JOINT INSTITUTE
FOR NUCLEAR RESEARCH

Custom NN for data classification using evolutionary training

Identificantion of ***μ*** and ***π*** tracks based on reconstructed parameters using NN

I.Yeletskikh

Intro

- Problems in common NN APIs
  - ROC auc or signal significance as fitness function
  - Accounting for uncertainties of the input variables
  - Switching on/off some of the input neurons
  - Control of overtraining directly for the observable parameters
  - Optimization of NN hyperparameters
  - Memory leaks in some python-based APIs

- Evolutionary algorithms for NN training
  - No need for differentiability of the fitness function
  - Simultaneous optimization of parameters and hyperparameters
  - Custom criteria of the overtraining
  - C++ based API with cpu optimization snd transparent memory management

- Application to the muon-pion identification

Problems in common NN APIs: differentiability of the loss

- The majority of training algorithms require (numerical) differentiability of the fitness (loss) function. Despite all of them combine stochastic and deterministic approaches, most are based on gradients or similar predictions of the fitness function, for which differentiability is needed.

- Finding steepest descend even for differentiabel multidimansional function can be a complex problem.

- Physically meaningful observables (e.g., signal over background significance or efficiency at a certain working point) are not (or poorly) differentiable. It means they can be poorly optimized by deterministic algorithms.

- AUC-ROC or ‘signal significance’ are simply not implemented as optional loss functions in existing NN APIs.

http://www.icml-2011.org/papers/198_icmlpaper.pdf

Problems in common NN APIs: uncertainties of inputs and switching off neurons

- Input variables are usually have experimental uncertainties, that have to be accounted for.

- Some physical input variables are not defined in some input events OR have different dimensions in different events. Accounting for such events is not straightforward without introducing bias to classification.

- It may be wasteful to ignore events where not all inputs defined.

Problems in common NN APIs: overtraining

- Overtraining consists in NN being trained to data fluctuations instead of real kinematics, giving better performance than that possible from signal/BG kinematic differences.

- In case real data would fluctuate in the opposite direction compared to the training MC sample, the classification performace will

- It’s desirable to control overtraining via the same fitness function for which training is performed plus (optionally) additional kinematic variables or NN output values. This is not directly implemented in most of existing NN API.

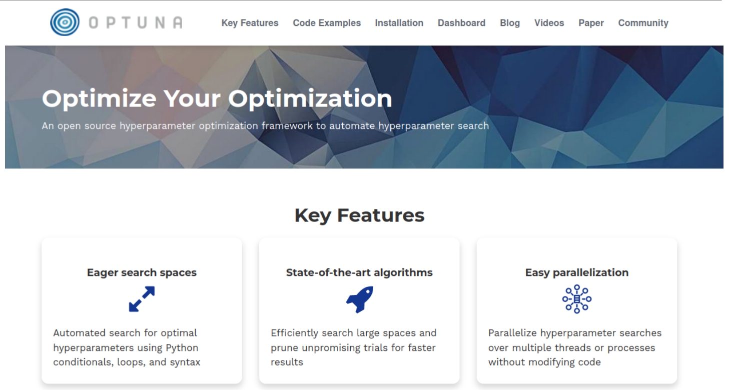
Problems in common NN APIs: hyperparameters optimization

- In addition to explicit parameters (synapse weights, neuron shifts) NN includes a lot of implicit parameters (number of layers, neurons in layers, activation functions, specific set of input variables, options for training algorithm, etc.) that are referred to as hyperparameters.

- Particular choice of hyperparameters can substantially affect the performance of the NN. Intuitive choices are often far from optimal.

- There are applications that allow optimization of hyperparameters (e.g., optuna). In practice they show poor performance, since hyperparameter space is essentially irregular.

- Moreover, memory leaks is a common problem for the hyperperameter optimization applications being applied to python based NN APIs.



