Data Analysis Exercises

Wouter Verkerke
(NIKHEF)

Getting started

- The input files (all very small) for the various exercises are located on the web at
  - http://www.nikhef.nl/~verkerke/statcourse_2011/
  - NB: If you run linux, easiest way to download a file to the local directory is 'wget <url>' (the file 'course.tar' contains all files if you find that easier)
- Very basic ROOT (for those who have never used it)
  - ROOT is *the* analysis environment used in High Energy Physics. The application consists of a C++ interpreter (C++ is the command line language) and a large series of classes that define the ROOT functionality (divided in several major topics such as IO, Graphics, Histogramming, Fitting & Minimization etc...)
  - Starting root: Lunix/MacOS: 'root –l' (-l suppressing splash screen when opening), Windows: click on ROOT icon from installations
  - The command line is interpreted C++
  - To quit ROOT type '.q'
  - To load a macro file (file with one ore more C++ functions) in the interpreter do '.L filename.C'. Loading a macro does not execute any code
  - You can then execute any function defined in the file in it by simply calling the function name on the command line
  - To load a macro *and* execute the function with the name identical to the macro type '.x blah.C' (will load macro and execute function blah(), if defined)

Getting started

- Very basic ROOT introduction (cont'd)
- In addition to plain C++, many classes are defined that implement ROOT functionality.
  - We will only use very few in this tutorial, and examples will be provided.
  - You can find the complete documentation for any class online at root.cern.ch → Documentation → Reference Guide
  - *Look at demo macro ex0.C* which illustrates the use some of the very basic ROOT classes that we need for this tutorial
  - Histogramming → class TH1
  - Random number generation → class TRandom
  - Math functions → class TMath
  - Vectors → class TVectorD
  - Matrices → class TMatrixD
  - Graphics windows → class TCanvas

ROOT practicalities when running locally on laptop

- Practical workflow:
  - Choose a directory where you put the downloaded input files and where you will work on your exercises
  - Have one window with ROOT session in the directory where you have put the input files
    - (1) **Linux/MacOS**: just cd to the directory where you put these files prior to starting ROOT. **Windows**: either change the startup location of ROOT (right-click on icon, select Properties, and change the "Start in field"), or on your ROOT command line type gDirectory->cd("C:\\your\\directory\\name") – please note that you need to use a double backslash here!
  - Have one editor open with the exercise file you are working on
  - Then iterate: edit your .C file, then run it by e.g. typing '.x ex1.C'

- Annotations in exercises:
  - 'CODE' – means that you need to write some code
  - 'EXEC' – means that you need to run your code and interpret its output
- Macros have been tested with ROOT 5.26, 5.28, 5.30
  - Problems? Please ask (I didn't test everything on all platforms)

Exercise 1 – The central limit theorem

- In module 1 we saw that the Central Limit Theorem predicts that the sum of N measurements has a Gaussian distribution in the limit of N → ∞, independent of the distribution of each individual measurement
  - In this exercise we will investigate how quickly this convergence happens as function of N.
  - We start with a 'fake' measurement resulting in a value x with a uniform distribution between [0,1] (i.e. this is very non-Gaussian)
  - Then we will look at the distribution of x_1+x_2, $x_1+x_2+x_3$, etc and compare these with the properties of a Gaussian distribution
- Start with file ex1.C
  - This macro books a ROOT histogram, runs 10000 experiments and fills the 'measured' value of x in the histogram and plots the histogram and the end of the run.
  - EXEC: Look at the macro and run it ('root -l ex1.C' from the OS command line, or '.x ex1.C' from the ROOT command line)

Exercise 1 – The central limit theorem

- Modify the loop so that instead of filling the result of a single measurement in the histogram you store the result of Nsum measurements
  - CODE: Allocate a variable xsum that it is initialized to zero
    - The variable Nsum is already defined in the macro as first argument to macro ex1(). Its default value when unspecified is 1.
  - CODE: Make a loop from from j=1,Nsum (inside the existing loop over i) and in new inner the loop add the value 'measurement' as returned by the 'gRandom...' line to the value of xsum.
    - The histogram defined by the macro has its range already defined as [0,Nsum] so that the summed measurement values always fit in the range of the histogram
  - EXEC: Run the macro again now passing value 2 as argument for Nsum '.x ex1.C(2)' (or root –l 'ex1.C(2)' from the OS command line. Note that in this case the quotations are essential). Look at the distribution
  - EXEC: Repeat for Nsum=3,5,10,20 and 100.

