ROOT - Based Analysis at CDF

Phillip Koehn
For the OSU Group and CDF
Root2001 Workshop
June 14, 2001

- Introduction
- CDF/OSU Analysis with ROOT
- ROOT Neural Net interface
- ROOT-based Standardized Ntuple
- Impressions and Requests
Introduction

- **Motivation – Why ROOT?**
  - We want a tool that is easy to use for Histogramming, looping, cutting, fitting, plotting, and writing histos to files.
  - CDF Analysis Control framework is in C++, data structure is ROOT-based, ROOT has powerful and nice features, and it is written in C++.

- **Skill Level**
  - We have not taken advantage of the ROOT courses.
  - Started with someone else's ROOT macro and moved on from there.
    - We consult the ROOT web page often (root.cern.ch).
  - While faculty and post-docs have used PAW to do analysis before, grad-students have not – ROOT is the only analysis tool they have used.

- **Analysis work with ROOT**
  - Using Artificial Neural Nets for measuring the single top cross section and the top Mass, and B-tagging.
  - Using standard ntuples composed of multi-branched ROOT trees.
Basic elements of ROOT that we use are...

- **ROOT Trees**
  - Generate ROOT trees from CDF analysis control Framework.
  - Make new ROOT trees from our own macros.

- **ROOT Macros with MakeClass**
  - root[] TFile f(“myfile.root”)
  - root[] MakeClass(“myana”)  
  - Looping, cutting, histogramming, plotting, writing histos to files.

- **ROOT GUI**
  - Touching up plots.
  - TBrowser to check contents of root trees.

- **Fitting with ROOT**
  - Using canned functions:
    - root[] myhist.Fit(“gaus”);
  - User specified functions:
    - root[] myhist.Fit(“MyFcn”);
  - Or fitting a histogram to two or more with the TMinuit class:
    - TMinuit *gMinuit = TMinuit(1)
    - gMinuit->setFcn(LogLiklihoodFcn);  
    - gMinuit->setFcn(ChiSqFcn);
    - gMinuit->mnexcm(“MIGRAD”,args,,);

- **Using gSystem class to interact with operating system**
  - gSystem->CompileMacro();
  - gSystem->Load(“myLib.so”)
  - gSystem->Exec(AnyExe);
Basic elements continued...

- Add our own overloaded methods to global_init.C that serve as command line shortcuts for manipulating histos:
  - Zoning canvases, adding titles, changing divisions.
  - Several Draw() methods to change histogram attributes such as color, marker type, etc…
  - Dump bin contents.
  - Normalizing histograms.
  - Ratio and efficiency of 2 histograms, compute the errors, and create new histograms.
  - Take the integral of a histogram, and plot the new one.
  - Fitting histos (canned or user defined), and subranges of histos, and printing fit results.
  - Printing, ghostviewing,…
In our analysis work we have been using Feed Forward Neural Networks implemented with JETNET.

- Designed for HEP applications and easy to get up and running.
- A collection of FORTRAN subroutines for training/testing NN’s.
- Anonymous ftp from: thep.lu.se (latest version 3.5).

Initially, we worked with many little macros to interact with JETNET and to analyze the performance of our Neural Nets.

We ended up pulling these together into a ROOT macro consisting of methods that provide a flexible and simple interface to setup ANN’s and run them with JETNET.
ROOT/JETNET Overview

- **Composed of two components:**
  - **ROOT macro** (root_to_jetnet.C) uses **command line methods** to:
    - Set up the Neural Net parameters (text file).
    - Initiate the training/testing/running of the Neural Net.
    - Plot input variables, NN performance, error, and output distributions.
    - Creates .C code to compute the Neural Net output based on the weights.
    - The macro is modified by user according to individual needs.
  - **JETNET .exe** (FORTRAN)
    - Reads Neural Net parameter file created by ROOT macro.
    - Performs training/testing with JETNET subroutines.
    - Creates performance and weights file.
    - User will not(or rarely) need to change it.

- **Released for general consumption:**
  - [http://cdfpc2.mps.ohio-state.edu/root_to_jetnet/rtj.html](http://cdfpc2.mps.ohio-state.edu/root_to_jetnet/rtj.html)
  - In the CVS repository at CDF
  - pkoehn@fnal.gov, catutza@fnal.gov, neu@fnal.gov.
Using the Interface

- **Input files.**
  - Contain a global set of all possible inputs one may choose.
  - Generated by user.
  - 1 file per sample (signal, bkg1, bkg2, ...).
  - Rows (Events) and Columns (Pattern Variables).

- **Set the parameters of the Neural Net.**
  - Choose input variables, the number of hidden and output nodes, the number of events to train and test, minimization method, the number of training cycles, etc...

- **Run JETNET.**
  - Modes: single shot, loop over input nodes, loop over the number hidden nodes, loop over combinations of both.
  - Produces output files: performance, error, and resultant NN weights.

- **Plot the results of your neural net.**
  - Performance, error, neural net output.

- **Use the net.**
  - Apply the C-code that computes the neural net output.
What features of ROOT were the most helpful?

