MUONMatcher TMVA interface

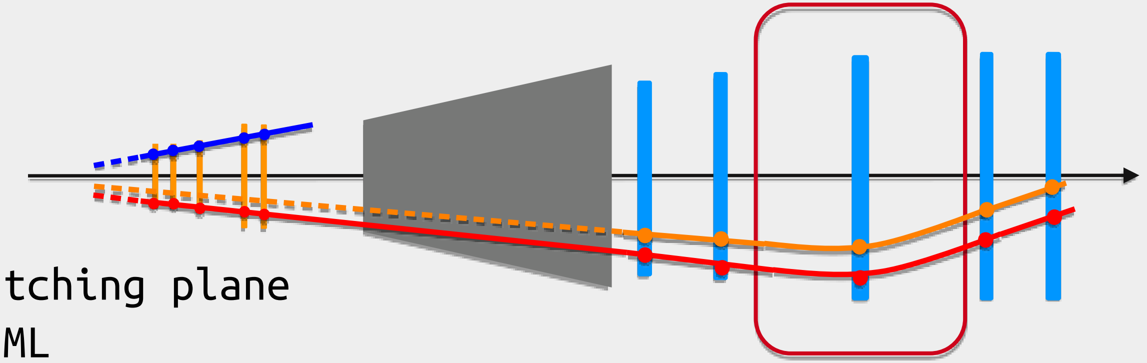
ALICE Machine Learning Meeting
Jun 15, 2021
Lucas Nunes Lopes

Outline

- → Muon Matching Overview
- → Single Definition for Features
  - Data export
  - Training ML
  - “Evaluate”
- → Flexible Interface for ROOT’s TMVA

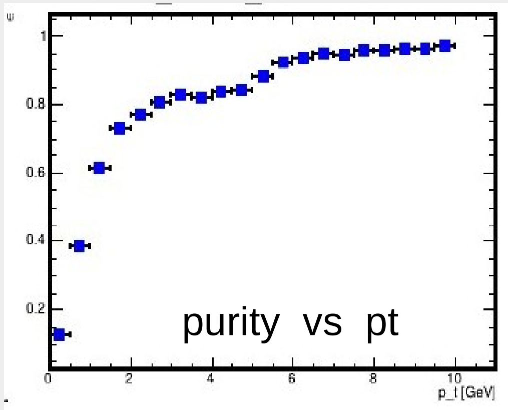
Matching Tool Overview

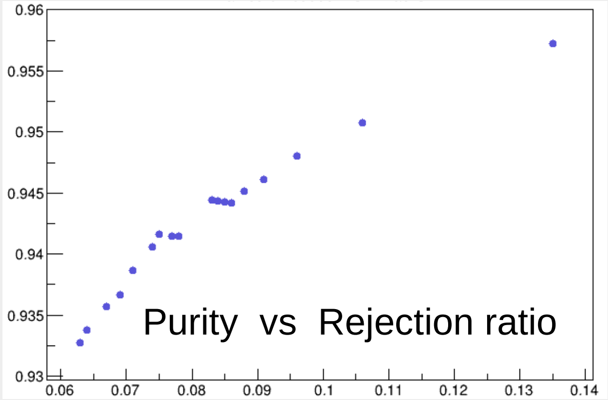
➔ **Track Matching Method**



- 1. Generate events
- 2. Reconstruct MCH and MFT tracks
- 3. Extrapolate MCH/MFT Tracks to matching plane
- 4. Find best match using chi2 - or ML
- 5. Check if it’s a correct match or a fake match

➔ **Matching Tool applies and assesses the matching method and ML alternatives**





Workflow

- **1. Generate events: simulate MCH and MFT tracks**
  - **1.1 Training data**
    - `matcher.sh --genMCH --genMFT -n 100 --nmuons 10 --npions 2 -o training_data_dir`

