Day 2

Wouter Verkerke, NIKHEF

WARNING - ROOT 5.34.21 Bug on MacOS

- If you have a (source-compiled) version of ROOT 5.34.21 (and .20) on MacOSX there is a bug in the graphics core routines that causes TMVA to crash once you ask to see plots.
- The bug has understood and been fixed yesterday by the ROOT team (on my request).
- To apply this fix to your local ROOT installation do the following:

1. Retrieve the bug-fixed file `TStyle.cxx` from http://www.nikhef.nl/~verkerke/statcourse_2014/
2. Go to your ROOT installation directory ($ROOTSYS)
3. Copy the fixed file to the subdirectory `core/base/src/`
4. Run 'make' again (in the ROOT installation base directory) Compilation should take less than 30 seconds

Wouter Verkerke, NIKHEF

Exercise 4 – 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 Lecture 2.
  - Here we will run it on a number of sample events
  - Copy input file `ex4_makesample.C`. This is a macro that can generate several 'toy' input samples.
  - Copy input file `ex4_driver_root532.C`.
    This is the driver macro to run the TMVA toolkit.

Exercise 4 – Multi-Variate Analysis

- Make sample #0
  - EXEC: Linux/MacOS: execute from the OS command line
    'root -l -b -q ex4_makesample.C(0)'

    Windows: do `.x ex4_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 4 – Multi-Variate Analysis

- To analyze the first sample, start a ROOT session
  - EXEC: Execute '`.L ex4_driver_root532.C`', this loads the driver application
  - EXEC: Now analyze the first sample by issuing the following command on the ROOT prompt
    `ex4_driver_root532(0,"x,y","Fisher,BDT,MLP");`
  - This will train a Fisher discriminant, a Boosted Decision Tree and a Multi-Layer Perceptron for sample 0 using variables x,y
  - 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 Lecture 2
  - When the training is finished, a window will pop up with various choices.

Exercise 4 – 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 4 – Multi-Variate Analysis

