Table of Contents

TMVA - Toolkit for Multivariate Data Analysis

Tutorial

This tutorial treats a full TMVA training analysis and the application of the training results to the classification of data sets with unknown sample composition. Regression problems are not treated here, but their training and application is very similar. We refer to the TMVA home page and the TMVA Users Guide for more information.

Preparation

The tutorial uses TMVA-v4.1.0 and has been tested for various ROOT 5 releases >= 5.14. Because remote ROOT graphics is slow it is preferred that the user runs the tutorial on a local computer (including a local installation of ROOT >=5.14 and TMVA-v4.1.0). Take care that the **ROOTSYS** environment variable is properly set. For non-local usage, it is assumed that the users have a CERN afs account (lxplus).

Local installation

It is assumed that ROOT >=5.14 has been installed (binaries are sufficient, they do include the necessary header files of all ROOT classes). If not, please consult the ROOT download page.

1. Download TMVA-v4.1.0 from Sourceforge.net
2. Unpack the tar.gz file:

```
tar xfz TMVA-v4.1.0.tgz
```

1. Check the contents of the package:

```
cd tmva-V04-01-00
ls -l
```

The top directory contains a Makefile for the TMVA library. The directory doc contains `README` and `LICENSE` text files. The setup scripts for bash (zsh) and (t)csh are located in the directory test. Content of the subdirectories (more directories may show up, but they are irrelevant for this tutorial):

- **`src`**: source files of all TMVA classes
- **`inc`**: header files for all TMVA classes
- **`include`**: symbolic link to directory containing include files (link created by setup script)
- **`lib`**: contains shared library `libTMVA.1.so` after source code compilation
- **`test`**: example scripts and executables for training and application analysis, and all plotting scripts for the TMVA evaluation

1. Setup the environment:

```
cd test; source setup.[c]sh; cd ..
```

2. Create the TMVA shared library: `libTMVA.1.so` (this will take a few minutes)

```
make
ls -l lib/
```

Note that if you use ROOT >= 5.11/03, (a possibly older) version of TMVA will also be part of the

ROOT distribution. There should not be any conflict with the newly created package as long as `libTMVA.1.so` is properly loaded (or linked), as is done in the macros that are executed during this tutorial. (Do not forget to source setup.sh in the test directory)

Using CERN afs

Use CERN afs on interactive lxplus with preinstalled TMVA version:

1. Login on lxplus.cern.ch
2. Setup ROOT (v5.14/00e, sys=amd64_linux26 on lxplus):

```
cd /afs/cern.ch/sw/lcg/external/root/5.14.00e/slc4_amd64_gcc34/root
source bin/thisroot.[c]sh
```

... and continue as above.

Local Installation on a MAC

In a few configurations with MAC OS there might be a problem with 32- vs. 64 bit libraries. Such problems may be solved by replacing the Makefile in the tmva main directory by this one here. Credits go to Ulrik Guenther who pointed this problem out to us. After updating the Makefile please continue as above.

Running an example training analysis

The TMVA distribution comes with a idealised toy data sample consisting of four Gaussian-distributed, linearly-correlated input variables. The toy is sufficient to illustrate the salient features of a TMVA analysis, but it should be kept in mind that this data is by no means representative for a realistic HEP analysis. TMVA foresees three ways to execute an analysis: (*i*) as a macro using the CINT interpreter, (*ii*) as an executable, and (*iii*) as a python script via PyROOT. The first two options are run in the following.

Running the example analysis as a ROOT macro

Note that there is no significant loss in execution speed by running the macro compared to an executable, because all time-consuming operations in TMVA are done with compiled code accessed via the shared library.

- To quickly run the example macro for the training of a Fisher discriminant, type the following:

```
cd test
root -l TMVAClassification.C\(\"Fisher\"\)
```

The macro should run through very quickly, and pop up a GUI with buttons for validation and evaluation plots. The buttons show the distributions of the input variables (also for preprocessed data, if requested), correlation profiles and summaries, the classifier response distributions (and probability and *Rarity* distributions, if requested), efficiencies of cuts on the classifier response values, the background rejection versus the signal efficiency, as well as classifier-specific validation plots. **Try them all!**

- We can now attempt to train more than one classifier, e.g., Fisher and the projective likelihood:

```
root -l TMVAClassification.C\(\"Fisher,Likelihood\"\)
```

Again the GUI will pop up, and it is interesting to check the background rejection versus the signal efficiency plot. Fisher performs better, which is due to the ignorance of correlations in the projective likelihood classifier. Plotting the likelihood response distributions indicates the problem: background slightly rises in the signal region, and vice versa.

