CERN SUMMER STUDENT
FINAL PRESENTATION

Cross Validation Improvements in TMVA

Mohammad Uzair

Supervisors:

Dr. Lorenzo MONETA

Mr. Hans Kim ALBERTSSON BRANN

About Me:

- **Name** : Mohammad Uzair
- **Country** : Pakistan 
- **Studies** : Final Year of bachelor’s degree in Computer Science
- **University** : National University of Sciences and Technology (NUST)
- **Interested in** : Artificial Intelligence and Machine Learning in Computer Vision

Tasks Assigned:

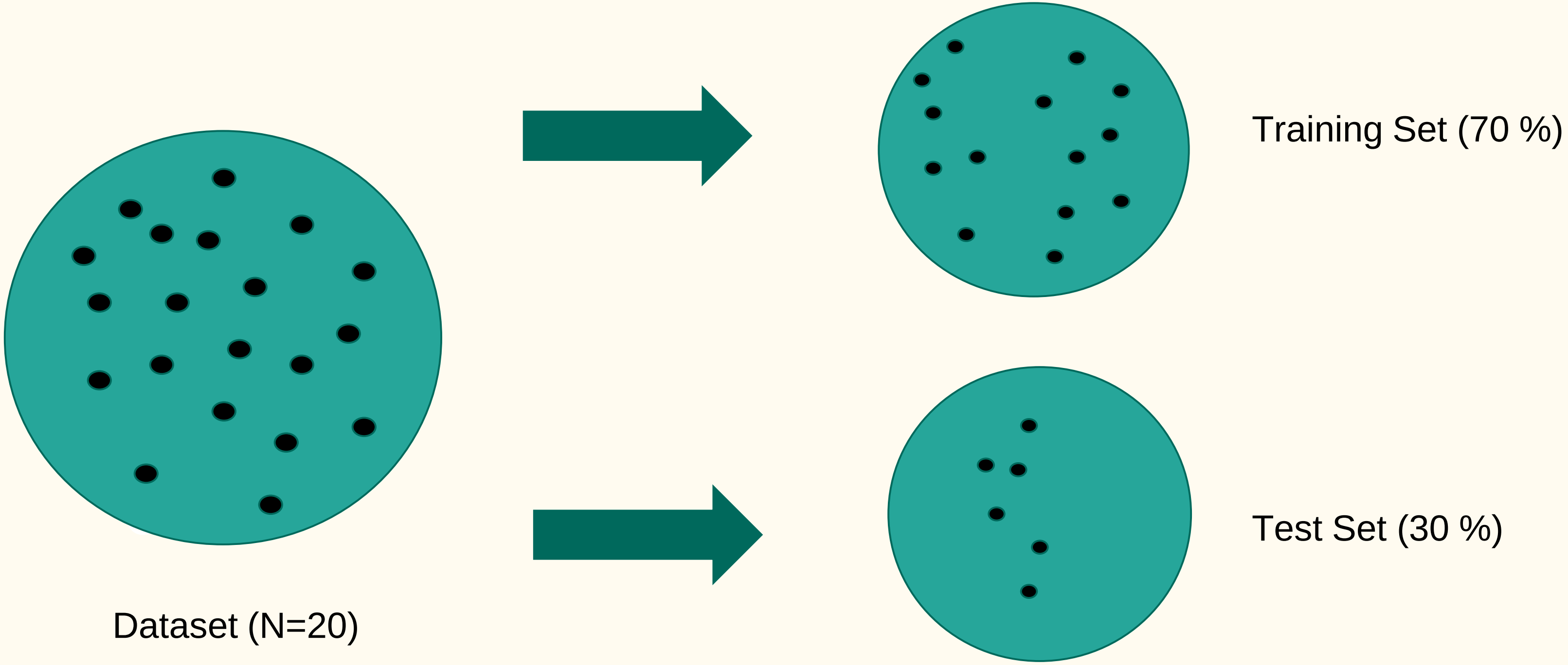
Improve Cross Validation in TMVA

- New tutorial introducing Cross Validation in TMVA and make it available through SWAN.
- Improve presentation of tutorial by extending the plotting functionality (Introducing the feature to draw an average ROC Curve).
- Improve CV fold generation targeting unbalanced datasets (Introducing the feature of Stratified Splitting)

Validation in Machine Learning

- Divide the dataset into Train and Test sets.
- Tries to estimate the expected error of the model.
- Allows us to get an honest assessment of the trained model.