- Now create sample 1
  - by running '`root -l -b -q ex4_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 ex4_driver_root532.C(1,"x,y,z","Fisher,BDT,MLP,Likelihood")
```

  from the ROOT command line and 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 4 – Multi-Variate Analysis

- Repeat the exercise for the other samples below
- 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
- If you have time left, you can edit ex4_makesample.cxx and code a few additional probability density functions of your choice
  - For syntax explanations see slides at the end of the following lecture (Lecture 3)

Exercise 5 – Unbinned Maximum Likelihood fit

- This a mostly a demonstration exercise only that shows some of the functionality and the syntax of the RooFit toolkit for data modeling that we'll be using in the next exercises
- Copy file `ex5.C`, look at it and run it. This macro does the following
  - It creates a Gaussian probability density function
  - It generates an unbinned dataset with 10k events from that function
  - It performs and unbinned ML fit of the Gaussian model to the toy dataset
  - It makes a plot of the data and overlays it with the Gaussian model
- Now comment out the part of the code labeled as 'block 1' and run the macro again
  - This code will print out the covariance matrix and correlation matrix of the fit parameters. Verify that cov(m,s)=corr(m,s)*σ(m)*σ(s) using the printed errors for mean.
- Now comment out the code labeled 'block 2' and run again.
  - This code will visualize the uncertainty of the model on the canvas using the error propagation technique. At 10K the event the uncertainty is very small (you can see it if you zoom in on the peak region of the pdf)

Exercise 5 – Unbinned Maximum Likelihood fit

  - Change the number of generated events from 10K to 100 and change the binning of the data in the plot from 100 bins to 10 bins (this is the argument in the `w.var("x")->frame()` call. Run again.
  - Lower the number of generated events from 100 to 10 and run again. The error on the shape will now be significant, but you see that an unbinned ML can reliably fit very small event samples
- Now comment out code block 3
  - This will visualize the error on the pdf shape due to the uncertainty on the mean parameter only

Exercise 6 – Analytical vs numeric MLE

- For certain simple cases it is possible to calculate the ML estimate of a parameter analytically rather than relying on a numeric estimate. A well known case is that of the fit of a lifetime of an exponential distribution
- Copy `ex6.C` and run it. This example performs an unbinned MLE fit to an exponential distribution of 100 events to estimate the lifetime.
- Now we aim to construct the *analytical* ML estimator of the lifetime (do this part on paper, not by computer)
  - Write down the probability density function for the exponential lifetime distribution.
  - It is essential that you formulate a *normalized* expression, i.e. Int F(t) dt = 1 when integrated from 0 to ∞.
    The easiest way to accomplish that is to divide whatever you expression you have by the integral of that expression over dt. (You can calculate that normalization integral on paper)

Exercise 6 – Analytical vs numeric MLE

  - Next write down the analytical form of the negative log-likelihood –log(L) for that probability density function given a dataset of N events labeled these x_i in your expression. *Be sure to also include the pdf normalization term in the expression*
  - The analytical ML estimate of the lifetime tau then follows the requirement that d(-logL)/dtau = 0. Calculate the expression for this derivative solve and derive the value of tau for which the requirement d(-logL)/dtau holds.
- Finally, implement the analytical calculation of MLE estimator for tau in the code of `ex6`.
  - Uncomment block one, which implements a look over the dataset, retrieving the values of the decay time one by one and build your calculation of the analytical estimate of tau with that of the numeric calculation from fitTo()
  - Explain why you might have minor discrepancies between the analytical and numeric calculations.
  - Increase the event count from 100 to 10000 and run again

Exercise 7 – The effect of outliers on fits

- Outliers in distributions that are strongly peaked can create serious converges problems in fits that model such peaked distributions and do not take outliers into account.
  - Copy `ex7.C`, look at it and run it. This example generates a Gaussian distribution with a width that is relatively narrow compared to the defined range on x, and fits it to a Gaussian model.
  - Now uncomment BLOCK1. The added code 'manually' adds an event at x=3 and refits the distribution. Look at both fits carefully (just by eye, no need to make a true quantitivative check using a χ^2). Zoom in on the x axis if necessary. Does the outlier impact the result of the fit? (Also check the impact of the fitted value of mean,sigma)
  - Now move the position of the outlier event to x=4, run again and evaluate the situation again. Calculate what is the probability to obtain an event at x=4 or larger for this model? (You can use the 'TMath::Erfc' formula of Ex 1 to calculate this, but keep in mind that that formula evaluate the probability for |x|>Z, rather than x>Z)
  - Repeat the above exercise (including evaluation) at x=9.
  - Now uncomment BLOCK2. This fits the data to an improved model that foresees in a (small) flat background component that absorbs the outlier events and make the fit of the Gaussian component of the model function well in the presence of outliers. What value of `fsig` do you expect a priori to get out from the fit?
  - Now change the code fragment '`fsig[0,1]`' by '`fsig[0.999']` modifies to model to have a one permille fixed background component instead of a floating component. Rerun the fit. Does it still work well? Explain why (not)?

Exercise 8 – An MLE fit for an efficiency function

- This exercise demonstrates the procedure of an unbinned ML fit for an efficiency curve.
  - Copy `ex8.C`, look at it and run it. This macro create a data sample (x,c) in which c is a discrete observable that tells us if the event with value x has passed a selection (or not). The goal is to determine the efficiency function `eff(x)` of the selection encoded in observable c.
  - The initial exercise creates an efficiency histogram from the dataset D(x,c) where each bin in x contains the fraction of events that have passed the cut. The efficiency histogram is constructed to have symmetric binomial errors on the data. Look at the slides of Module 1 to remind yourself how symmetric binomial errors are defined. An efficiency function 'fitfunc' is then fit to the data using a χ^2 fit. Explain what approximation we are making by doing this?
  - Now uncomment BLOCK 1 of the the exercise and run again. This will make a plot 'all data' and 'accepted data', as well as perform an unbinned ML fit to measure the efficiency function. This fit is done using a probability density function F(c|x) that returns 'effFunc(x)' if the event is accepted and '1-effFunc(x)' if the event is rejected and is fit to the full dataset D(x,c)
  - Lower the number of events generated from 1000 to 200 (change variable N) and see what happens. Do the χ2 and likelihood fits return correct results? Then lower N to 50 and run again.