ROOT multivariate analysis package (TMVA)

http://tmva.sourceforge.net

TMVA package

- The TMVA package performs a multivariate analysis on a number of input variables and returns a single MVA variable
- TMVA is included in ROOT versions 5.30 and above, the development version can be found at **`http://sourceforge.net/projects/tmva`**
- All source files and examples can be found under **`$ROOTSYS/tmva`**
- The typical TMVA analysis is divided into:
  - TMVAClassification: training and testing of a MVA method with included data set (**`$ROOTSYS/tmva/test/TMVAClassification.cxx`**)
  - TMVAClassificationApplication: applying MVA cut to a data set (**`$ROOTSYS/tmva/test/TMVAClassificationApplication.cxx`**)
- User guide: **`http://tmva.sourceforge.net/docu/TMVAUsersGuide.pdf`**

Preparing for MVA

- Example MVA:
  `http://sabotin.ung.si/~gkukec/gkukec/tmva/tmva_example.tar.gz`
- `$TMVAEX` will from now on designate the folder where the above example is extracted to
- The above example trains and tests a neural network method on a data set (`$TMVAEX/example.root`) → check the contents of the file by opening a TBrowser in ROOT
- The data set should be saved in a ROOT file, where a tree holds input variables:

```
TreeS1
    Variable1
    Variable2
    ...
TreeS2
    Variable1
    Variable2
    ...
...
```

Program structure (`$TMVAEX/tmva_simple.cpp`)

- At the beginning, we open two root files – one for reading input variables, the other to save the MVA output:

```
Tfile *ifile = Tfile::Open(“<input.root>”, “READ”);
Tfile *ofile = Tfile::Open(“<output.root>”, “RECREATE”);
```

- Then we create a Factory class object that will take care of training and testing (optionally, we can designate the MVA weights folder):

```
TMVA::Factory *factory = new TMVA::Factory(“<JobName>”, ofile,
“<options>”);
(TMVA::gConfig().GetIONames()).fWeightFileDir = “./weights”;
```

- Each variable that will be used in the MVA is then added (the last argument is the variable type **F** for float, **I** for integer):

```
factory->AddVariable(“<variable name”, ‘<variable type>’);
```

Program structure (`$TMVAEX/tmva_simple.cpp`)

- Each tree that will be used needs to be designated as signal or background – in the MVA, all signal trees will be combined together:

```
TTree *signalTree = (TTree*)ifile->Get(“<TreeName>”);
TTree *backgroundTree = (TTree*)ifile->Get(“<TreeName>”);
factory->AddSignalTree(signalTree, <weight>);
factory->AddBackgroundTree(backgroundTree, <weight>);
```

- Both trees are then prepared for training and testing:

```
factory->PrepareTrainingAndTestTree(“<sigSelectionCuts>”,
“<backSelectionCuts>”, “<options>”);
```

- If preselection cuts are not needed, first two arguments are left empty

- Some options:
  - **nTrain_Signal, nTrain_Background, nTest_Signal, nTest_Background:** Number of signal and background events used for training and testing – if set to 0, half of the events will be used for training and half for testing
  - **<u>SplitMode:</u>** Selection of events for training/testing (Random, Alternate, Block)

Program structure (**$TMVAEX/tmva_simple.cpp**)

- Then we select the MVA method that will be used for the analysis – more information on each can be found: **http://tmva.sourceforge.net/optionRef.html**

```
factory->BookMethod(<TMVA type>, “<TMVA name>”, “<options>”);
```

- Some method types (with names):
  - Likelihood: **TMVA::Types::kLikelihood** (Likelihood, LikelihoodD, LikelihoodPCA,...)
  - Function discrimination analysis: **TMVA::Types::kFDA** (FDA_GA, FDA_SA, FDA_MC,...)
  - Artificial neural networks: **TMVA::Types::kMLP** (MLP, MLPBNN,...), **TMVA::Types::kCFMlpANN** (CFMlpANN),...
  - Boosted decision trees: **TMVA::Types::kBDT** (BDT, BDTG, BDTB, BDTD,...)
- See **$TMVAEX/def_methods.cpp** for default options for a collection of different methods
- For classification, more than one method can be used (with additional BookMethod definitions)

Program structure (`$TMVAEX/tmva_simple.cpp`)