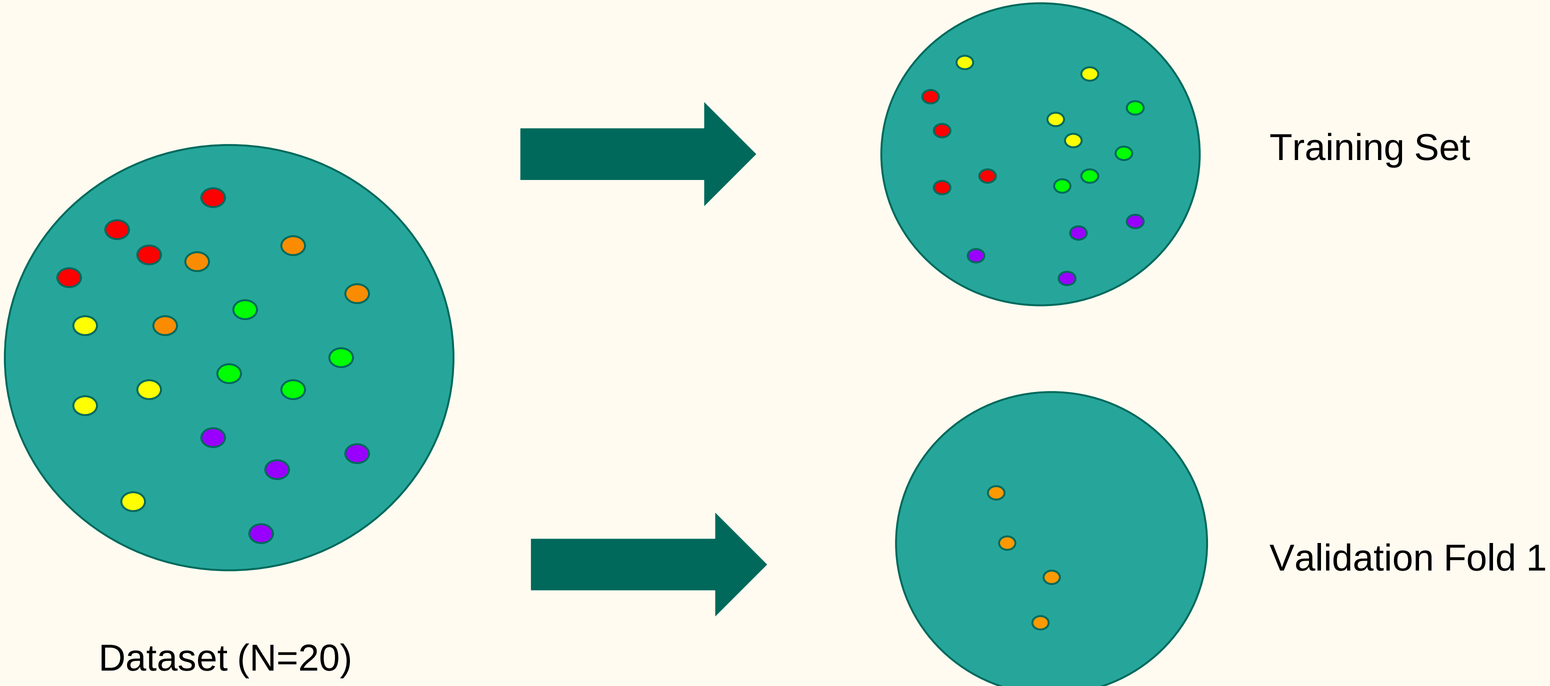
Validation in Machine Learning



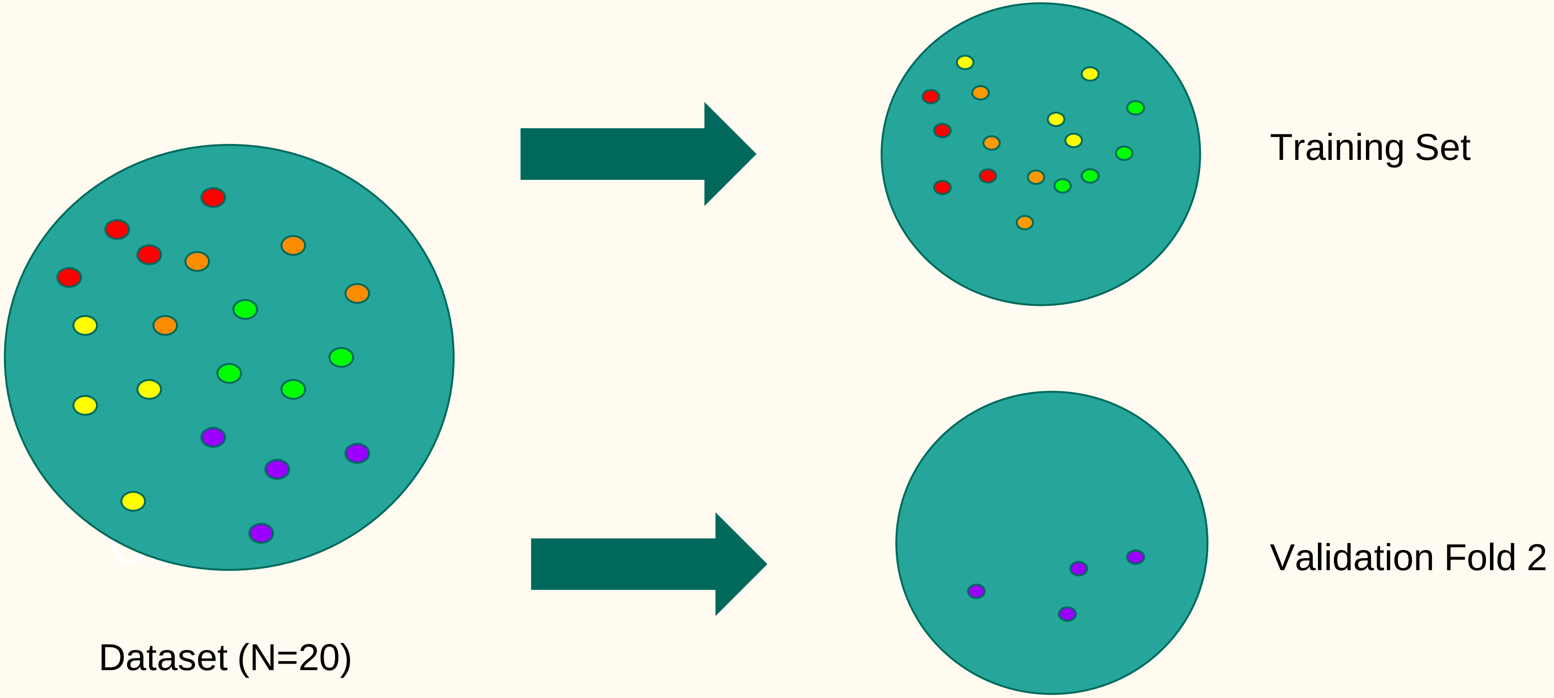
Drawbacks of Validation

- Test errors can be highly variable depending on how much data we use for each set.
- Model Developed on only a subset of data. Ideally we want 100% data for training and 100% for validation.

