CERN Summer Student 2018
Project Report

Cross Validation Improvements in TMVA

Mohammad Uzair
mohduzair9@gmail.com

Supervisors:

Dr. Lorenzo Moneta
Mr. Hans Kim Albertsson Brann

August 29th, 2018

TABLE OF CONTENTS

PROJECT PROPOSAL

TMVA is a machine learning framework primarily targeted at enabling physics research and is distributed as part of ROOT. This project sets out to improve the validation tools, and cross validation of TMVA. Cross validation is an important tool for making the most out of limited data sets trading increased data efficiency for decreased computation efficiency. This is relevant to HEP applications since experiments are limited by the prohibitive cost of running detector and reconstruction simulation. Tasks include: implement and evaluate the performance of different cross validation splitting functions; extend the current implementation to seamlessly handle nested cross validation; evaluate and compare the performance of the TMVA implementation to industry standard libraries using physics datasets; parallelise training and evaluation.

TASKS ASSIGNED

- ❏ To update and design a new tutorial on TMVA Cross Validation using jupyter notebook and make it available online through SWAN.

- ❏ To improve the presentation of the newly designed tutorial notebook by extending the plotting functionality i.e. to introduce a new feature of drawing an average ROC curve.

- ❏ To improve the cross validation fold generation especially targeting unbalanced datasets i.e. to implement a new technique for making cross validation folds known as stratified splitting.

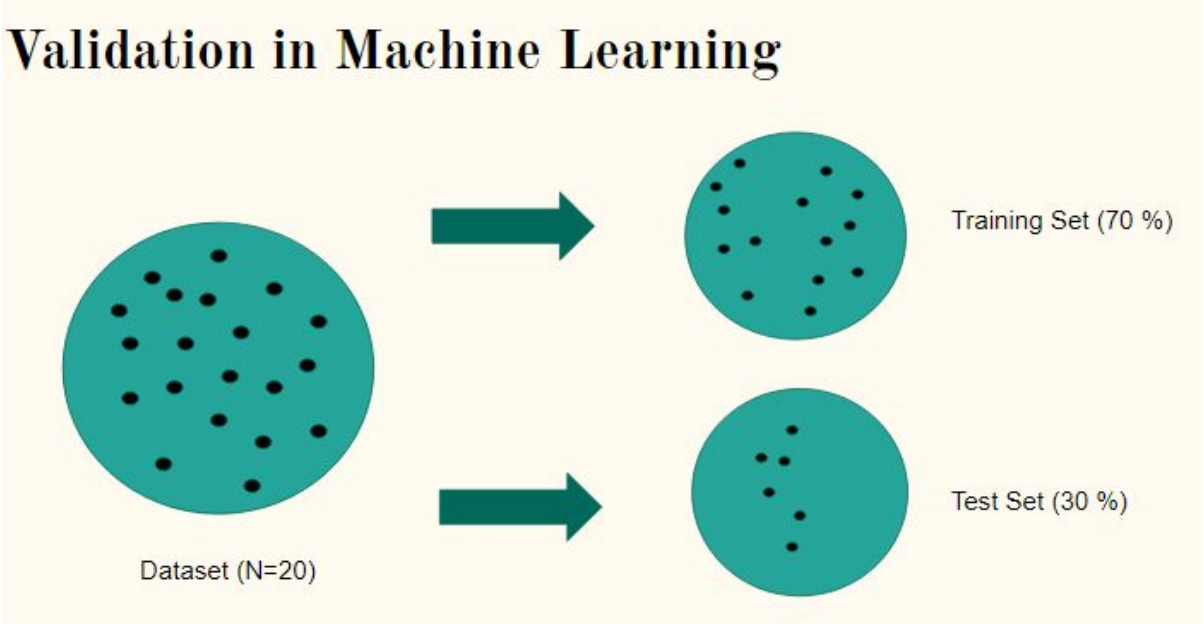
BACKGROUND

VALIDATION IN MACHINE LEARNING

Validation is a technique in machine learning by which we split our dataset into train and test sets. Training set is the set of data points which the model can see and learn from while test set is used to evaluate this model.

Validation does not only allow us to estimate the expected error of the trained model but also provides us an honest assessment of the performance of the model that is trained.

For instance, if we have 20 data points in the dataset and we decide to split it up into train set (70%) and the test set (30%) then out of these 20 points, 14 points will end up in train set and the rest of 6 points will comprise the test set.



