New and modern interfaces for TMVA

Kim Albertsson

Luleå University of Technology

2018-09-03

Agenda

- New interfaces in TMVA
  - Cross validation (exists now)
  - Data ingestion (future feature)

Cross validation

TODO: Overview of valdiation, training scheme, xval.

CV interface

- New interface (similar to Factory)

```
TString options =
    "!V:AnalysisType=Classification";

TMVA::Factory factory{"<jobname>",
    dataloader, outputFile, options};
```

CV interface

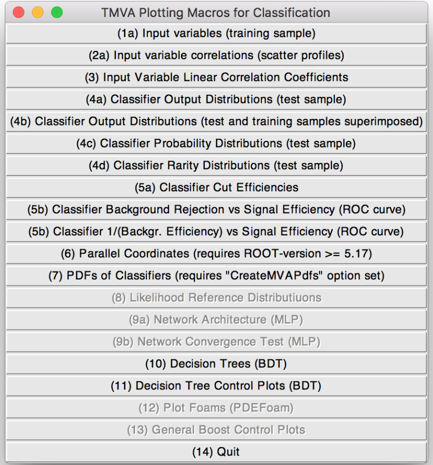
- New interface (similar to Factory)
- Integrates with TMVA GUI
- plus per fold information

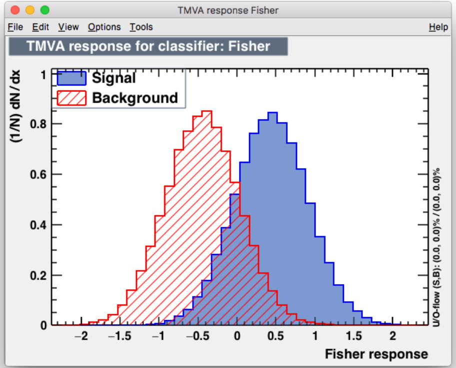
```
TString options =
    "!V:AnalysisType=Classification"
    ":NumFolds=2";

TMVA::CrossValidation cv{"<jobname>",
    dataloader, outputFile, options};
```

TMVA GUI

- Uses the same GUI as for non-cv





Error estimation

Standard cross validation procedure

- Estimate training sceme performance using cv
- Apply training scheme to entire dataset to get final model
- Avg. behaviour of final model determined by estimate

Problem: Errors on e.g. distribution histograms are correct *on average*.

Solution: CV in application

CV in application

- CV in application
- Deterministic split on uncorrelated quantity
  - Event number for physics
  - Random number at event generation

```
TMVA::DataLoader * dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddVariable("x");
dataloader->AddVariable("y");
dataloader->AddSpectator("eventId");

// ...snip...

TString options =
    "!V:AnalysisType=Classification:"
    "NumFolds=2:"
    "SplitExpr=int([eventId])%"
    "int([NumFolds])";

TMVA::CrossValidation cv{"<jobname>",
    dataloader, outputFile, options};
```

CV Conclusion

- New interface, `CrossValidation`
  - Works like factory
  - Integrates with normal workflow
  - Support “CV in application”

RDataFrame quick introdcution

ROOT 6.14 introduced RDataFrame

- High level interface for data transformations
- Efficient implementation (implicit multithreading)
- See talk of Enrico!

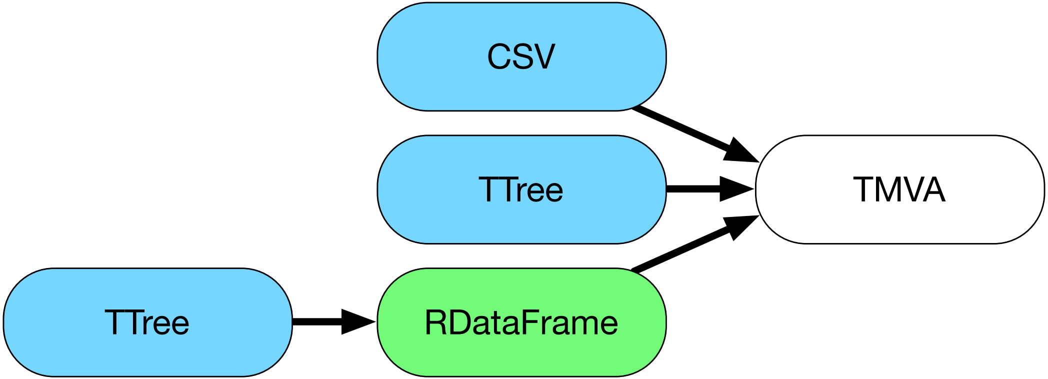
How does it integrate with TMVA?

```
RDataFrame rdf{"Tree", "data.root"};

// x and y are branches already in tree
auto df = df.Define("z", "sin(x + y)");
auto df = df.Define("q", [](){
  Double_t val = do_some_processing();
  return val;
});

// Save to a new file
df.Snapshot("NewTree", "NewFile",
            {"x", "y", "z", "q"});
```