- We’re running just a simple interpreted macro.

- Efficient as a command line interface and graphics tool.

- Interaction with the operating system:
  - The work processing the NN is done by the fortran jetnet.exe – run from the ROOT macro.

- We have converted the macro into a class that may be compiled.
  - Easier to add methods and keep track of code.
  - Will not run much faster as most of the execution time is in running the fortran jetnet.exe.
STNTUPLE: ROOT based standardized ntuple.

- Closely related components: a data format and a set of utility classes.

- The data format is a multibranched ROOT tree.
  - Generated with a Stntuple (CDF Analysis Control) Module running on input raw or processed data.
  - Reconstructed data objects such as e’s, muons, taus, photons etc... Several RAW data branches are also included.
  - One can add new branches to the standard ones, and to switch off filling of the branches one doesn't need.

- The utility classes provide access to the data.
  - Implemented in a framework for specialized or user defined analysis modules.
Doing Analysis using STNTUPLE

- Easy to access low level and high level data objects.
- Modular framework allows user to write more complicated analysis scripts.
- Fast: One edits, compiles, then runs a ROOT script.
  - When the ROOT script compiler is used, recompilation and reloading of a file about 2000 lines long takes of the order of 10 seconds on 500 MHz PentiumIII box. You can modify your analysis, rerun it and see the results within a minute.
  - Process 10k single track events from the XFT data block in about 15 seconds.
- Less painful introduction to C++ and a good way to learn and exercize ROOT- based analysis tools.
STNTUPLE Framework: Data Blocks

- **StnDataBlock Class**
  - The data written into Stntuple are organized in blocks, similar to the data blocks of HBOOK column-wise ntuples.
  - Each block corresponds to a top-level branch of ROOT tree.
  - A tree can contain an arbitrary number of branches, so user can decide which branches to create/fill in the beginning of the job.
  - An analysis job can read only those branches which are necessary, improving the I/O performance.

- **Data Block Types:**
  - TCalDataBlock
  - TCesDataBlock
  - TClicDataBlock
  - TCmuDataBlock
  - TCmpDataBlock
  - TCmxDataBlock
  - TCprDataBlock
  - TGenpDataBlock
  - TStnJetDataBlock
  - TStnMetDataBlock
  - TStnEleDataBlock
  - TStnMuonDataBlock
  - ...

•
The StnAna class provides a framework for data I/O and adding analysis modules.

- Specify input/output Stntuple.root files.
- Use singly or chain multiple analysis modules together.
- Access methods and data elements of individual modules. (e.g. Grab a set of pointers to e’s, mu’s, jet’s passing cuts, fitting, plotting.)
- Run(): Initiate the processing events

StnModule is the base class for analysis modules.

- Contains overloaded methods: BeginJob(), BeginRun(), Event(), EndJob(), EndRun().
- Access Data Blocks.
- Access methods and data members of other modules in the chain.
- Implement filter (or derive from StntupleFilterModule class).
- Booking, filling, plotting, and saving of histograms.
- Other module types include: InitStntuple, StntupleMaker, StntupleFill
{ 
  TStnAna x("results/ttbar_prod_cdfSim.root");
gSystem->CompileMacro("TTopCand.cc","k");
gSystem->CompileMacro("TTopFindModule.cc","k");
  TTopCand* tc = new TTopCand("TopC","TopC");
x.AddModule(tc);
  TTopFindModule* tf = new TTopFindModule(tc,"TopFind","TopFind");
x.AddModule(tf);
  TStnOutputModule out("goodevents.root");
x.SetOutputModule(&out);
x.Run();
  tf->SaveHistograms("MyFavouriteHistos.root");
}
What we like...

- It is easy to get up and running and do the basics quickly.
- Fast turn around time from editing to running.
- Writing command line shortcuts to manipulate histos.
- Accessing the operating system from a macro.
- Standardized Ntuples – STntuple Classes.
- The GUI is nice, but we don’t really use it that much.
- Resources at the ROOT website are useful:
  - ROOT TALK - we get the most help from this
  - Tutorials
  - ROOT Class Categories
  - Documentation area
difficulties...

- We have the ability to make ROOT crash often.
  - Usually need to recover by quitting then restarting.
  - Unloading code does not seem to work that well.

- It would be helpful if the error output from crashes were more informative.
  - “segmentation violation…”

- Debugging code.
  - We’ve used the CINT debugger on simple macros. It is useful, but we would like to debug compiled ROOT macros as well.
  - Have recently used gdb.
  - End up doing things like running in the Trace mode:“ root[] .T ” or resorting to the insertion of print statements.
Requests

- A place on the ROOT website other than the tutorials and ROOT Talk, where any user may submit and search for code.
- Maybe something like hotscripts.com?
- Macros do not have to be guaranteed to work.
- Organized by categories like:
  - Histogramming/formatting
  - Fitting
  - Debugging
  - Etc…
Conclusion

- We enjoy using ROOT and will continue to use it for our needs as much as possible.