- Train, test and evaluate the methods, then close both files:

```
factory->TrainAllMethods();
factory->TestAllMethods();
factory->EvaluateAllMethods();
ifile->Close();
delete factory;
ofile->Close();
```

- The output file can now be opened using a TMVA GUI with:
  - `root -l ‘TMVAGui.C(“<output.root>”)’`
  - `./tmvagui <output.root>`
- Now we continue with the classification application part of the program
- Wish to apply a MVA variable cut onto the data set – we create a Reader class object and variables we will read values to:

```
TMVA::Reader *reader = new TMVA::Reader(“<option>”);
float obsvars[nrvars];
```

Program structure (`$TMVAEX/tmva_simple.cpp`)

- We add all variables we wish to apply the MVA cut to and prepare the method we used in the classification:

```
reader->AddVariable(“<variable name>”, &obsvars[0]);
...
reader->BookMVA(“<method name>”, “<path to weights XML
file>”);
```

- We now open the file that holds the trees and variables for our case and set the MVA cut (can be determined from optimal cut from GUI):

```
TFile *ifile = TFile::Open(“<input.root>”, “READ”);
double cut = <cut value>;
```

- Making a loop through all the trees, we set the address each variable (the names here must be the same as variables we gave to the reader):

```
for(...) {
   TTree *treeCur = (TTree*)ifile->Get(“<current tree name>”);
   treeCur->SetBranchAddress(“<variable name>”, &obsvars[0]);
   ...
}
```

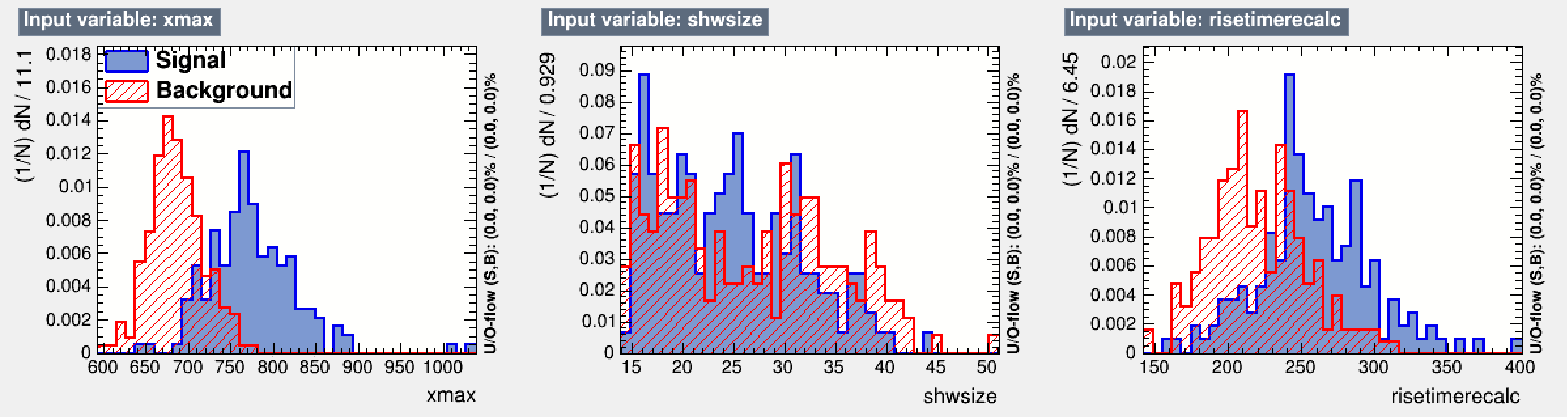
Program structure ($TMVAEX/tmva_simple.cpp)

- Inside the existing loop, we now do another loop through all events in the tree and check, if event is considered as signal or background:

```
for(...) {
   TTree *treeCur = (TTree*)ifile->Get(“<current tree name>”);
   treeCur->SetBranchAddress(“<variable name>”, &obsvars[0]);
   ...
   for(int ievt = 0; ievt < treeCur->GetEntries(); ievt++) {
      treeCur->GetEntry(ievt);
      if(reader->EvaluateMVA(“<method name>”) >= cut)
         // event is signal
      else
         // event is background
   }
}
```

