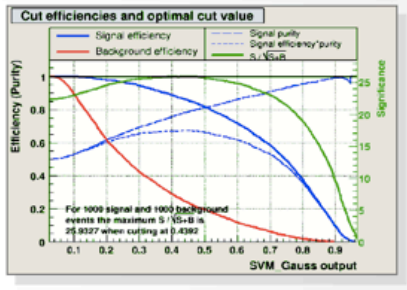
and a GUI is provided to look at many features of the trained Classifiers











Plot options and additional info

- Axis labeling

```
root [1] histo->GetXaxis()->SetTitle("#sqrt{s}")_
```

ROOT support latex syntax

- Legends

```
root [15] leg = new TLegend(0.1,0.5,0.8,0.9);
root [16] leg->AddEntry(h1,"description of hist 1");
root [17] leg->AddEntry(h2,"description of hist 2");
root [18] leg->Draw("FLP")
```

- Printing

You can print TPad in many different formats using TPad::Print function

F = show the "Fill" color/style
L = show the "Line" color/style
P = show the "Point" color/marker style
E = show error bars

http://root.cern.ch/root/html/TPad.htm

```
void Print (const char* filename = "") const

Save Pad contents in a file in one of various formats.

  if filename is "", the file produced is padname.ps
  if filename starts with a dot, the padname is added in front
  if filename contains .eps, an Encapsulated Postscript file is produced
  if filename contains .gif, a GIF file is produced
  if filename contains .gif+NN, an animated GIF file is produced
  if filename contains .C or .cxx, a C++ macro file is produced
  if filename contains .root, a Root file is produced
  if filename contains .xml,  a XML file is produced

  See comments in TPad::SaveAs or the TPad::Print function below
```

http://root.cern.ch/root/html/TLegend.htm

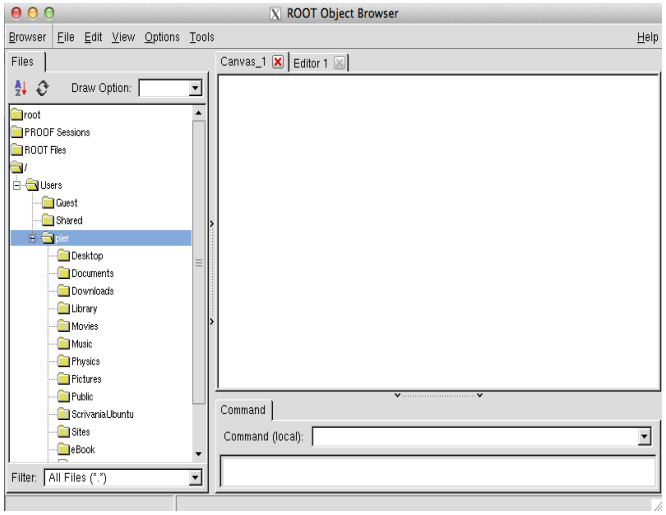
- L: draw line associated with TAttLine if obj inherits from TAttLine
- P: draw polymarker associated with TAttMarker if obj inherits from TAttMarker
- F: draw a box with fill associated wit TAttFill if obj inherits TAttFill
- E: draw vertical error bar

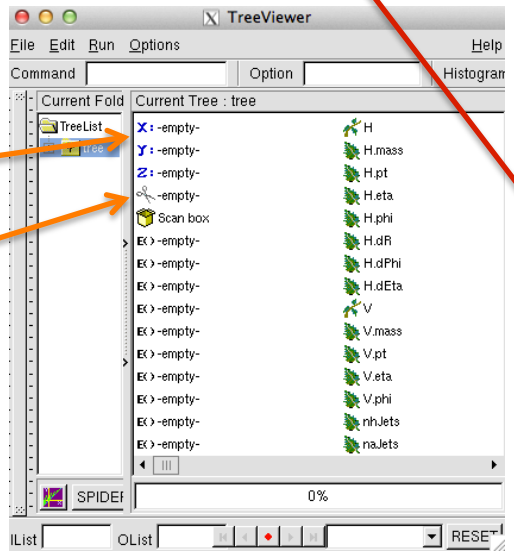
TBrowser, TreeViewer

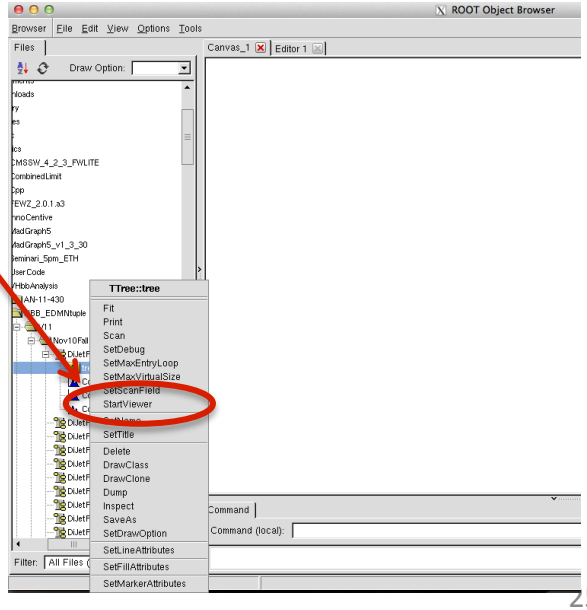
- You can open a new TBrowser in a interactive ROOT session

```
root [20] TBrowser b
```

- Can be useful to interactively browse the content of root files, available histograms, etc…
- For the tree you can use the tree viewer. Right click on the tree and StartViewer







Advantages:
- Draw complex expressions
- Impose your cuts
- Draw multidimensional histograms putting different variables in different axis

Example of standalone app

- If you write your own main, say myapp.cxx

```
File Edit Options Buffers Tools C++ Help
#include <iostream>
#include "TH1F.h"

int main(int argc, char **argv){

  TH1F * h = new TH1F("h","my first TH1F",20, 0, 120);
  h->Draw();

}
```

you can compile it with g++ just importing the ROOT libraries

```
g++ -o myapp myapp.cxx -I $ROOTSYS/include `root-config --glibs`
```

And then run it!

```
./myapp
```

Today exercise

- We try to set up with each of you the ROOT environment
- We try to run some of the tests shown in this presentation, go to:

http://tinyurl.com/rootexamples