Robust regression and model compression +

AliNDFunctionInterface

TMVA wrapper

Marian Ivanov, Martin Kroesen

Outlook

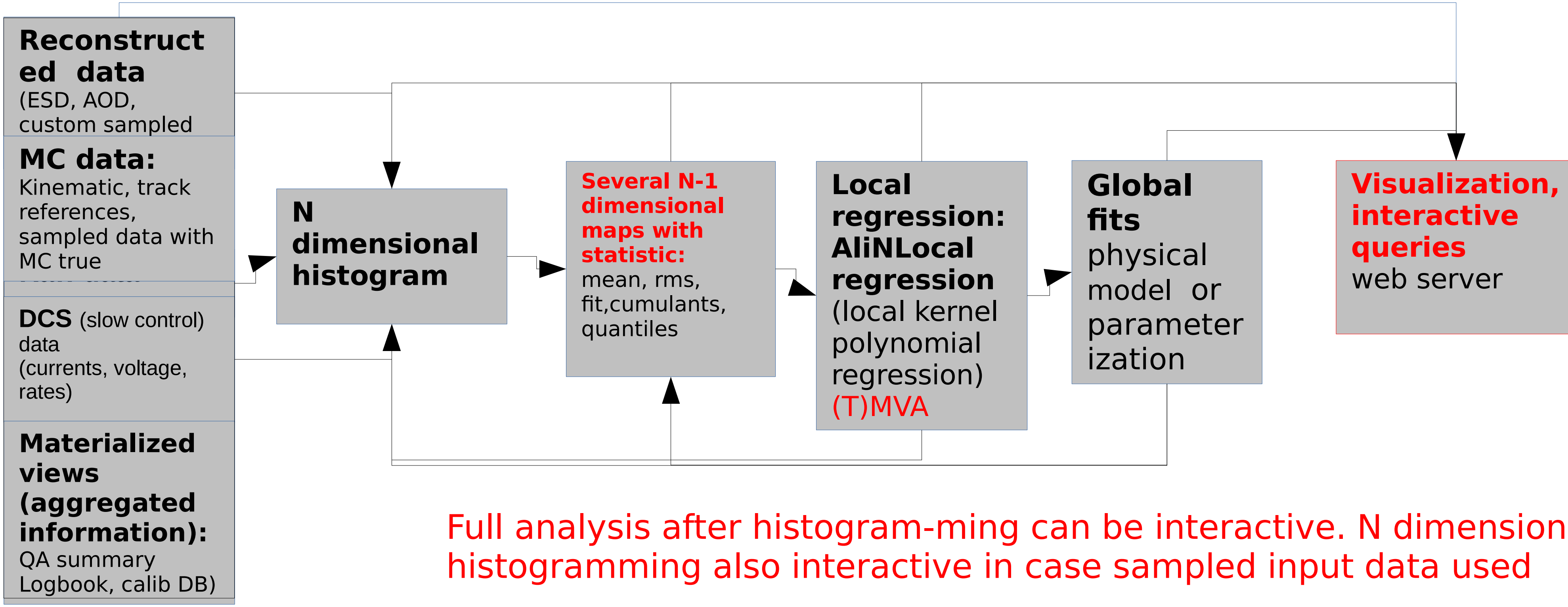
N dimensional analysis pipeline in ALICE

(T)MVA interface wrapper

- quantiles
- prediction/confidence intervals
- model compression

TMVA wrapper

ALICE N-Dimensional analysis pipeline



Full analysis after histogram-ming can be interactive. N dimensional histogramming also interactive in case sampled input data used

Visualization/query - generic support exist (libSTAT)

- TFormula (TF1, TTreeFormula, AliTreeFormula), TStatToolkit (Metadata describing input data)
- AliTreePlayer
- AliPainter and AliDrawStyle(CSS)

Declarative analysis
Integration with RDataFrame (JIT, parallelism) under development

New extensions:

- CSS library
- Multidimensional drawing support (Histogram, TFormula, TTree)
- Web server (static and jupyter interactive, bokeh)