CROSS VALIDATION

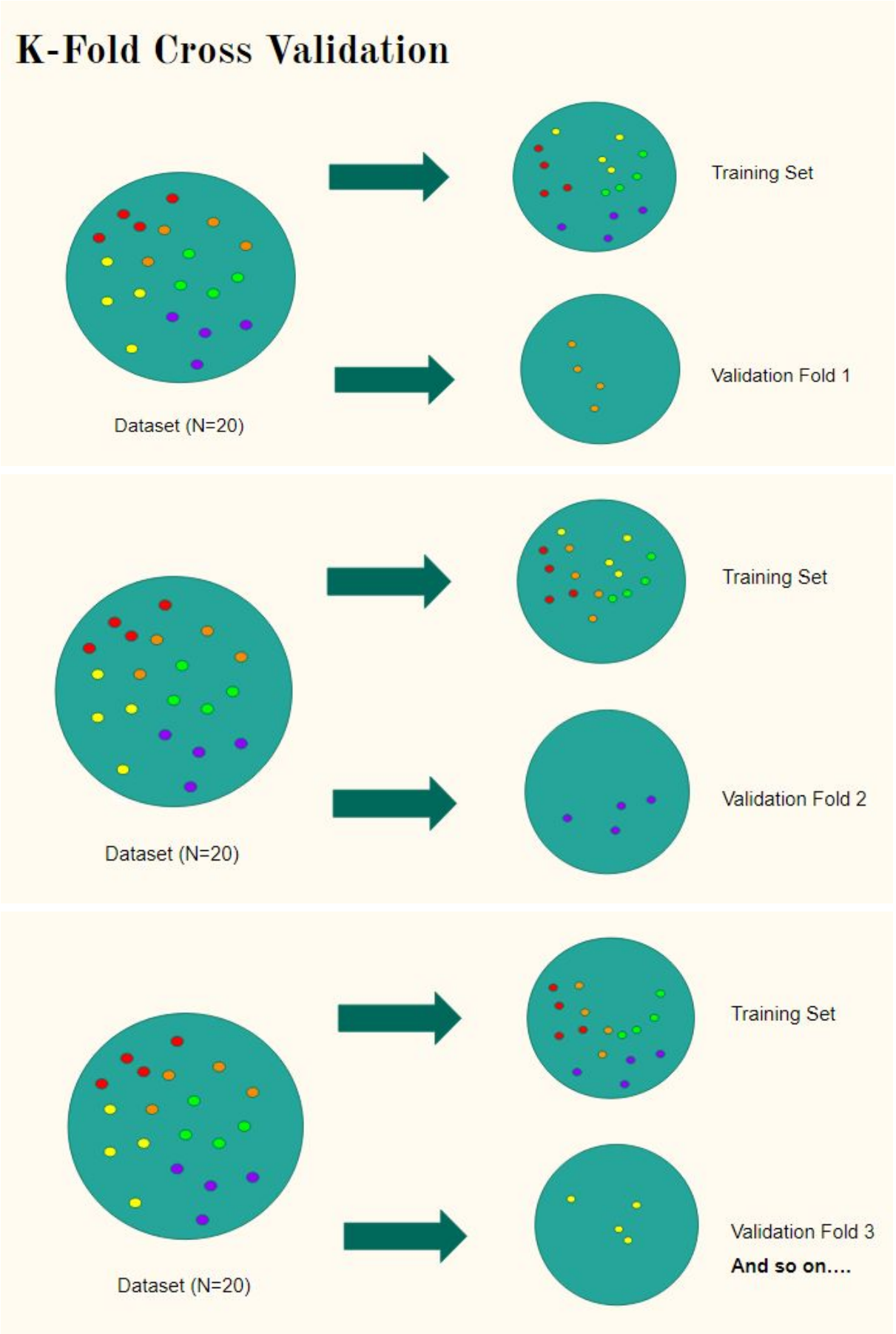
Problem with Validation is that the error estimate is highly variable as is dependent on the percentage of dataset chosen as the test set. If we use a large amount of our data for testing then the error estimate can be very precise but the error will be high because a small subset of the data will be used for training. Similarly, if we use a small amount of data for testing then the test error might be much lower but it not be precise as we did not use enough data points for testing. This problem especially arises when we have small datasets. In this case we ideally want to use all the data for training and all the data for testing. One method to do something like this is Cross Validation.

K-FOLD CROSS VALIDATION

We essentially need a method in which we can use all the data for training and all of them for testing. K-Fold Cross Validation provides does exactly that.

In K-Fold Validation we split our data into K subsets. Then for each of the K subsets we use them once as a test set and use all the other K-1 sets for training. After we have used all the K subsets once for testing we average over all the errors to get an error estimate of our trained model. This way we are able to use full dataset for testing and full dataset for training.

For instance, if we have 20 points in our dataset and we decide to take K=5 then we will have 5 folds with 4 data points in each. Then we will use folds 2, 3, 4 and 5 for training and fold 1 for testing. After this we will use folds 1, 3, 4 and 5 for training and fold 2 for testing and so on.