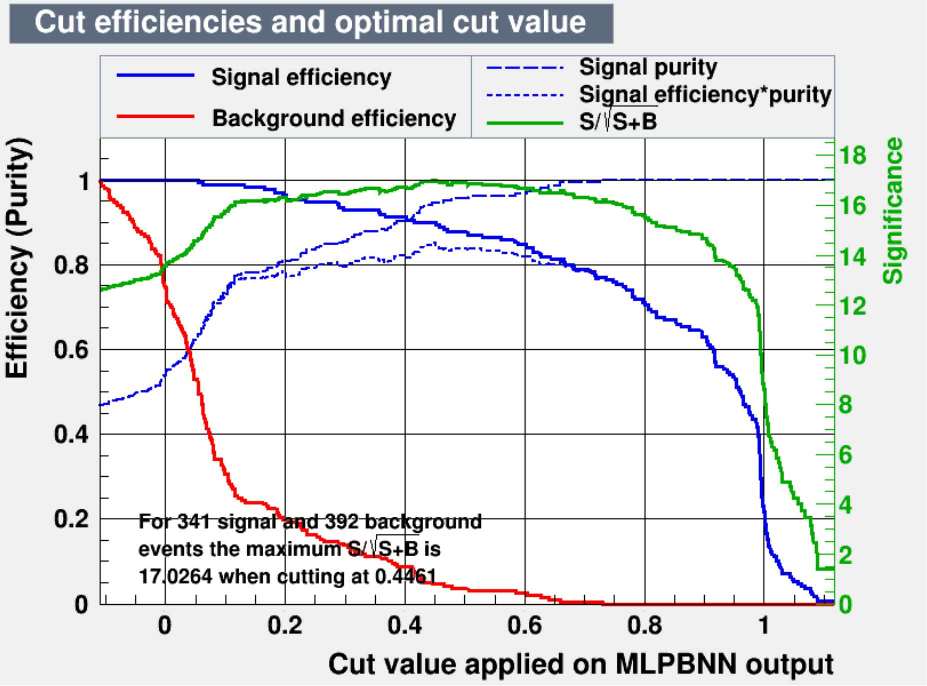
- Make sure that the used MVA method has the classifier that above the cut is signal and below the cut is background (some could have it reversed)

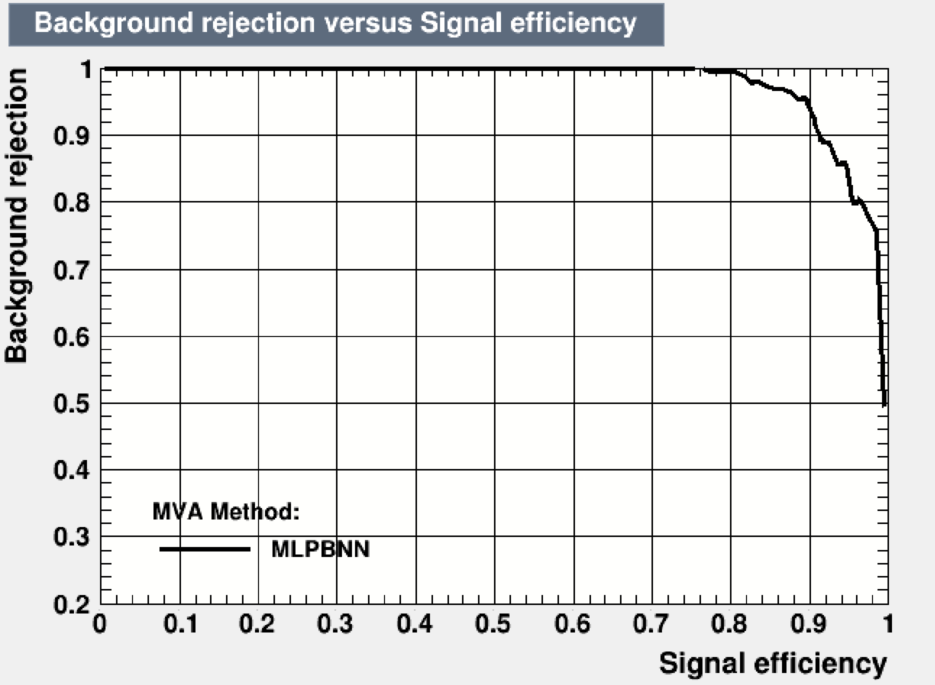
Plots from example


Input variable: xmax

Input variable: shwsize

Input variable: risetimerecalc


Cut efficiencies and optimal cut value


Background rejection versus Signal efficiency