Exercise 1 – The central limit theorem

- You will see that around Nsum=10 the distribution is already looks quite Gaussian.
  - This is however mostly for the 'core' of the distribution. The convergence of the tails of the distribution is much slower as we will see next in this exercise
- To compare the distribution to a Gaussian we compare the number of events in the 1,2,3,4,5 sigma range to that expected for a true Gaussian distribution
  - I.e. we expect for a true Gaussian that 68% of the events is in the ±1 sigma range. Then we count which fraction of the xsum distribution is in that range
  - And we repeat for 2,3,4,5 sigma
- To do so we need to calculate the expected sigma of the Gaussian by calculating the root of the variance of the distribution
  - Calculate first (on a piece of paper) the variance of a uniform distribution in the range [0,1].
  - To do so, use the formula $\sigma \equiv \sqrt{V(x)} = \sqrt{\overline{x^2} - \overline{x}^2}$

    where you can use $\overline{\alpha} = \int \alpha \cdot F(x) dx$ to calculate it, where F(x) is the distribution you are averaging over (F(x) is uniform distribution in range [0,1] in this case)

Exercise 1 – The central limit theorem

- Then once you have variance for a single measurement of x, determine what the variance is for the sum of N identical measurements
  - If you need help, look at the slides on Central Limit Theorem of module one
- Update the code to add this additional information
  - CODE: At the beginning allocate a variable Nsigma1 and initialize it to zero. This will hold the number of events in the 'one-sigma' range.
  - CODE: In the 'experiment loop', once you have calculated Xsum, determine if the answer is inside or outside the 'one-sigma range', i.e. it is outside the range [-1*sigma,+1,sigma].
  - CODE: At the end of the loop print the fraction of events Nsigma1/Ntot, which is the fraction of events ***outside*** the one-sigma range of the distribution.
  - Compare it the fraction expected for a Gaussian distribution. Tip: You can get the exact fraction of events *outside* a n-sigma Gaussian distribution from the following ROOT expression:

    double gaussfrac = 1-TMath::Erfc(n/sqrt(2))

    where 'n' is the number of sigmas (i.e. 1 will give you 100%-68%≈32%)

Exercise 1 – The central limit theorem

- Note that there is a *statistical error* on the measurement of Nsigma1/Ntot which is (to good approximation) sqrt(Nsigma1)/Ntot
  - Compare Nsigma1, error(Nsigma1), and the 'true value' of Nsigma1 for a Gaussian distribution on one line
  - EXEC: Do this for Nsum=2,5,10,20,100
  - You will see that 10000 experiments provides plenty precision to see that the one-sigma range of the distribution of Xsum converges rapidly to that expected for a Gaussian distribution.
- Now repeat the exercise for 2,3 sigma range.
  - CODE: To do so, add variables Nsigma2, Nsigma3, fill them in the event loop with the corresponding ranges and compare them (with their errors) to the matching fractions for a true Gaussian distribution.
  - EXEC: Do this for Nsum=2,5,10,20,100
  - Do you have enough statistics to measure the convergence for 2 and 3 sigma? If not, increase the number of experiments by e.g. a factory of 10
- Finally add the 4,5 sigma range
  - CODE & EXEC: How many experiments do you need to verify 5-sigma convergence?
    (Feel free to stop this exercise if runs start to take too long)