K-Fold Cross Validation
Dataset (N=20)
Training Set
Validation Fold 1
Dataset (N=20)
Training Set
Validation Fold 2
Dataset (N=20)
Training Set
Validation Fold 3
And so on....

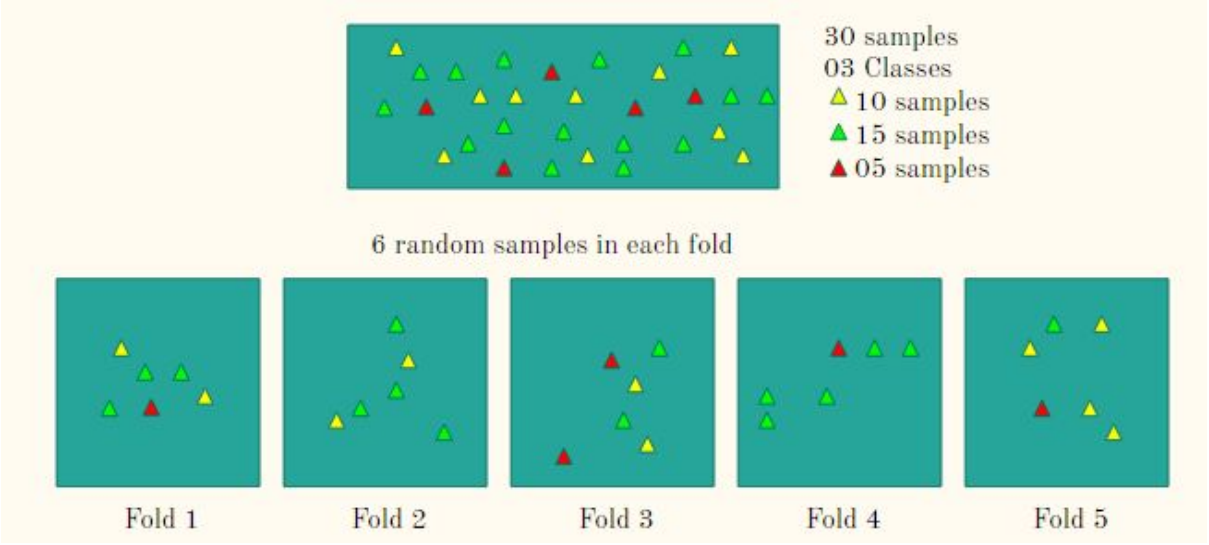
STRATIFIED K-FOLD CROSS VALIDATION

In some cases there can be large imbalance between the distribution of classes in the dataset. Some classes may have very small amount of data points while some may have unusually large amount of data points.

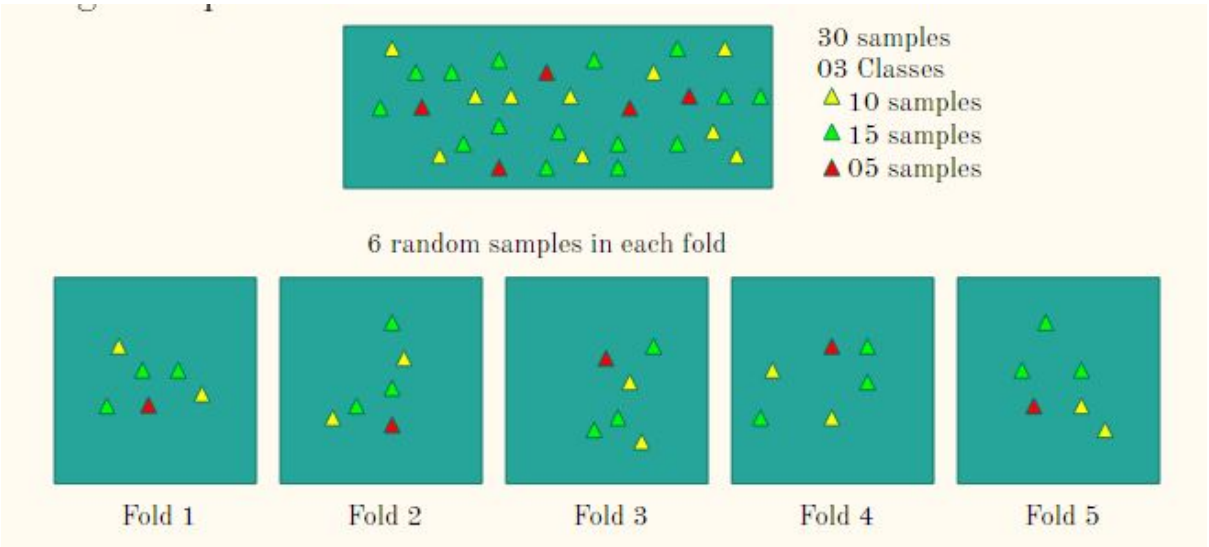
The problem then arises that if we use random splitting for fold generation then, some folds can have a different ratio of distribution than the whole dataset. This might increase the error of some of the folds as they are not representing the data distribution as it was.