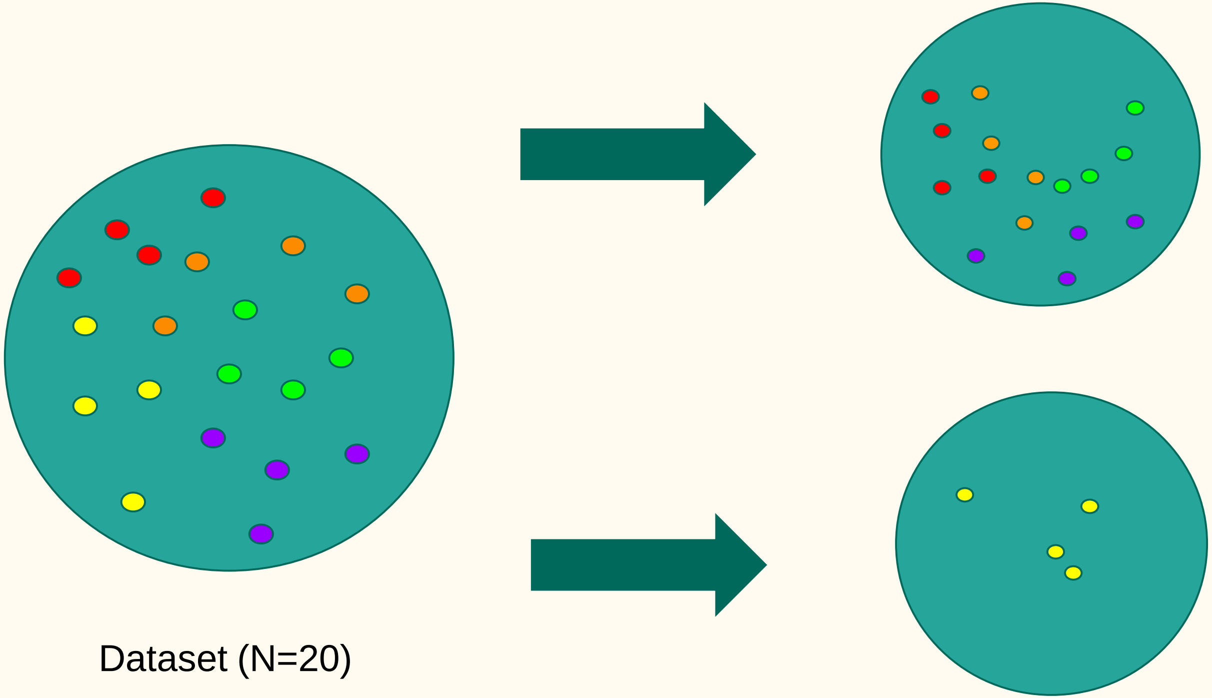
K-Fold Cross Validation



K-Fold Cross Validation



K-Fold Cross Validation



Dataset (N=20)

Training Set

Validation Fold 3

And so on….

Tutorial Introducing TMVA Cross Validation

Tutorial introducing TMVA Cross Validation

- The tutorial was outdated.
- Was not easy to understand for new users.
- A new and updated tutorial in python jupyter notebook with proper explanation so that it is easy to use and understand and make it available through SWAN.

Python Jupyter Notebook for basic Tutorial on TMVA Cross Validation

ROOT CrossValidation (autosaved) Logout Terminal

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

Run Markdown

TMVA

TMVA Cross Validation Example

Define input / Output Files

We declare the files for input and output

In [1]:
```
TMVA::Tools::Instance();

TString inputFileName = "../data/inputdata.root";

auto inputFile = TFile::Open( inputFileName );

auto outputFile = TFile::Open("CV_Output.root", "RECREATE");
```

TMVA Factory

Start by creating the Factory class. We can use the factory to choose the methods whose performance you'd like to investigate.

The factory is the major TMVA object you have to interact with. Here is the list of parameters you need to pass

- The first argument is the base of the name of all the output weightfiles in the directory weight/ that will be created with the method parameters
- The second argument is the output file for the training results
- The third argument is a string option defining some general configuration for the TMVA session. For example all TMVA output can be suppressed by removing the "!" (not) in front of the "Silent" argument in the option string

In [2]:
```
TMVA::Factory factory("TMVAClassification", outputFile,
                      "!V:ROC:!Silent:Color:!DrawProgressBar:AnalysisType=Classification" );
```

DataLoader

The next step is to declare the DataLoader class which provides the interface from TMVA to the input data