Exercise 1 – The central limit theorem

- What does it mean?
  - If you have done your exercise correctly you'll see the following results for the Nsum=20 run with Nexp=1.000.000 for 1,2,3,4,5 sigma
    - `n = 3198780 frac = 0.319879  +/- 0.00017 Gauss = 0.317311   rel. = 0.008`
    - `n = 450384  frac = 0.0450384 +/- 6.7e-05 Gauss = 0.0455003  rel. = -0.010`
    - `n = 22954   frac = 0.0022954 +/- 1.5e-05 Gauss = 0.0026998  rel. = -0.149`
    - `n = 329     frac = 3.29e-05 +/- 1.8e-06  Gauss = 6.33425e-05 rel. = 0.480`
    - `n = 0       frac = 0        +/- 0        Gauss = 5.73303e-07`
  - While the 2,3 sigma fractions are fairly close to Gaussian (rel=frac-Gauss/Gauss) the 4-sigma number is 50% off
  - E.g. your interpretation of how often a result 4 times the sqrt(variance) away from the central value happens is 50% off w.r.t the Gaussian distribution
  - Verifying 5 sigma results is a very time consuming business (even when a simulation of your measurement is as trivial as throwing a single random number)

Exercise 2 – Error propagation

- In module 1 we saw that errors on measurements x,y can be propagated to any function f(x,y) as follows

$$V(f) = \left(\frac{df}{dx}\right)^2 V(x) \quad ; \quad \sigma_f = \left|\frac{df}{dx}\right| \sigma_x$$

- If we assume that df/dx is linear over the range of σ_x one can also calculate V(f) as follows
  - $V(f) = [0.5*(f(x+\sigma_x)-f(x-\sigma_x))]^2$ → $\sigma(f) = 0.5*(f(x+\sigma_x)-f(x-\sigma_x))$
- We will now exercise this type of error propagation in a ROOT macro
  - Input file ex2.C is a self-contained ROOT macro with a function ex2() and a function f(x,a,b).
  - EXEC: Look at the contents of ex2.C and run the macro once (run 'root –l ex2.C' from the OS command line, or start root first with 'root –l' and then type '.x ex2.C' on the ROOT command line

Exercise 2 – Error propagation

- Calculate the error on f, propagating the error on input variables x,a,b (without correlations)
  - CODE: Calculate the value of f(x+dx,a,b)-f(x,a,b) and store it in a variable dfx
  - CODE: Along the same lines, calculate 0.5*(f(x,a+da,b)-f(x,a,b)) and store it in dfa and the equivalent things for parameter b
  - CODE: Calculate the variance on f from dfx, dfa, dfb
- Now we redo this exercise in the matrix formalism
  - Given a vector d=(dfx,dfa,dfb) we can express V(f) as follows

$$V(f) = d \cdot (C \cdot d^T)$$

    where C is the correlation matrix
  - For example in 2 dimensions w/o correlations for f(x,y) we can write

$$V(f) = \begin{pmatrix} dfx & dfy \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} dfx \\ dfy \end{pmatrix}$$

Exercise 2 – Error propagation

- Now write this in C++ code
  - ROOT has vector and matrix classes that can do the matrix algebra for us
  - CODE: Create a TVectorD object named df and fill it with (dfx,dfa,dfb)
  - CODE: Create a 3x3 identity TMatrixD object named C and fill it with the contents of the identity matrix (all elements are initialized to zero so you only have to fill the diagonal) Calculate V(f) using vector df and matrix C as follows

    double vf = df*(C*df) ;

    and compare the answer to the first calculation

Exercise 2 – Error propagation

- In the matrix formalism it is trivial to add correlations between the variables a,b,x in the error propagation
  - CODE, EXEC: Add a 50% correlation between a,b (remember that the correlation matrix must be symmetric) and evaluate the variance on F again
  - CODE, EXEC: Now add a 50% anti-correlation between a,b and evaluate the variance again.

  (NB: Remember that the error on f is always sqrt(Vf))

Exercise 3 – Multi-Variate Analysis

- ROOT is distributed with the 'Toolkit for Multivariate Analysis', a toolkit that allows to train and apply many of the multi-variate analysis techniques shown in Module 2.
  - Here we will run it on a number of sample events
  - Copy input file ex3_makesample.C. This is a macro that can generate several 'toy' input samples.
  - Copy input file ex3_driver_rootXXX.C. where XXX is the 3-digit version code of ROOT (526,528,530) This is the driver macro to run the TMVA toolkit.
  - **Copy both files (makesample and driver) to the ROOT ROOT installation subdirectory tmva/test *and set your working directory there***
    - Linux/MacOS: cd $ROOTSYS/tmva/test, copy files here
    - Windows: right-click on root icon, click on properties and then set 'Start in' to the right directory. To get the ROOT base path, look at the path to the root executable and remove, '/bin/root.exe'. Then your starting directory will be the ROOT base path with /tmva/test appended.