To rectify this problem we use a technique called Stratified splitting for dividing our data into the folds. The data in this technique, is split randomly but making sure that it follows the actual distribution of the whole dataset.

For instance, if we have 20 data points and we split them using random splitting then our distribution can be as follows which is for sure not a true representation of the distribution of the actual dataset.



But on the other hand, using stratified splitting solves this problem efficiently while maintaining the randomness of the fold generation.[1]

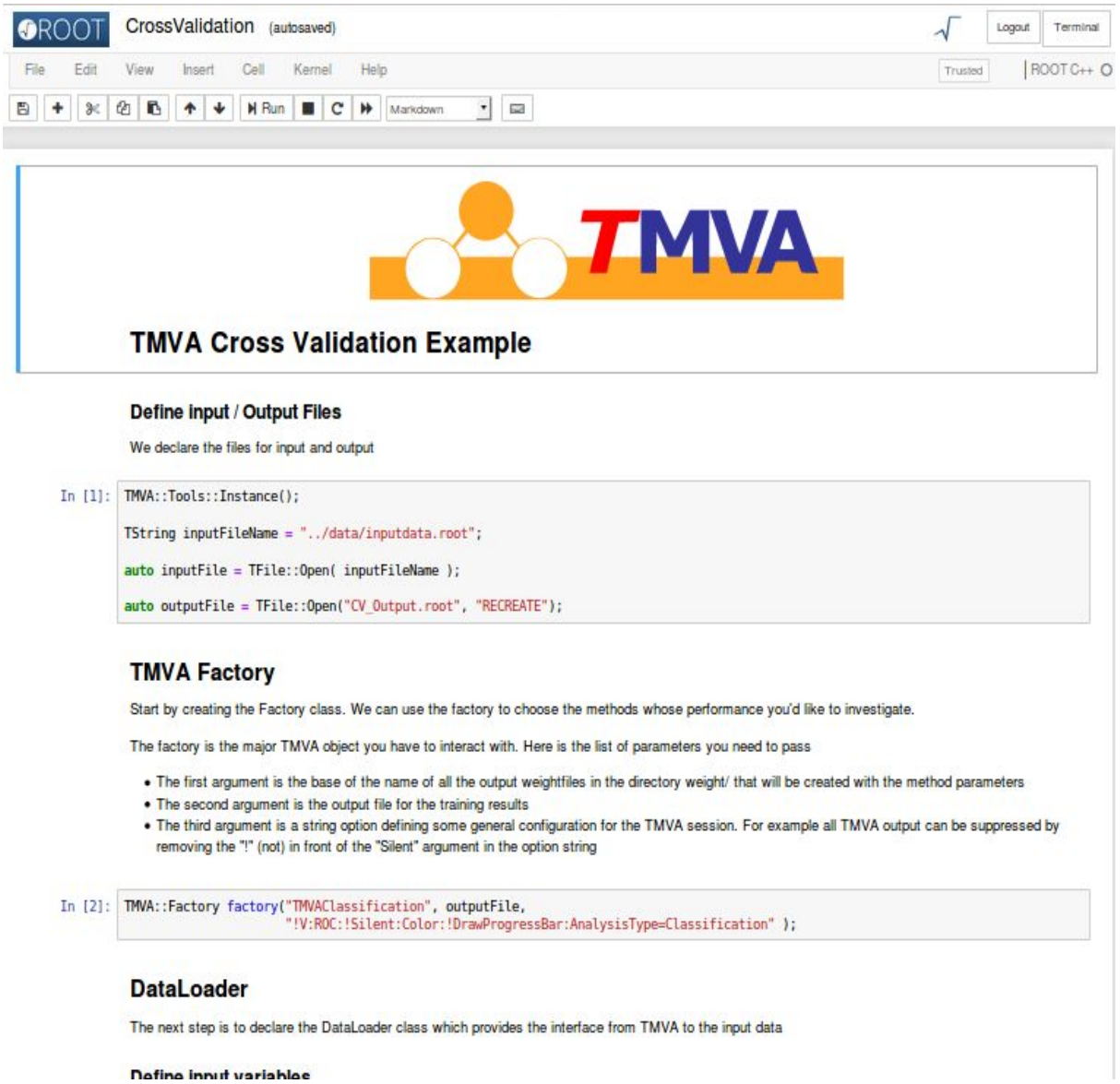


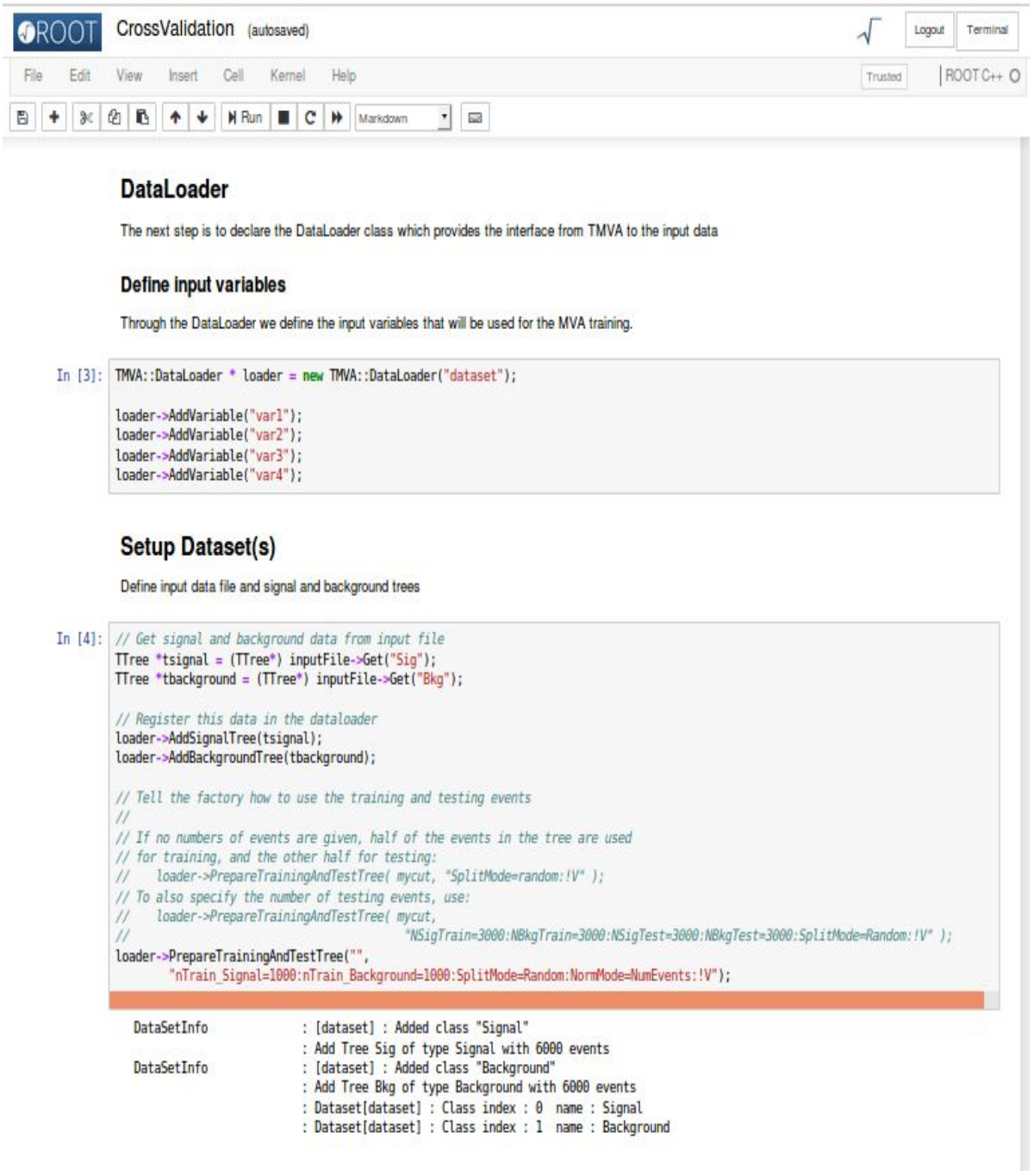
[1] https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f