Future ingestion interface



Data ingestion – Now

```
auto datafile = TFile::Open("data.root");
TTree * tree  = datafile->Get("Signal");

DataLoader dl{"dataset"};
dl.AddVariable("x");
dl.AddVariable("y");
dl.AddVariable("z := x + y");
dl.AddTree(tree, "Signal");

dl.PrepareTrainingAndTestTree(
    "<cut>", "<splitopt>");
```

Data ingestion – Future

```
RDataFrame rdf{"Signal", "data.root"};
auto df = df.Define("z", "x + y");

DataLoader dl{"dataset"};
dl.AddVariable("x");
dl.AddVariable("y");
dl.AddVariable("z");
dl.AddDataFrame(df, "Signal");

dl.PrepareTrainingAndTestTree(
    "<cut>", "<splitopt>");
```

Data ingestion – Future

Benefits

- Move data processing to RDataFrame
- Improved performance (multithreading)
- Expressive transformations
- “Single language”
- Further interface simplifications

```
RDataFrame rdf{"Signal", "data.root"};
auto df = df.Define("z", "x + y");

DataLoader dl{"dataset"};
dl.AddVariable("x");
dl.AddVariable("y");
dl.AddVariable("z");
dl.AddDataFrame(df, "Signal");

dl.PrepareTrainingAndTestTree(
    "<cut>", "<splitopt>");
```

“Single language”

```
RDataFrame rdf{"Signal",
   "data.root"};
auto df = df.Define("z", "x + y");

DataLoader dl{"dataset"};
dl.AddVariable("x");
dl.AddVariable("y");
dl.AddVariable("z");
dl.AddDataFrame(df, "Signal");

// ...snip...
```

```
RDataFrame rdf{"Signal",
   "data_preprocessed.root"};

DataLoader dl{"dataset"};
dl.AddVariable("x");
dl.AddVariable("y");
dl.AddVariable("z");
dl.AddDataFrame(df, "Signal");

// ...snip...
```

One way to express data transformations

- Easily change transformation setup
  - Removes boilerplate
- Benefits in application as well

Further interface simplifications

- Repeated use of `dl.AddVariable("x");`
- Information already in `RDataframe`
- Default initialisation of Variables
  - Unless overridden by you!

```
RDataFrame df{"Signal",
   "data_preprocessed.root"};

DataLoader dl{"dataset"};
dl.AddDataFrame(df, "Signal",
    {"x", "y", "z"});

// ...snip...
```

Conclusion – Ingestion

With the new ingestion interface we can go from

```cpp
auto datafile = TFile::Open("data.root");
TTree * tree  = datafile->Get("Signal");

DataLoader dl{"dataset"};
dl.AddVariable("x");
dl.AddVariable("y");
dl.AddVariable("z := x + y");
dl.AddTree(tree, "Signal");

// ...snip...
```

Conclusion – Ingestion

To

- Providing the same functionality
- Potentially better speed
- Simpler code in training/testing and application

```cpp
RDataFrame rdf{"Signal", "data.root"};
auto df = df.Define("z", "x + y");

DataLoader dl{"dataset"};
dl.AddDataFrame(df, "Signal",
    {"x", "y", "z"});

// ...snip...
```

Conclusion

- K-Folds cross validation, since ROOT 6.14
  - With support for “cv in application”
- Better data ingestion
  - Coming to a ROOT near you in (6.16?)

Blank

Supplementary

```
TMVA::DataLoader * dataloader =
    new TMVA::DataLoader("dataset");

// Declare tree structure
dataloader->AddVariable("x", 'F');
dataloader->AddVariable("y", 'F');
dataloader->AddSpectator("eventId", 'F');

// Add `TTree`s
dataloader->AddSignalTree(getSigTree(), 1.0);
dataloader->AddBackgroundTree(getBkgTree(), 1.0);

// We have the possibility to set aside some events
// in a separate data set (currently unused)
dataloader->PrepareTrainingAndTestTree("", "",
    "nTest_Signal=1:nTest_Background=1"
    ":SplitMode=Block:!V");
```