- **2. Generate training data file**

  `matcher.sh --exportTrainingData 1000 -o training_data_dir`

- **3. Train neural network**

  `matcher.sh --train DNN --layout DL3.0 --strategy ts1 --MLoptions oo1 --trainingdata training_data_dir/MLTraining_1000_MCHTracks.root -o training_data_dir`

- **4. Run Track matching using a trained network + check results**
  - `matcher.sh --match --checks --matchFcn trainedML --weightfile training_data_dir/trainedMLs/weights/Regression_DNN_DL3.0_ts1_oo1__MLTraining_1000_MCHTracks.weights.xml -o matching_dir`

General Issue: Extracting input data

→ **From a set of objects, extract ML input data (features)**

Reconstructed objects

(User's extraction function; defined once, used where needed)

ML Features Set (float vector) + "Truth"

→ **For this: <u>setMLFeatures Function</u>**

→ For instance, for the MCH-MFT matching:

MCH Track | MFT Track

ML Features Set + "Truth"

Data Export

→ **Step #2: Data Export**

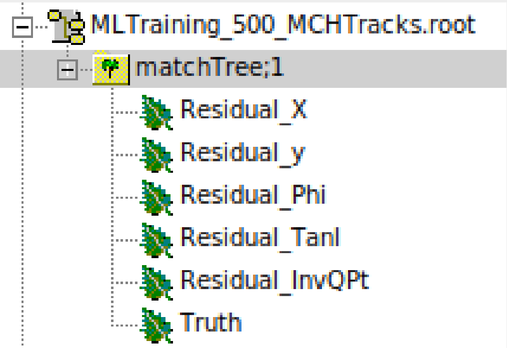
- Uses the setMLfeatures function.

MCH Track	MFT Track

⇩

Res_x	Res_y	Res_Phi	Res_Tanl	Res_InvQPt

- Exports in root format;



ROOT’s TMVA

- **Toolkit for Multivariate Data Analysis with ROOT.**
  - As is a ROOT-integrated environment, it’s already integrated in O2;

- **TMVA includes several machine learning methods.**
  - supervised learning;

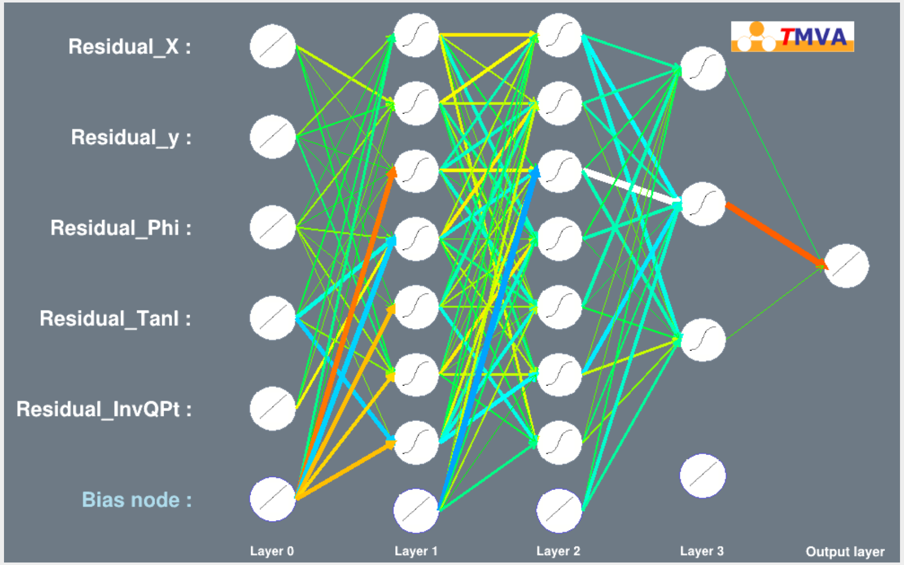
- **Saves information from training, testing and evaluation in Root file**
  - Can be displayed via TMVA’s GUI;

- **Trained ML saved in a “weight” file (XML format)**
  - also a standalone C++ class (only for some classification methods).