Define input variables

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
```

Python Jupyter Notebook for basic Tutorial on TMVA Cross Validation

ROOT CrossValidation (autosaved)

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

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)

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

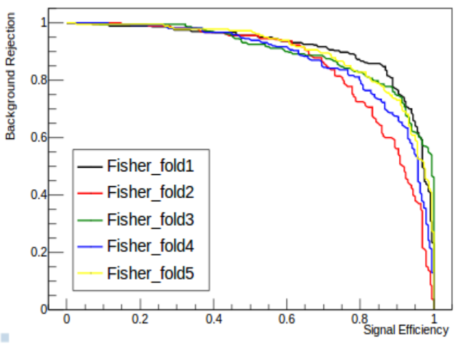
```
<HEADER> CrossValidation          : ==== Results ====
                          : Fold  0 ROC-Int : 0.9115
                          : Fold  1 ROC-Int : 0.8566
                          : Fold  2 ROC-Int : 0.9001
                          : Fold  3 ROC-Int : 0.8769
                          : Fold  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

Improve the presentation of tutorial

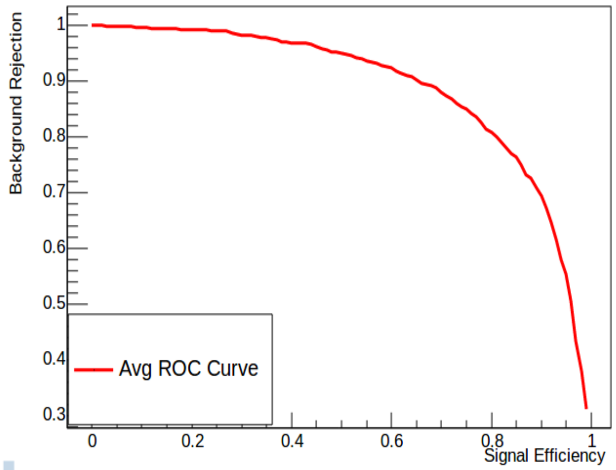
- Currently, we can only show ROC's of individual folds.
- Often, we can be interested in the average behaviour.
- An added feature of visualising the average ROC Curve.
- Addition of this feature in the tutorial to improve the presentation.

Average ROC Curve

With drawFolds = False

Plot Average ROC Curve

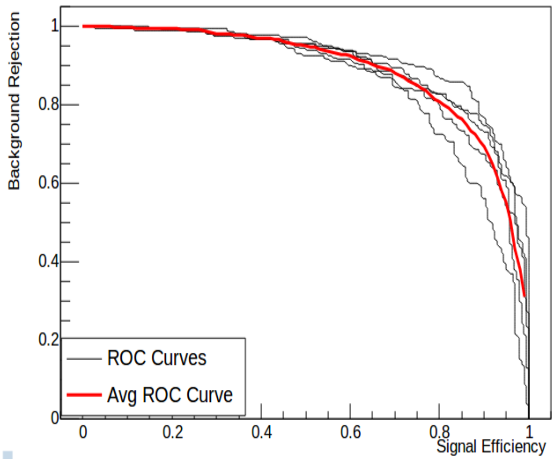
```
In [11]: result.DrawAvgROCCurve("CrossValidation Avg ROC Curve",kFALSE);
```



With drawFolds = True

Plot ROC Curves and the Average ROC Curve

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



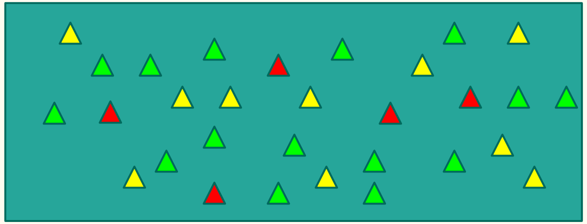
Improve CV fold generation

Random Splitting VS Stratified Splitting

- Determines the distribution of input data.
- Random Splitting just randomly splits the data equally.
- Data of a fold can be distributed differently than the whole.
- Problems, in particular, arise with unbalanced classes (can more easily occur in multi-class classification).
- Stratified Splitting ensures that each fold follows the same distribution as the whole.