AliNDFunctionInterface: TMVA wrapper in ALICE analysis framework AliRoot

- **Simple and compact user interface**
  - similar to TTree::Draw and Histogram::Fit queries
- Store all the data as ROOT objects in ROOT files (instead of weight files, no xml files)
  - possibility to store data in Alice calibration DB
- Easy usage providing TFormula/TTreeformula interface
  - possibility to combine/normalize/operate with other formulas (other TMVA, global fits, NDimensional local tables (e.g AliNDLocalRegression object))

Additional functionality to be provided:

- Local error estimates (prediction/confidence intervals), local robust estimators (bootstrapping)
- Quantile regression
- Combined/weighted evaluation, caching and model compression
  - optimally - compressed results to agree with model within the prediction
- **New Python wrapper under development**

Quantile regression and prediction intervals

Most estimators during prediction return E(Y|X), which can be interpreted as the answer to the question, what is the expected value of your output given the input?

- Quantile methods, return y at q for which F(Y=y|X)=q where q is the percentile and y is the quantile.

Quantile regression forest:

- https://scikit-garden.github.io/examples/QuantileRegressionForests/

Deep Quantile Regression

- Quantile Regression Loss function
  - March 2018
- https://towardsdatascience.com/deep-quantile-regression-c85481548b5a

Confidence/prediction intervals

Confidence Intervals for Scikit Learn Random Forests

- http://contrib.scikit-learn.org/forest-confidence-interval/
- https://github.com/scikit-learn-contrib/forest-confidence-interval
- forestci package
  - This package adds to scikit-learn the ability to calculate confidence intervals of the predictions generated from scikit-learn

Neural network prediction:

- 1: Delta method
- 2: Bayesian method
- 3: Mean variance estimation
- 4: Bootstrap
- For the moment, not aware of the standard procedure …

Bootstrap approach

- provides “prediction” intervals for all methods

Combination of the different methods based on the local properties

- **Problem to solve - significant extrapolation error in some local regions**
    - e.g KNN conservative extrapolation less sensitive than BDT or MLP
- local error estimate using bootstrap
    - **combine methods using weighted average**

Speed up MVA evaluation using caching:

- In case of small number of dimensions tabulated kernel local regression (AliNDLocalRegression)
- In case of multidimensional problem cache “results” and feed it into fast methods (Neural network?)

Model compression - KERAS

https://www.cs.cornell.edu/~caruana/compression.kdd06.pdf

An ensemble is a collection of models whose predictions are combined by weighted (local) averaging or voting.

- Well known ensemble methods include bagging[2], boosting [14], random forests[3], Bayesian averaging [9] and stacking [17].

Ensembles - disadvantage: many ensembles are large and slow

Goal: compress the function that is learned by a complex model into a much smaller, faster model that has comparable performance (within prediction intervals)

- To be used in reconstruction
- The main idea behind model compression is to use a fast and compact model to approximate the function learned by a slower, larger, but better performing model.
- Unlike the true function that is unknown, the function learned by a high performing model is available and can be used to label large amounts of pseudo data.

Current assumption: KERAS to be used as an compression model.

Problem - current ROOT interface too slow. Need to speed up evaluation.

- lwtnn library investigated

AliNDFunctionInterface

AliNDFunctionInterface TMVA usage

Step 1: Register methods

Step 2: Register factory

Step 3: Fit Method(s)

Step x: Compress model

Step 4: Load /Register reader

Step 5: Eval MVA as an formula

Step 1: Register (regression/classification) methods

AliNDFunctionInterface::registerMethod() with parameters:

- std::string method: assign name
- std::string content: method registration string used in TMVA
- TMVA::Types::EMVA: type within TMVA

Example:

```
AliNDFunctionInterface::registerMethod("MLPBNN", "H:!
   V:NeuronType=tanh:VarTransform=N:NCycles=20:HiddenLayers=N+5:Te
   stRate=5:TrainingMethod=BFGS:UseRegulator",TMVA::Types::kMLP);
```

Experts provide default methods (for particular categories of problems)

- In example experts provided function MLPBNN as equivalent of other function (e.g gauss or exp functions)

Step 2: Register factory

AliNDFunctionInterface::registerFactory() with parameters:

- std::string factory: assign name
- std::string content: string used in PrepareTrainingAndTestTree() to define sample size etc.
- If empty or unavailable: use default settings

Example:

```
AliNDFunctionInterface::registerFactory("testFactory",
   "nTrain_Regression=50%:nTest_Regression=50%:SplitMode=Random:No
   rmMode=NumEvents:!V");
```

Experts provide default methods

Step 3: Fit Method - Regression

Training and testing for Regression provided by:
AliNDFunctionInterface::FitMVARegression()

Parameters:

- const char * output: output root file and directory separated by "#"
- TTree *tree: input tree
- const char *varFit: variable to fit in regression separated by ":"
- TCut cut
- const char * variables: variables used for training
- const char *methods: previously registered method names
- const char * factoryString: previously registered strings to define sample for training and testing

```
Int_t  AliNDFunctionInterface::FitMVARegression(const char *
output, TTree *tree, const char *varFit, TCut cut, const char *
variables, const char *methods,const char * factoryString);
```

```
AliNDFunctionInterface::FitMVARegression("TMVA_RegressionOutput.root#tes
   t","regTree","fvalue","var1>3","var1:var2","MLPBNN:BDT","test_factory")
   ;
```

Step 3: Fit Method Regression

Interface:

```
Int_t  AliNDFunctionInterface::FitMVARegression(const char *
   output, TTree *tree, const char *varFit, TCut cut, const char *
   variables, const char *methods,const char * factoryString);
```

Example:

```
AliNDFunctionInterface::FitMVARegression("TMVA_RegressionOutput.ro
   ot#test",regTree,"fvalue","var1>3","var1:var2","MLPBNN:BDT","te
   st_factory");
```

Step 4: Load /Register reader

Load TMVA Reader via AliNDFunctionInterface::LoadMVAReader()

Weights from xml file and variables are stored to root file before and are loaded via LoadMVAReader to apply the methods to independent data

Parameters:

- Int_t id: to identify booked reader
- const char * inputFile: root file where parameters are stored
- const char *method: method to be booked
- const char *dir: directory of stored method

Example:

```
AliNDFunctionInterface::LoadMVAReader(0,"TMVA_Output.root","PyRand
   omForest","cleanK0");
```

Step 5: Eval MVA as an formula

Evaluate MVA (variadic function)

- TMVA mathod to be loaded (regisetered using id) before
- `EvalMVA(int id, T v, Args... args);` (Regression)
- `EvalMVAClasification(int id, T v, Args... args);`(Classification)

Parameters: Reader ID (usually registered using hash value) and variables

Example usage:

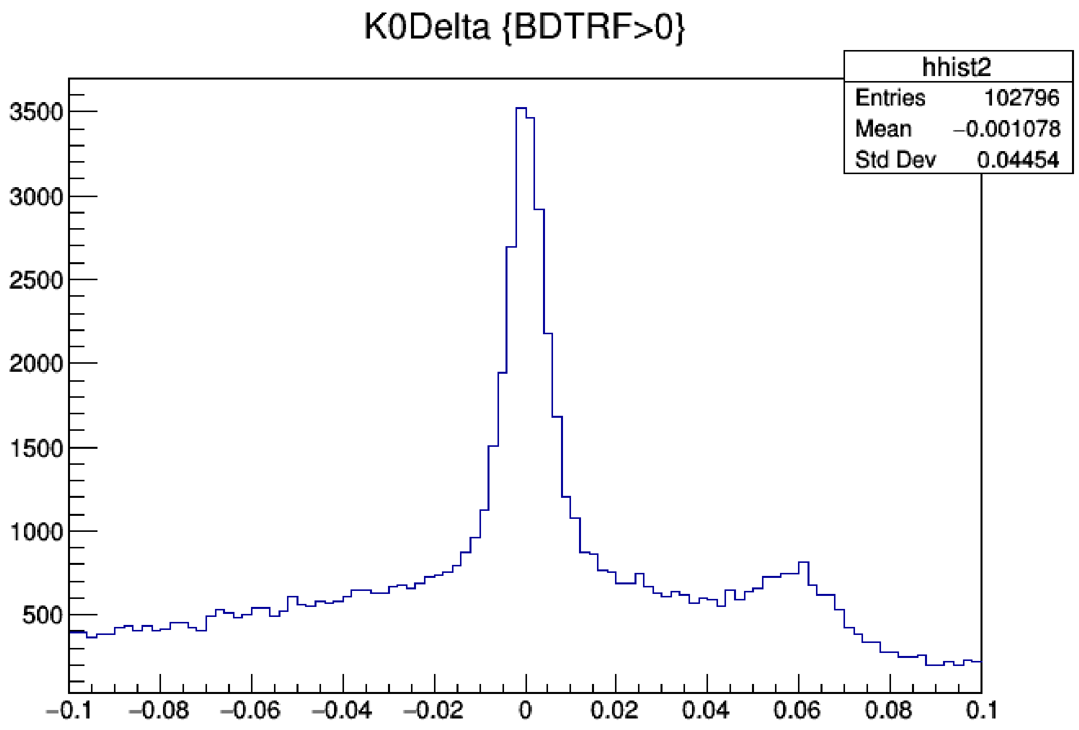
- register TMVA as an function in tree and use it in queries

```
treeMVA -
   >SetAlias("BDTRF","AliNDFunctionInterface::EvalMVAClassificatio
   n(2,errChi2,v0ErrM,pointAngleN,mpt,armV0S,fRr+0)");


treeMVA->Draw("K0Delta>>hist(100,-0.1,0.1)","BDTRF>0")
```

Step 5: Eval MVA as an formula

Example:

```
treeMVA -
   >SetAlias("BDTRF","AliNDFunctionInterface::EvalMVAClassificatio
   n(2,errChi2,v0ErrM,pointAngleN,mpt,armV0S,fRr+0)");


treeMVA->Draw("K0Delta>>hist(100,-0.1,0.1)","BDTRF>0")
```



Conclusion

N dimensional analysis pipeline in ALICE

(T)MVA interface wrapper

- quantiles
- prediction/confidence intervals
- model compression

TMVA wrapper

- MVA regression almost as simple as gaussian fit

New MVA wrapper

- Work in progress

Backup

Step 3: Fit Method - Classification

Training and testing for classification provided by:
AliNDFunctionInterface::FitMVAClassification()

Parameters:

- const char * output: output root file and directory separated by “#”
- const char *input_trees: root file and tree separated by “#” - the trees define the classes
- const char *cuts: cuts for each tree separated by “:”. If only one tree available, the cuts define the classes
- const char * variables: variables used for training
- const char *methods: previously registered method names
- const char * factoryString: previously registered strings to define sample for training and testing

Example:

```
Int_t  AliNDFunctionInterface::FitMVAClassification(const char *
   output, const char *inputTrees, const char *cuts, const char *
   variables, const char *methods, const char * factoryString)
```

```
AliNDFunctionInterface::FitMVAClassification("TMVA_Output.root#clean
   K0","TMVAInput.root#MVAInput","cleanK0:isBackground","errChi2:v0Er
   rM:pointAngleN:mpt:armV0S:fRr","BDTRF:PyRandomForest:KNN","");
```

Interface:

```
Int_t  AliNDFunctionInterface::FitMVAClassification(const char *
   output, const char *inputTrees, const char *cuts, const char *
   variables, const char *methods, const char * factoryString)
```

Example one tree:

```
AliNDFunctionInterface::FitMVAClassification("TMVA_Output.root#clea
   nK0","TMVAInput.root#MVAInput","cleanK0:isBackground","errChi2:v
   0ErrM:pointAngleN:mpt:armV0S:fRr","BDTRF:PyRandomForest:KNN","")
   ;
```

Example multiple trees:

```
AliNDFunctionInterface::FitMVAClassification("TMVA_Output.root#clea
   nK0","TMVAInput.root#MVAInput1:TMVAInput.root#MVAInput2","","err
   Chi2:v0ErrM:pointAngleN:mpt:armV0S:fRr","BDTRF:PyRandomForest:KN
   N","");
```