- **More at TMVA User’s Guide**

ROOT's TMVA: DNN Method

- ➔ **The interface allows use of every TMVA method;**
- ➔ **TMVA's Deep Neural Networks are part of the Deep Learning module.**
  - Artificial neural network, much like MLP (Multi Layer Perceptron)
  - deep learning module also supports Convolutional Networks (CNN) and Recurrent netoworks (RNN)



TMVA

➔ Training Inicialization

- TMVA::Factory *factory = new TMVA::Factory(
  "Regression_DNN", methodname.root,"!V:!Silent:AnalysisType=Regression");
  - methodname = “DL3.0_ts2_oo1_MLTraining_1000_MCHTracks”

- TMVA::DataLoader* dataloader = new TMVA::DataLoader("trainedML");
  - Handles input data

- dataloader->AddVariable(mMLInputFeaturesName[i], mMLInputFeaturesName[i], "units", 'F');

- dataloader->AddTarget("Truth");

TMVA

➔ **Step #3: Training**

- methodname = “DL3.0_ts2_oo1_MLTraining_1000_MCHTracks”

- trainingstr = “Layout=RELU|8,RELU|8,RELU|4,LINEAR:
  TrainingStrategy=LearningRate=1e-3, ConvergenceSteps=300,BatchSize=50,
  TestRepetitions=10,Regularization=L2,MaxEpochs=2000,Repetitions=1:
  H:V:ErrorStrategy=SUMOFSQUARES:VarTransform=G:RandomSeed=42:
  WeightInitialization=XAVIERUNIFORM:Architecture=CPU:ValidationSize=0.2”

- dataloader->AddRegressionTree(regTree, regWeight, Types::kTraining);

- factory->BookMethod(dataloader,
  TMVA::Types::Instance().GetMethodType(DNN),
  methodname, trainingstr);

- factory->TrainAllMethods();

TMVA

➔ Step #4: Matching

- Inicialization:
  - TMVA::Reader *reader = new TMVA::Reader("!Color:!Silent");
  - Add variables (similar to the training step);

- mTMVAReader->BookMVA("MUONMatcherML", mTMVAWeightFileName);
  - Weightfile: Regression_DNN_DL3.0_ts2_oo1__MLTraining_1000_MCHTracks.weights.xml

- <u>Call setMLFeatures function:</u>

MCH Track	MFT Track

Res_x	Res_y	Res_Phi	Res_Tanl	Res_InvQPt

- mTMVAReader->EvaluateRegression(0, "MUONMatcherML");

Quick examples

➔ Each method has their own configuration block, that can be formed up to three differents group of settings: “layouts”, strategies, MLoptions.

- **Examples:**

  - `matcher.sh --train BDT --MLoptions tmva_tuto --trainingdata <training_file.root> -o sample_dir`

  - `matcher.sh --train MLP --layouts ml4.0 --strategies mlp_ts --MLoptions ex1 --trainingdata <training_file.root> -o sample_dir`

  - `matcher.sh --train DNN --layouts DL4.1 --strategies ts1 --MLoptions oo1 --trainingdata <training_file.root> -o sample_dir`

➔ **Options for the available methods can be found at TMVA User’s Guide**

Summary

→ **It was shown the use of TMVA within the MFT Track Matching Tool**

- TMVA available in O2 via ROOT

→ **SetMLfeatures function (mostly indepent of ROOT)**

- Can be generalized for any data
- Exportation can be easily set to other formats

→ **Flexible TMVA ML interface**

- Tested and validated for Regression methods
- Classifications methods are being considered (essentially read; see backup slides)

Tutorials:

➔ MUONMatcher TMVA interface is inspired on TMVA tutorials:

- Training:
  - Regression
  - Classification

- Application (evaluation):
  - Regresson
  - Classification

Backup slides

ML input Features configuration:

➔ **Setting the Feature Function**

- We provide the arguments (besides the tracks):
  - Function defining the features;
  - Number of features;
  - Function alias;
  - Function defining features names (optional).