- The likelihood performance on this toy data can be improved significantly by removing the correlations beforehand via linear transformation using a TMVA data preprocessing step:

```
root -l TMVAClassification.C\(\"Fisher,Likelihood,LikelihoodD\"\)
```

  Plotting the background rejection versus the signal efficiency graph shows that the decorrelated likelihood classifier is now as performing as Fisher.
- Let's have a look at the latest **log file**.
- If you have time you could now run **all** preset classifiers, which may take O(10 mins):

```
root -l TMVAClassification.C
```

  After terminating, try how the the background rejection versus the signal efficiency plot looks like. Also, it is interesting to look into the response distributions of the classifiers.

Running the example analysis as an executable

- For the proof of principle, let's only run the training of a Fisher discriminant.
- Build the executable:

```
make TMVAClassification
```

- And execute it:

```
./TMVAClassification Fisher
```

- Start the GUI and plot:

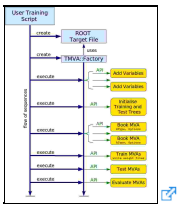
```
root -l TMVAGui.C
```

The training phase

Let's look at the analysis script:

```
emacs TMVAClassification.C
```

Discussion of main analysis steps:



1. Instantiation of the **Factory**
2. Registration of the signal and background trees
3. Registration of the input variables
   (note that the types 'F' or 'I' given indicate whether a variable takes floating point or integer values, respectively: 'F' stands for both float and double, 'I' stands for int, short, char, and unsigned integers)
4. Preparation of independent training and test trees
5. Booking of the classifiers (discussion of the classifier labels and the configuration options)
6. Train, test and evaluate all booked classifiers
7. Created outputs: weight files, standalone C++ classes, target file

Applying the trained classifiers to data

Once the training phase (including training, testing, validation and evaluation) has been finalised, selected classifiers can be used to classify data samples with unknown composition of signal and background events:

```
root -l TMVAClassificationApplication.C\(\"Fisher,Likelihood,LikelihoodD\"\)
```

The macro runs over signal events only. Plot the produced histograms:

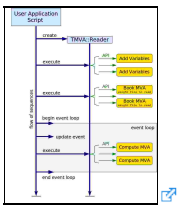
```
.ls
MVA_Fisher.Draw()
MVA_Likelihood.Draw()
MVA_LikelihoodD.Draw()
```

The application phase using the Reader

Let's look at the application script:

```
emacs TMVApplication.C
```

Discussion of main application steps:



1. Instantiation of the **Reader**
2. Registration of the input variables
3. Booking of the classifiers via weight files
4. Run the event loop and request classifier response for each event (special case: the *Rectangular Cut* classifier)

The application phase using *standalone C++ classes*

These classes are automatically created during the classifier training, and are fully self-contained. The classes are ROOT-independent. Let's look at the application script:

```
emacs ClassApplication.C
```

and

```
root -l ClassApplication.C\(\"Fisher,Likelihood,LikelihoodD\"\)
```

The script prints error messages to standard output that can be savely ignored. A version of the macro without errors can be found here.

Discussion of main application steps:

1. Create vector with input variable names
2. Load C++ class for classifier response
3. Instantiate classifier response class object
4. Run the event loop and request classifier response for each event

Note that C++ standalone classes are not available for all classifiers. Please consult the manual for a list of classifiers that support standalone classes.

Discussion

A reminder of useful discussion topics

- Strengths and weaknesses of the classifiers
- How good are the default configuration settings of the classifiers ?
- How much training statistics is required ?
- How to avoid overtraining - is independent validation required ?
- Is data preprocessing, such as decorrelation always useful ?

- How to treat systematic uncertainties (in the training, in the application) ?
- Better use the Reader or the standalone C++ classes for the application ?

Links

- TMVA Home
- TMVA Download
- TMVA Users Guide
- TMVA Sourceforge Project Page

TMVA Web Utilities

- - advanced search
- WebTopicList - all topics in alphabetical order
- WebChanges - recent topic changes in this web
- WebNotify - subscribe to an e-mail alert sent when topics change
- WebRss, WebAtom - RSS and ATOM news feeds of topic changes
- WebStatistics - listing popular topics and top contributors
- WebPreferences - preferences of this web

-- AndreasHoecker - 18 Jun 2007, Modified 28 Dec 2010 -- EckhardVonToerne - 28-Dec-2010

This topic: TMVA > WebHome
Topic revision: r24 - 2019-09-01 - ShuitingXin