DS@LHC 2015 Workshop

TMVA Tutorial – Helge Voss

General remarks on the computer exercise using TMVA

Get the exercises from:

```
  wget http://hvoss.home.cern.ch/hvoss/DSLHC2015TMVAExercises.tgz
tar -zxvf DSLHC2015TMVAExercises.tgz
cd DSLHC2015TMVAExercises/
```

In ROOT6 (unlike ROOT5) all TMVA GUIs, plotting macros are part of the ROOT libraries. There is no need to copy files like TMVAGui.C and alike in the working directory or macro path anymore. If you are using ROOT5 you might still get away with it by creating a .rootrc file in the working directory which should look like:

```
Unix.*.Root.MacroPath:    .:$(ROOTSYS)/tmva/test:$(ROOTSYS)/macros
Unix.*.Root.DynamicPath: .:$(ROOTSYS)/lib
```

AND uncommenting the line

```
// else gROOT->ProcessLine(".L TMVAGui.C"); TMVAGui(outfileName);
```

in all the example scripts/macros.

In every root6 installation you find a couple of example scripts that demonstrate the usage of TMVA in `$(ROOTSYS)/tutorials/tmva`. I personally find them too overloaded because they allow the choice of many different classifiers and many different example configurations to be chosen simply via a command line argument of the script. The scripts used in this tutorial on the other hand only contain the essentials and therefore need to be 'edited' in order to select/deselect classifiers.

1 Generate toy MonteCarlo events for this exercise

Please execute the program generator:

```
cd ~/DSLHC2015TMVAExercises/MonteCarlo/
make
./RunMonteCarlo
```

which will create a root tree with MC events (`MonteCarlo/MonteCarloEvents8D.root`) used in the tutorial.

2 Understanding your variables

The first step in any event selection is to collect and study the discriminating variables. Have a look root Trees in `MonteCarlo/MonteCarloEvents8D.root`, plot the variables compare signal and background distributions. This task is facilitated (to some extent) in TMVA as it provides plots of the variables with signal and background overlayed.

Look at `TMVAClassificationPlot.C`. It provides the basics of how TMVA is used which can be summarized in a few steps:

- instatiation of the TMVA::Factory
  ```
  TMVA::Factory *factory = new TMVA::Factory( "TMVA", "TMVAoutput.root","");
  ```
- telling the factory which variables to be used
  ```
  factory->AddVariable( "var0", "Variable 0", "units", 'F' );
  ```
- telling the factory which event trees to use
  ```
  factory->AddSignalTree    ( signalTree , optionalEventWeightFactor );
  ```
- telling the factory about how to use the events for the training/testing
  ```
  factory->PrepareTrainingAndTestTree( "","", "nTrain_Signal=0:nTrain_Background=0:S
  ```
- telling the factory which classifiers (methods) are used and what their configuration values are. e.g.:
  ```
  factory->BookMethod( TMVA::Types::kBDT, "BDTG","!H:!V:BoostType=Grad");
  ```
- train/test/evaluate
  ```
  factory->TrainAllMethods();
  factory->TestAllMethods();
  factory->EvaluateAllMethods();
  ```

In addition, you can specify individual event weight if that's what is given by your Monte Carlo generator (or sWeight algorithm or whatever) using one of the three following ways to do so:

```
factory->SetSignalWeightExpression    ("weight1*weight2");
factory->SetBackgroundWeightExpression("weight1*weight2");
factory->SetBackgroundWeightExpression( "weight" );
```

where weight1, weight2 and weight are examples of how the variable that carries the event weights is called in your root tree.