➔ **Features defined by separated functions (built in)**

- Useful for sets with promissing results;
- Definition such as the already existing functions:
  - features;
  - names;

➔ **lambda functions at the steering macro:**

```
matcher.setMLFeatureFunction([](const MCHTrackConv& mchTrack,
                                const MFTTrack& mftTrack,
                                float* features) {
    features[0] = mftTrack.getX() - mchTrack.getX();
    features[1] = mftTrack.getY() - mchTrack.getY();
    features[2] = mftTrack.getPhi() - mchTrack.getPhi();
    features[3] = mftTrack.getTanl() - mchTrack.getTanl();
    features[4] = mftTrack.getInvQPt() - mchTrack.getInvQPt();
  },
    5, "ML5ParDeltas",
    [](string* featuresNames) {
    featuresNames[0] = "Residual_X";
    featuresNames[1] = "Residual_Y";
    featuresNames[2] = "Residual_Phi";
    featuresNames[3] = "Residual_Tanl";
    featuresNames[4] = "Residual_InvQPt";
  }
);
```

- Useful to test new sets of features

Configuration File

- Each method has their own configuration block, that can be formed up to three differents group of settings: “layouts”, strategies, MLoptions.

```
<BDT>
  <Options>
    <tmva_tuto>!H:!V:NTrees=100:MinNodeSize=1.0%:BoostType=AdaBoostR2:SeparationType=RegressionVariance:nCuts=20:PruneMethod=CostComplexity:PruneS
trength=30</tmva_tuto>
  </Options>
</BDT>

<MLP>
  <layouts>
     <ml4.2>HiddenLayers=30,20,15,5</ml4.2>
  </layouts>
  <Training_Strategies>
     <mlp_ts>NCycles=500:TestRate=10:TrainingMethod=BFGS:Sampling=0.3:SamplingEpoch=0.8:ConvergenceImprove=1e-4:ConvergenceTests=10:BatchSize=50:L
earningRate=1e-4</mlp_ts>
  </Training_Strategies>
  <options>
     <ex1>!H:!V:VarTransform=Norm:UseRegulator:RandomSeed=0</ex1>
  </options>
</MLP>

<DNN>
  <layouts>
    <DL4.1>Layout=RELU|35,RELU|20,RELU|15,RELU|5,LINEAR</DL4.1>
  </layouts>
  <Training_Strategies>
    <ts1>TrainingStrategy=LearningRate=1e-3,ConvergenceSteps=50,BatchSize=50,TestRepetitions=10,Regularization=L2,MaxEpochs=2000,Repetitions=1</ts
1>
  </Training_Strategies>
  <Other_Options>
    <oo1>H:V:ErrorStrategy=SUMOFSQUARES:VarTransform=G:WeightInitialization=XAVIERUNIFORM:Architecture=CPU:ValidationSize=0.2:RandomSeed=42</oo1>
  </Other_Options>
</DNN>
```

TMVA

➔ **Step #3: Training (comments)**

- When using classification:
  - signal and background data added separately;
  - No “addTarget” function (dataloader);

- Input data can be in root format (TTree) or text file;

- If we want factory to make some analysis on training:
  - factory->TestAllMethods();
  - factory->EvaluateAllMethods();

Open Issues / WIP

- ➔ TMVA interface teste and validate for Regression methods

- ➔ Interface with Classification methods is essentially ready
  - Missing ingredient: training data format
  - Different Training data format as compared to regression
    - Classification: Signal and background data stored in different trees
    - Desired feature: use same training data file for both Regression and Classification methods
      - technically feasible, but splitting correct and fake matches in different trees may affect Regression training performace due to poor randomization.