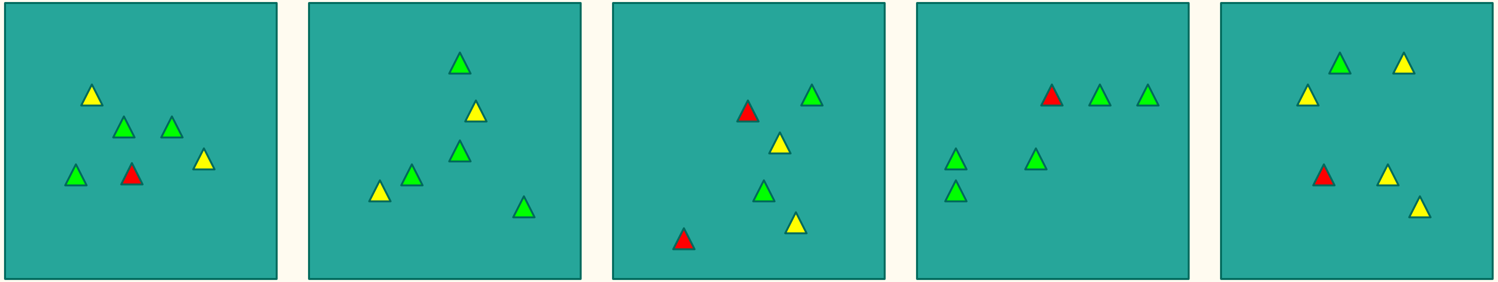
Random Splitting VS Stratified Splitting

- Random Splitting just randomly splits the data equally



30 samples
03 Classes
10 samples
15 samples
05 samples

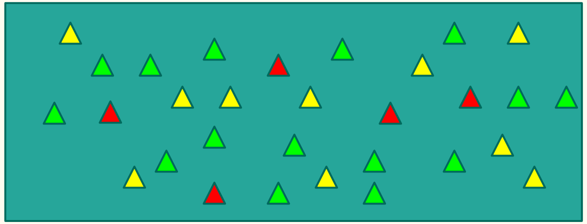
6 random samples in each fold



Fold 1 Fold 2 Fold 3 Fold 4 Fold 5

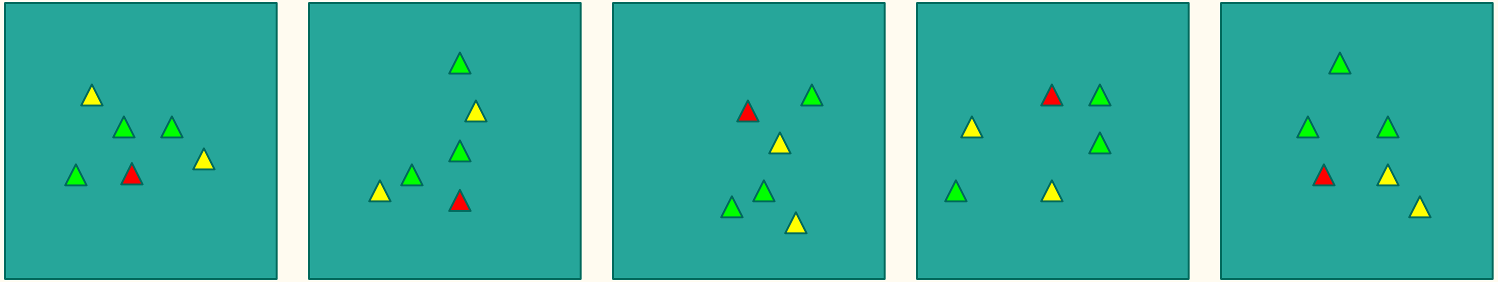
Random Splitting VS Stratified Splitting

- Stratified Splitting randomly splits data ensuring that each fold is good representative of the whole.



30 samples
03 Classes
10 samples
15 samples
05 samples

6 random samples in each fold



Fold 1 Fold 2 Fold 3 Fold 4 Fold 5

Stratified Splitting in TMVA

```
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 kFolds Splitting
- Investigate feasibility of integrating TMVA GUI with tutorials
- Investigate feasibility of adding validation set (this would mainly be beneficial for methods with a large number of parameters, e.g. DNN and BDT's)

Special Thanks to my Supervisors:

Dr. Lorenzo MONETA

Mr. Hans Kim ALBERTSSON BRANN