Now, run the script: `root -l TMVAClassification.C`
and look at the GUI that pops up. The top most buttons are used to inspect the variables and their correlations. The linear correlations are displayed as the correlation matrix, the "non-linear correlations" (interactions) can be looked at by studying the various 2D scatter plots. Please also look at `MonteCarlo/Log8D.png` which simply plots the 2D scatter plots as an overlapp between signal and background. (note: sometimes it

is helpfull to have 'less events' in these plots as otherwise, the plot is totally overshadowed by whatever (signal or background) is ploted last. Compare `MonteCarlo/Log8D.png` `MonteCarlo/Log8D_5000events.png`

Look at the log printout of TMVA. Rather than just looking at possible separation power between signal and background of the individual variables, the printout also contains the calculated separation between signal and background for the individual variables. Look for:

```
--- IdTransformation         : ---------------------------------
--- IdTransformation         : Rank : Variable   : Separation
--- IdTransformation         : ---------------------------------
```

- Which variables are the most promising 2 variables according to this printout?
- The separation of the individual variables obviously does not take into account and separation power of the variables that is only visible/obtainable when treating them correlated. Which 2 variables look most powerful from the 2D scatter plots?
- Are these variables 'uncorrelated'? What does the correlation matrix say ?

3 Train a classifier

- Run a linear classifier (LD) on these two variable pairings (var1:var2) and (var0:var1). To do so, please comment out in the macro
  `TMVAClassification.C`
  all `factory->AddVariable(..)` statments, appart from the two for the variables var1 and var2 or var0 and var1. Then add the classifier by calling
  `factory->BookMethod( TMVA::Types::kLD, "LD", "H:!V:" );`
  (just before the call to 'TrainAllMethods()')
  the first argument in `BookMethod` specifies the classifier type. The second is a 'unique' name that you need to give to any classifier you book. It is the one that will be displayed for that particular classifier but has no meaning by itself. The same name (in addition to the name given to the factory) will also be used to name the 'weight file', in which the parameters of the trained classifier are stored. The third argument is a string which communicates the various configuration options for the classifiers. For a full list of available options for each classifier please consult:
  `http://tmva.sourceforge.net/optionRef.html`
  Note: You might want to re-name the name given to the factory e.g.
  `TMVA::Factory *factory = new TMVA::Factory( "TMVA_Var1VsVar2", "TMVAoutput.root","");` .
  The name (first argument) is taken as part of the 'weight file names' and changing this name allows you not to overwrite the training results for the two different variable pariings.
- Which 2 variable pairs gives better performance.

- Now run a boosted decision tree (default settings) on these two variable parings. Which pariing is now better? Guess why ?
  `factory->BookMethod( TMVA::Types::kBDT, "BDT", "!H:!V");`

4 Tune the classifiers configuration parameters

The default parameters are far from optimal and ALLWAYS need to be adjusted. The performance of the BDT from the previous example can still be increased substantially. Let's do that first for just using the 2 variables 'var0 and var1'.

The most important parametes for the BDTs are:

- `MinNodeSize:`
  The minimum fraction of the event sample (e.g. 2%) requested at least in any leaf node

- `MaxDepth:`
  The maximum tree depth (depends on the interaction of variables. Typically somehwere between 2 and 5.

- `AdaBoostBeta:`
  Something like the 'learning rate' (typically somewhere between 0.01 and 0.5)

- `NTrees:`
  The number of boost steps. (just needs to be large enough. Too large mainly costs time (which is important, too)

- `BoostType:`
  `AdaBoost` or the somewhat more powerfull `'Gradient Boost'`. For the latter, the role of `AdaBoostBeta` is replaced by the `''Shrinkage''` parameter

"Play" with those parametes until you've reached a performance of at least:

```
--- Factory  : Testing efficiency compared to training efficiency (overtraining check)
--- Factory  : ------------------------------------------------------------------------
--- Factory  : MVA         Signal efficiency: from test sample (from training sample)
--- Factory  : Method:      @B=0.01          @B=0.10         @B=0.30
--- Factory  : ------------------------------------------------------------------------
--- Factory  : BDTTuned    : 0.092 (0.107)   0.440 (0.472)   0.825 (0.844)
--- Factory  : BDTDefault  : 0.050 (0.072)   0.378 (0.441)   0.811 (0.830)
```