Exercise 3 – Multi-Variate Analysis

- Make sample #0
  - EXEC: Linux/MacOS: execute from the OS command line 'root -l –b -q ex3_makesample.C(0)' (windows: do .x ex3_makesample.C(0) from the ROOT command line)
  - This make a file sample0.root which contains a sample of 'toy' background events and 'toy' signal events
  - Signal: Gaussian distribution in x (mean=-3, sigma=3)
  - Background: Gaussian distribution in x(mean=+3, sigma=3)
  - NB: There is a dummy (uniform) Y variable in the data because TMVA refuses to work with a single variable in some versions

Exercise 3 – Multi-Variate Analysis

- To analyze the first sample, start a ROOT session
  - EXEC: Execute '.L ex3_driver_rootXXX.C', this loads the driver application
  - EXEC: Now analyze the first sample by issuing the following command on the ROOT prompt ex3_driver_rootxxx(0,"x,y","Fisher,BDT,MLP") ;
  - This will train a Fisher discriminant, a Boosted Decision Tree and a Multi-Layer Perceptron on this sample
  - Observe how e.g. a BDT *trains* a lot quicker than a MLP, but takes longer to evaluate on the data.
  - While the training is running (~5 minutes), take a moment to review the techniques being trained in the slides of Module 2
  - When the training is finished, a window will pop up with various choices.

Exercise 3 – Multi-Variate Analysis

- Explore the following menu items in this order
  - 1a) Input distributions (just one for this example)
  - 4a) Classifier output distributions (observe characteristic spikiness of BDT output)
  - 4b) Same, but overlay of both test and training samples. Difference in these are indicative of overtraining (not in this sample)
  - 5b) ROC curve (signal vs background efficiency)
  - 5a) Efficiency curves (show signal and background efficiency vs discriminant, as well as S/sqrt(S+B) which helps to find optimal cut for a given amount of signal and background **(Skip this one if you run on windows)**
  - Change the amounts of signal and background in the dialog box and see how the S/sqrt(S+B) changes shape
  - 9) – 11) Control plots for individual algorithms (these show e.g. network architecture, BDT structure etc...)
  - *NB: **If you run on Windows** or don't have a compiler installed not all options will work (notable 5a will crash ROOT on windows w/o compiler, but some others may also not work.*

Exercise 3 – Multi-Variate Analysis

- Now create sample 1
  - by running 'root -l –b –q ex3_makesample.C(1)'
  - Sample 1 has three observables :x,y,z
  - Signal = Gaussian(x,0,3)*Gaussian(y,0,3)*Gaussian(z,0,3)
  - Background = Flat in (x,y,z)
- Now analyze sample 1
  - Add the likelihood discriminant: execute
  - .x ex3_driver_rootXXX.C(0,"x","Fisher,BDT,MLP,Likelihood")

    from the ROOT command line nd look at the plots as for sample 0, but add the plots that the correlation (2a) and the plots that show the performance of decorrelation (2b,2c,2d,1b,1c,1d)
- You will see that the performance of Fisher is very good for sample 0, but much worse for sample 1
  - Try to understand why that is (see slides on Fisher discriminant and MLP)

Exercise 3 – Multi-Variate Analysis

- Sample 2
  - Signal and background differ only in their correlation information
  - Signal = Gaussian({x,z,y},0,3) 80% correlation between (x,y) and 50% correlation between (y,z)
  - Background = Gaussian({x,z,y},0,3) 80% anti-correlation between (x,y) and 50% anti-correlation between (y,z)
- Sample 3
  - Multi-dimensional variant of sample 0
  - Signal = Gaussian(x,-3,3)*Gaussian(y,-3,3)*Gaussian(z,-3,3)
  - Background = Gaussian(x,+3,5)*Gaussian(y,+3,5)*Gaussian(z,-+3,5)
- Sample 4
  - Intertwined donuts. Two observables (x,y)
  - Signal = donut in (x,y) centered at (-2,-2), radius 5, width 1
  - Signal = donut in (x,y) centered at (+2,+2), radius 5, width 1