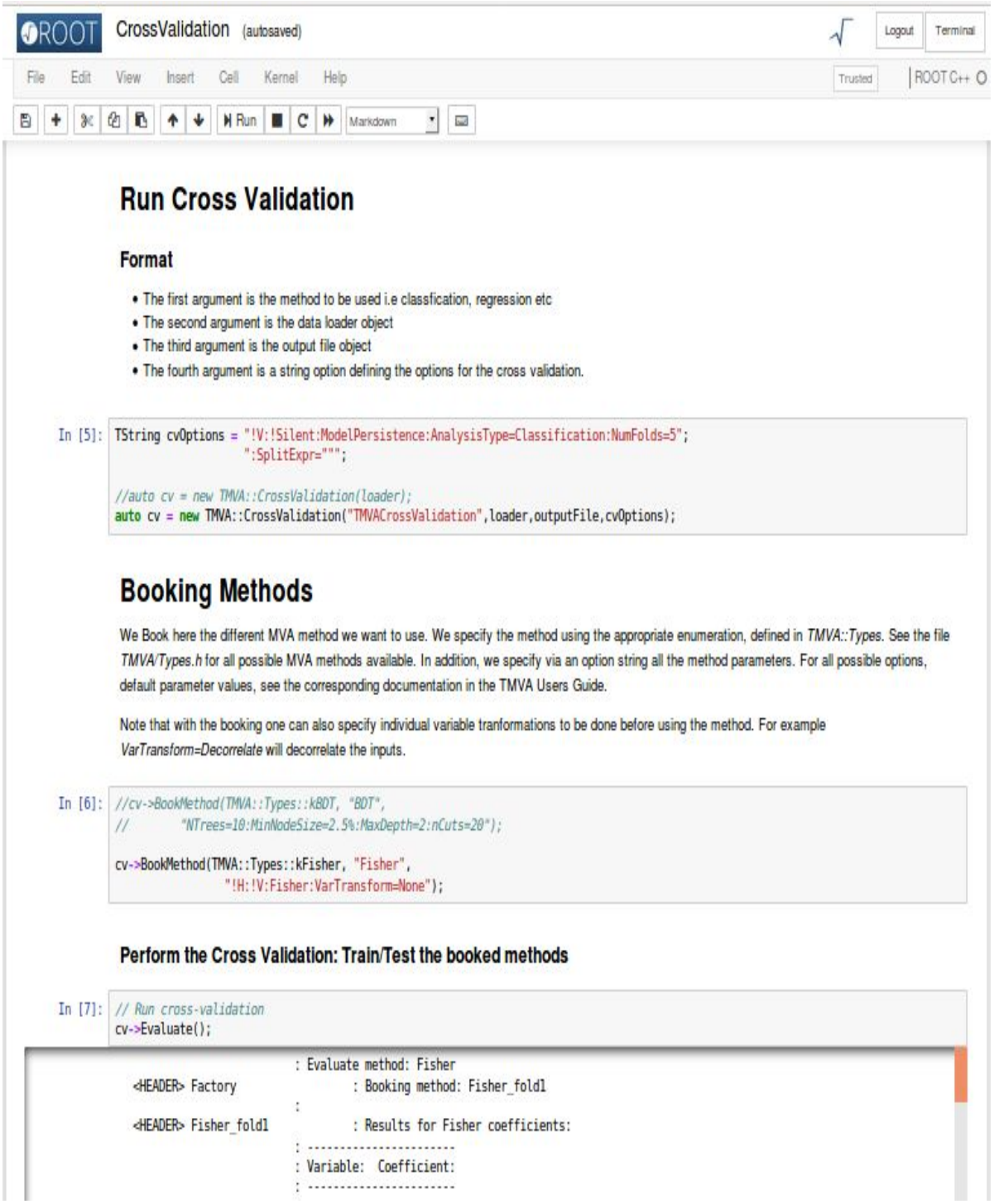
IMPLEMENTATION OF THE TASKS

TUTORIAL INTRODUCING CROSS VALIDATION

Old and outdated tutorial on TMVA Cross Validation was updated to introduce the Cross Validation in TMVA to the users. This tutorial is made and implemented on jupyter notebook using C++ kernel and is also made available through SWAN for online viewing and running. Screenshots of this tutorial is shown below:




ROOT
CrossValidation (autosaved)
Logout
Terminal
File Edit View Insert Cell Kernel Help
Trusted
ROOT C++
Run
Markdown
DataLoader
The next step is to declare the DataLoader class which provides the interface from TMVA to the input data
Define input variables
Through the DataLoader we define the input variables that will be used for the MVA training.
In [3]:
TMVA::DataLoader * loader = new TMVA::DataLoader("dataset");
loader->AddVariable("var1");
loader->AddVariable("var2");
loader->AddVariable("var3");
loader->AddVariable("var4");
Setup Dataset(s)
Define input data file and signal and background trees
In [4]:
// Get signal and background data from input file
TTree *tsignal = (TTree*) inputFile->Get("Sig");
TTree *tbackground = (TTree*) inputFile->Get("Bkg");
// Register this data in the dataloader
loader->AddSignalTree(tsignal);
loader->AddBackgroundTree(tbackground);
// Tell the factory how to use the training and testing events
//
// If no numbers of events are given, half of the events in the tree are used
// for training, and the other half for testing:
//   loader->PrepareTrainingAndTestTree(mycut, "SplitMode=random:!V");
// To also specify the number of testing events, use:
//   loader->PrepareTrainingAndTestTree(mycut,
//   "NSigTrain=3000:NBkgTrain=3000:NSigTest=3000:NBkgTest=3000:SplitMode=Random:!V");
loader->PrepareTrainingAndTestTree("",
       "nTrain_Signal=1000:nTrain_Background=1000:SplitMode=Random:NormMode=NumEvents:!V");
DataSetInfo : [dataset] : Added class "Signal"
: Add Tree Sig of type Signal with 6000 events
DataSetInfo : [dataset] : Added class "Background"
: Add Tree Bkg of type Background with 6000 events
: Dataset[dataset] : Class index : 0 name : Signal
: Dataset[dataset] : Class index : 1 name : Background