The above lines are found in the log-printout. They give the signal efficiency at various background efficiencies (1%, 10% and 30%). These values are obtained from the 'test sample', the part of the data that has not been used in the training in order to get an unbiased performance estimate. Note: The above numbers mean, that for example at if form my analysis I would need a background reduction of at 99%, (i.e. 1% background

efficiency), the tuned classifier is about a factor of 0.092/0.050=1.82 more efficient !!! Almost DOUBLED the efficiency! Well worth the little effort in 'tuning', isn't it? B.t.w, the numbers in parenthesis are the corresponding efficiency values if evaluated on the training sample. If they differ by a lot, it hints at a possible overtraining which then typically results in a less then optimal classifier performance on the test sample. If observed, try to lower the flexibility of the classifier. (i.e. larger MinNodeSize, smaller MaxDepth, smaller AdaBoostBeta etc..)

Using the GUI, check also the ``Classifier Output Distributions'' to see if the classifiers look 'healthy' (nice and smooth, distributions) and the ``ROC curves'' (backgr. rejection vs signal eff.) or (1/backgr.efficiency vs signal eff) to asses and compare nicely the performance of the classifier(s). The later ROC curves are 'truncated' towards the zero backgr. efficiency as obviously the tend to infinity there. This representation however shows more clearly the difference in performance of the classifiers in the very high purity (high background rejection) region which is often important. To see which classifier does best for things like cross section measurements, check the ``Classifier Cut Efficiencies'' from the GUI which also shows the statistical significance of the selected event sample (S/Sqrt(S+B)) and at which working point (mva-cut value corresponding to a certain signal(backgr.) its maximum is reached.

5 Use a Neural Network

Boosted Decision trees are very popular. They are more robust (require in real life examples less effort in tuning the parameters). But you'll later see why I think it is well work also experimenting with Neural Networks. Repeat the previous exercise for the 'TMVA::kMLP' classifier.
The most important parametes to tune for the NNs are:

- `HiddenLayers :`
  The number of nodes in the various NN layers. HiddenLayers=N+2:N for example means: two hidden layers, the first with N+2 nodes and the second one with N nodes, where N stands for the number of input variables. This is the most important parameter (the network architecture) that determines mainly the model flexibility. NOTE: Try 1 or maximum 2 layers. "Deep networks" are not really supported yet in TMVA.

- `VariableTransform:`
  Too bad, that's in most cases best set to 'normalize' which means, all variables are normalized to the same value range between [0,1], but the TMVA default is set to 'none' :(. In this tutorial it is unnecessary, as all the variables have the same value range anyway. I thought about that too late :(.

- `UseRegulator:`
  Apply a L2 norm regulator to avoid overtraining (prefers small weights, and weight

close to zero translate to an effective almost linear classifier. Hence this regulation procedure prefers 'simple models' and hence helps to avoid overtraining, even if the number of layers/nodes allows in principle for a very complicated model.

- :
  The number of boost steps. (just needs to be large enough. Too large mainly costs time (which is important, too)

- `NeuronType:`
  `sigmoid` or the often better because of its symmetrie`'tanh'`.

"Play" with those parametes until you've reached a performance like this:

```
--- Factory  : Testing efficiency compared to training efficiency (overtraining check)
--- Factory  : ------------------------------------------------------------------------
--- Factory  : MVA         Signal efficiency: from test sample (from training sample)
--- Factory  : Method:      @B=0.01          @B=0.10         @B=0.30
--- Factory  : ------------------------------------------------------------------------
--- Factory  : MLPTuned   : 0.099 (0.104)   0.459 (0.467)    0.839 (0.845)
--- Factory  : MLPDefault : 0.061 (0.052)   0.270 (0.297)    0.590 (0.601)
```

6 More on Variable selection

Now as you have already resasonably tuned BDTs and Neural Networks for 2 variables, add the other available variables and have a look again at the performance.

- run the training on the 'tuned classifiers' from before on all 8 variables.

- which variable would you first try to eliminate again? Do so, re-train and compare the performance.