ROOT
CrossValidation (autosaved)
Logout
Terminal
File Edit View Insert Cell Kernel Help
Trusted
ROOT C++
Run
Markdown
Run Cross Validation
Format
• The first argument is the method to be used i.e classfication, regression etc
• The second argument is the data loader object
• The third argument is the output file object
• The fourth argument is a string option defining the options for the cross validation.
In [5]:
TString cvOptions = "!V:!Silent:ModelPersistence:AnalysisType=Classification:NumFolds=5";
":SplitExpr="";
//auto cv = new TMVA::CrossValidation(loader);
auto cv = new TMVA::CrossValidation("TMVACrossValidation",loader,outputFile,cvOptions);
Booking Methods
We Book here the different MVA method we want to use. We specify the method using the appropriate enumeration, defined in TMVA::Types. See the file TMVA/Types.h for all possible MVA methods available. In addition, we specify via an option string all the method parameters. For all possible options, default parameter values, see the corresponding documentation in the TMVA Users Guide.
Note that with the booking one can also specify individual variable tranformations to be done before using the method. For example VarTransform=Decorrelate will decorrelate the inputs.
In [6]:
//cv->BookMethod(TMVA::Types::kBDT, "BDT",
// "NTrees=10:MinNodeSize=2.5%:MaxDepth=2:nCuts=20");
cv->BookMethod(TMVA::Types::kFisher, "Fisher",
"!H:!V:Fisher:VarTransform=None");
Perform the Cross Validation: Train/Test the booked methods
In [7]:
// Run cross-validation
cv->Evaluate();
: Evaluate method: Fisher
<HEADER> Factory : Booking method: Fisher_fold1
:
<HEADER> Fisher_fold1 : Results for Fisher coefficients:
: ----------------------
: Variable: Coefficient:
: ----------------------

ROOT CrossValidation (autosaved)

Logout Terminal

File Edit View Insert Cell Kernel Help Trusted ROOT C++

Run Markdown

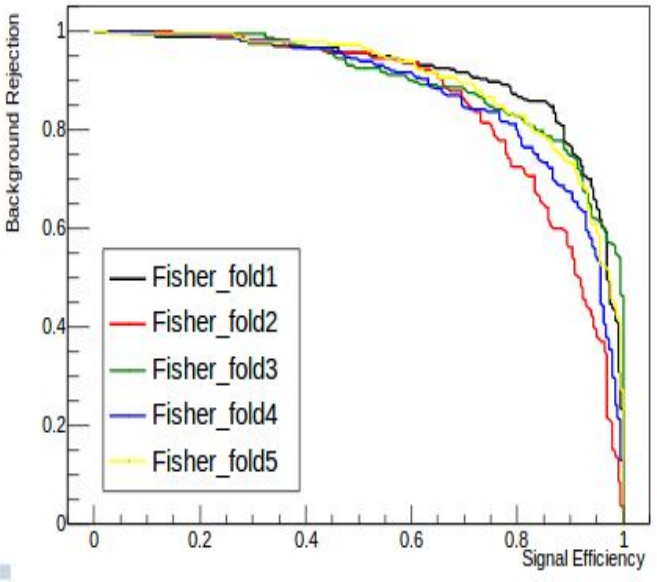
```
<HEADER> CrossValidation         :  ==== Results ====
                         : Fold  0 ROC-Int : 0.9115
                         : Fold  1 ROC-Int : 0.8566
                         : Fold  2 ROC-Int : 0.9001
                         : Fold  3 ROC-Int : 0.8769
                         : Fold  4 ROC-Int : 0.9011
                         : -----------------------
                         : Average ROC-Int : 0.8892
                         : Std-Dev ROC-Int : 0.0222
```

Plot ROC Curves

We enable JavaScript visualisation for the plots

In [9]:
```
%jsroot on
```

In [10]:
```
result.Draw();
```



EXTENDING THE PLOTTING FUNCTIONALITY

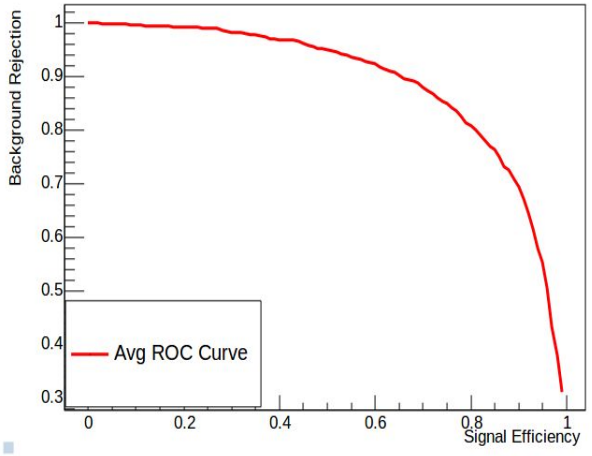
Currently, the users only had the feature to view ROC Curves of the individual folds only. But sometimes we might want to view the average behaviour of our trained model especially when performing Cross Validation.

Hence, a new feature of drawing the average ROC curve is implemented and added to the code base which allows the users to now view the average ROC Curve of the trained model using Cross Validation. This is also added to the new tutorial to provide an example of how to use this.

Following screenshots show how we can use this functionality:

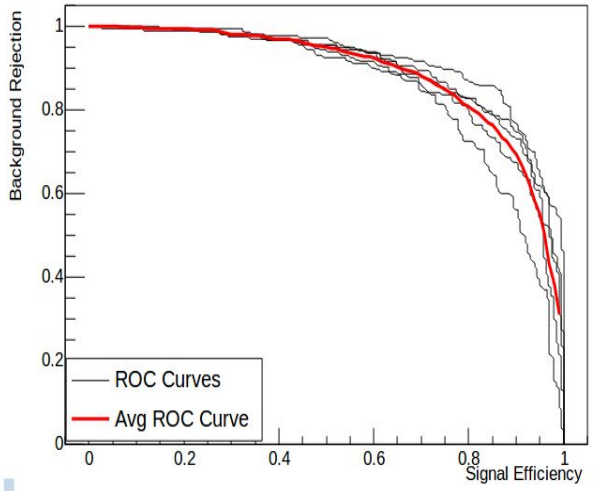
Plot Average ROC Curve

```
In [11]: result.DrawAvgROCCurve("CrossValidation Avg ROC Curve",kFALSE);
```



Plot ROC Curves and the Average ROC Curve

```
In [12]: result.DrawAvgROCCurve("CrossValidation ROC Curves and Avg ROC Curve",kTRUE);
```



IMPROVING CROSS VALIDATION FOLD GENERATION TARGETING UNBALANCED DATASETS - IMPLEMENTING STRATIFIED SPLITTING

As mentioned above, stratified splitting is much better than random splitting.

It allows us to split the dataset randomly into K folds while also maintaining the actual distribution of the whole dataset and making sure that each fold is a true representation of the real dataset.

Hence Stratified splitting is now introduced in TMVA Cross Validation and is available to use for the users. If set KFalse random splitting is used and if set KTrue stratified splitting is used for fold generation.

An example of how to use is available below:

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

d->AddSignalTree(std::get<1>(data_class0));
d->AddBackgroundTree(std::get<1>(data_class1));

d->AddVariable("x", 'D');
d->AddSpectator("id", "id", "");
d->PrepareTrainingAndTestTree(
      "", Form("SplitMode=Block:nTrain_Signal=%i:nTrain_Background=%i:!V", nPointsSig, nPointsBkg));

d->GetDataSetInfo().GetDataSet(); // Force creation of dataset.
```

```
//For Random Splitting
TMVA::CvSplitKFolds split{NUM_FOLDS, "", kFALSE, 0};
   d->MakeKFoldDataSet(split);
```

```
//For Stratified Splitting
TMVA::CvSplitKFolds split1{NUM_FOLDS, "", kTRUE, 0};
   d->MakeKFoldDataSet(split1);
```

FUTURE IMPROVEMENTS

- Weighted Stratified Splitting is an improvement over Stratified Splitting that not only makes sure that the fold generation follows the same distribution as the whole dataset but also make sure that the data is distributed in a way that each fold has the equal weighted data points. This little improvement over the stratified splitting can be added to TMVA Cross Validation to provide much efficient Cross Validation for the users.

- TMVA GUI is an interesting and useful part of the TMVA functionality. IT allows the user to view data visually after training the model. Unfortunately, this feature is only available locally and can be shown to users online using the notebook. Another improvement that can be made is to investigate the feasibility of integrating the TMVA GUI with the tutorial notebooks and make it available online