- when I did so with my 'tuned classifiers'
  (BDTTuned: NTrees=2000:MinNodeSize=1.%:BoostType=AdaBoost:AdaBoostBeta=0.05:MaxDep
  (MLPTuned: NeuronType=tanh:VarTransform=N:HiddenLayers=N+10:UseRegulator)
  the performance of the BDT stays basicall the same, the MLP however actually performs better after removing the variables. Explain WHY this happens ?

Note: The tuned config paramters for the 2variable case are not necessarily the best also for the 8 variables. But for now let's not spend too much time on re-tuning those. With the tuned parameters as given above and w/o variables 3 and 4, the neural network actually performs better than the boosted decision tree. This is often the case for 'toy examples'. In real life however it is often difficult to train a network to get better than the BDT because of lack of data etc. A feature that is however almost always in favour of the neural network and hardly ever considered is demonstrated in the next section.

7 Decision Boundaries

In multidimensional parameters space, the visualization of the actual decision boundary that an MVA classifier cut corresponds to is difficult. It is however instructive at least to have a look at them in this example. You find for this a script called `PlotDecisionBoundary.C`. It plots the 2D projection of two of the variables, and plots the decision boundary (for an MVA-cut value corresponding to minimum miscalssificaton of signal and background events) where all the other variables are set to their 'mean value'.

Now: In order to keep things simple and not having to edit the variable list in the `PlotDecisionBoundary.C` script, you need to re-run the BDTTuned and MLPTuned classifier again with all 8 variables enables.

Then start a root session and execute: `root [0] .L PlotDecisionBoundary.C`

```
root [1] PlotDecisionBoundary(8, "weights/TMVA_BDTTuned.weights.xml", "var0", "var1");
```

This will plot the decision boundary of the classifier in weights/TMVA_BDTTuned.weights.xml for the variables var0 and var1. now compare this decison boundary to the one obtained for the Neural network?

```
root [1] PlotDecisionBoundary(8, "weights/TMVA_MLPTuned.weights.xml", "var0", "var1");
```

- What do you observe? Which boundary would you rather base your a physics analysis on ? Why ?

8 (1D-projective) Likelihood or "Naive Bayes Classifier"

This is one of the first 'multivariate classifiers' which managed to enter the HEP analyses. It approximates the `PDF(S|x)` and `PDF(B|x)` assuming uncorrealted variables (meaning the joint PDF is the product of the one dimensional PDFs, which then can be easily estimated with good accuracy by histogramming the MonteCarlo generated events. The classifier then, just like given by the Neyman-Pearsons lemma, is given by the ratio `PDF(S)/PDF(B)` (typically transforms this linearly to "`PDF(S)/(PDF(S)+PDF(B))`") which then, if the approximation of uncorrelated variables were true, is bascially the best possible classifier. Unfortunatly, the approximation is hardly ever 'correct'. But it's worth trying to make it 'as good as possible'. Even if the resulting classifier might in the end still be slightly worse than an BDT or NN, it is a classifier very easy to study (i.e. study impact the impact of systematic errors on the classifier performance by simply modifying the likelhood reference distributions)

Therefore, we look at what we can do here in this example with this classifier. Book the likelhood classifier:

```
factory->BookMethod( TMVA::Types::kLikelihood, "Likelihood", "H:!V:" );
```

run it, compare its performance to that of the BDT/NN. Why aren't you surprised ?

- config parameter tuning.
  - Look at he `Likelihood Reference Distributions` from the GUI. What do you observe?
  - Use the parameters `NAvEvtPerBin` (i.e. the average number of events per bin in the reference distribution. The number of bins will be adjusted accordingly) `PDFInterpol=Spline2` and `NSmooth=3` to get a better estimate of the 'true underling PDFs'
  - The Likelihood still performs very poor. Why ? Look also at the 'correlation matrix' (i.e. from the TMVAGui)
  - 'decorrelate' variables 'var5' and 'var6' . Do this directly in the "Factory::AddVariable" call, which takes as first argument any string specifying a TFormula, that can also be given to a TTree::Draw command. (e.g. "var5+var6")
  - after this, the performance is still poor, why?
  - try to come up with a transformation for variable var1, that tries to remove the 'correlation' in the 2D scatter plot var0:var1. NOTE! A TFormula can also contain statements like: `var1>0 ? 1 : 0` which would correspond to taking the heavyside function of the variable var1.
  - after you have experimented with the likelihood and the variable transformations, run also the BDT/NN calssifiers on these transformed variables. What do you observe.
  - It seems that a transformation like
    `factory->AddVariable("var1>0.5*var0?var1-var0: var1", "Variable 1", "units", '`
    give the best performance for the Liklihood. Quit a boost compared to the original one, actually. However, BDT/NN seem to loose slightly, although BDT's (and for sure NN's to some extend too) DO also train much easier on variables that are not correlated. BDTs in particulary don't like strong linear correlations, as in the end, it operates on many small rectanguar cuts. Therfore one concludes that some information seems to get lost using this transformation. But .. nobody forbids us to use 'only 8' combinations of 'the 8 variables' .. so ... try to add in addition the 'other' option of combining var1 and var2, which does not simply map the var1 down onto the var0 axis, but rather maps it over to the var1 axis. This TOO allows for more rectangular cuts like separation of signal and background. So add a nineth variable like:
    `factory->AddVariable( "var1>0.5*var0?var0-0.9*var1:var0", "Variable 0a", "unit`
    and run BDT/NN and Likelihood again. Is there a surprise ??

9 Remarks

- Variable Transformations:

We've seen how to apply simple variable transformations directly when adding the variables to the factory. There are a couple of (decorrelation) transformations avaiable in TMVA that can be invoked individually for each classifier. In order to see how they transfrom the variables (1D-projections and the 2D Variable-vs-variable) distributions, they can also be invoked at the instatiation of the 'TMVA::Factory' (:Transformations=I;D;P;G,D: in the option string). Note however, it is typically much better to 'manually' construct your own custom transformation of the variables.

- "Most useful" variable transformations

  Probably the most useful variable transforamations that should be considered is to 'smooth' out variables that exhibit sharp peaks. Typically such peaks occure 1 (e.g a cosine of an opening angle) or zero as result of some pole in the cross section. Such sharp features are 'poison' for any numeric algorithm ! They can however easily be avoided using 'acos', 'log' or similar simple functions as transformation to the numeric value of the variable

- Break up the variable space into 'categories'

  Unfortunatly the MethodCategory does not work as intended. Nontheless, it is important and often results in much simpler classifiers and better performance if the event sample is split into subsamples with systematically differnt behviour. (i.e. in the toy example, one part of the sample that exhibits a strong linear correlation between var0 and var1 and the other half which does not.) In the exercise, we applied an intermediate solution using a piecewise transformation to make the variable easier to handle. In real life, one would probably split the event sample, train, tune and later apply two indipendent classifiers to events of the respective regions. Give it a try ! You might improve on the combined performance. I would not be surprized if in this toy example, the Likelhood performance could almost match that of the BDT/NN if treated that way. Let me know if it does !

- In this tutorial, we only looked at the 'training' of the classifier. Once this is done, use the TMVA::Reader to 'retrieve' the trained classifier from the weight files and apply the selection in your 'event loop' Look at then example in 'TMVAClassificationApplication.C' . Note: Any parameter transformations that were easily done in the 'Factory::AddVariable' e.g.

  ```
  factory->AddVariable("var5+var6", "variable 5");
  factory->AddVariable("var5-var6", "variable 6");
  ```

  still need to be announced to the reader by saying:

  ```
  factory->AddVariable("var5+var6", \&transVar5);
  factory->AddVariable("var5-var6", \&transVar6);
  ```

but obviously the addresses &transVar5 and &transVar5 cannot simply be refering to the addresses of variable var5 and var6 in the TTree. Here one needs now to calcualte oneself transVar5 = var5+var6 ; transVar6 = var5-var6; in the event loop.