OPTUNA   Key Features   Code Examples   Installation   Dashboard   Blog   Videos   Paper   Community

Optimize Your Optimization

An open source hyperparameter optimization framework to automate hyperparameter search

Key Features

Eager search spaces

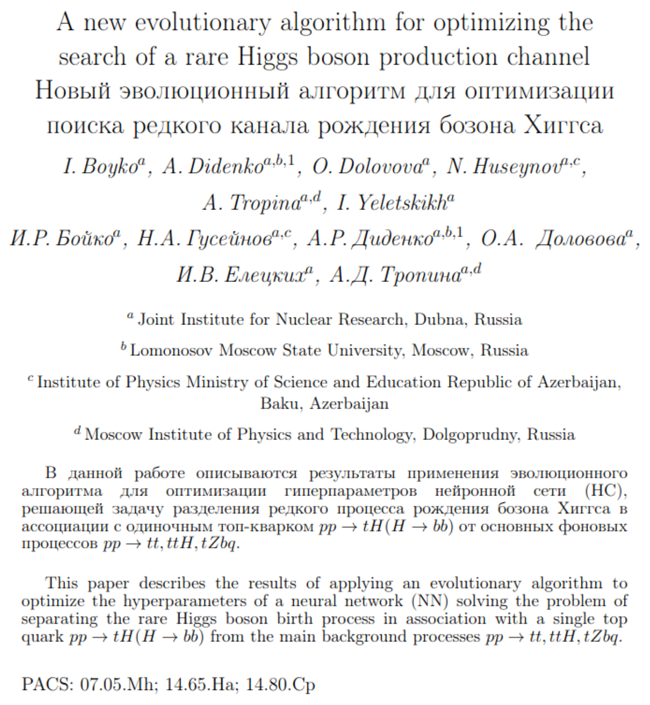
Automated search for optimal hyperparameters using Python conditionals, loops, and syntax

State-of-the-art algorithms

Efficiently search large spaces and prune unpromising trials for faster results

Easy parallelization

Parallelize hyperparameter searches over multiple threads or processes without modifying code



A new evolutionary algorithm for optimizing the search of a rare Higgs boson production channel

Новый эволюционный алгоритм для оптимизации поиска редкого канала рождения бозона Хиггса

I. Boyko[a], *A. Didenko*[a,b,1], *O. Dolovova*[a], *N. Huseynov*[a,c], *A. Tropina*[a,d], *I. Yeletskikh*[a]

И.Р. Бойко[a], *Н.А. Гусейнов*[a,c], *А.Р. Диденко*[a,b,1], *О.А. Доловова*[a], *И.В. Елецких*[a], *А.Д. Тропина*[a,d]

[a] Joint Institute for Nuclear Research, Dubna, Russia

[b] Lomonosov Moscow State University, Moscow, Russia

[c] Institute of Physics Ministry of Science and Education Republic of Azerbaijan, Baku, Azerbaijan

[d] Moscow Institute of Physics and Technology, Dolgoprudny, Russia

В данной работе описываются результаты применения эволюционного алгоритма для оптимизации гиперпараметров нейронной сети (НС), решающей задачу разделения редкого процесса рождения бозона Хиггса в ассоциации с одиночным топ-кварком $pp \to tH(H \to bb)$ от основных фоновых процессов $pp \to tt, ttH, tZbq$.

This paper describes the results of applying an evolutionary algorithm to optimize the hyperparameters of a neural network (NN) solving the problem of separating the rare Higgs boson birth process in association with a single top quark $pp \to tH(H \to bb)$ from the main background processes $pp \to tt, ttH, tZbq$.

PACS: 07.05.Mh; 14.65.Ha; 14.80.Cp

Evolutionary algorithm for NN training

- Most of this problems can be (partially) solved. However, a lot of custom code required.
- Some of the problems (like feeding non-differentialble loss to gradient training algorithm, optimization of hyperparameters and memory leaks) are problematic to address.

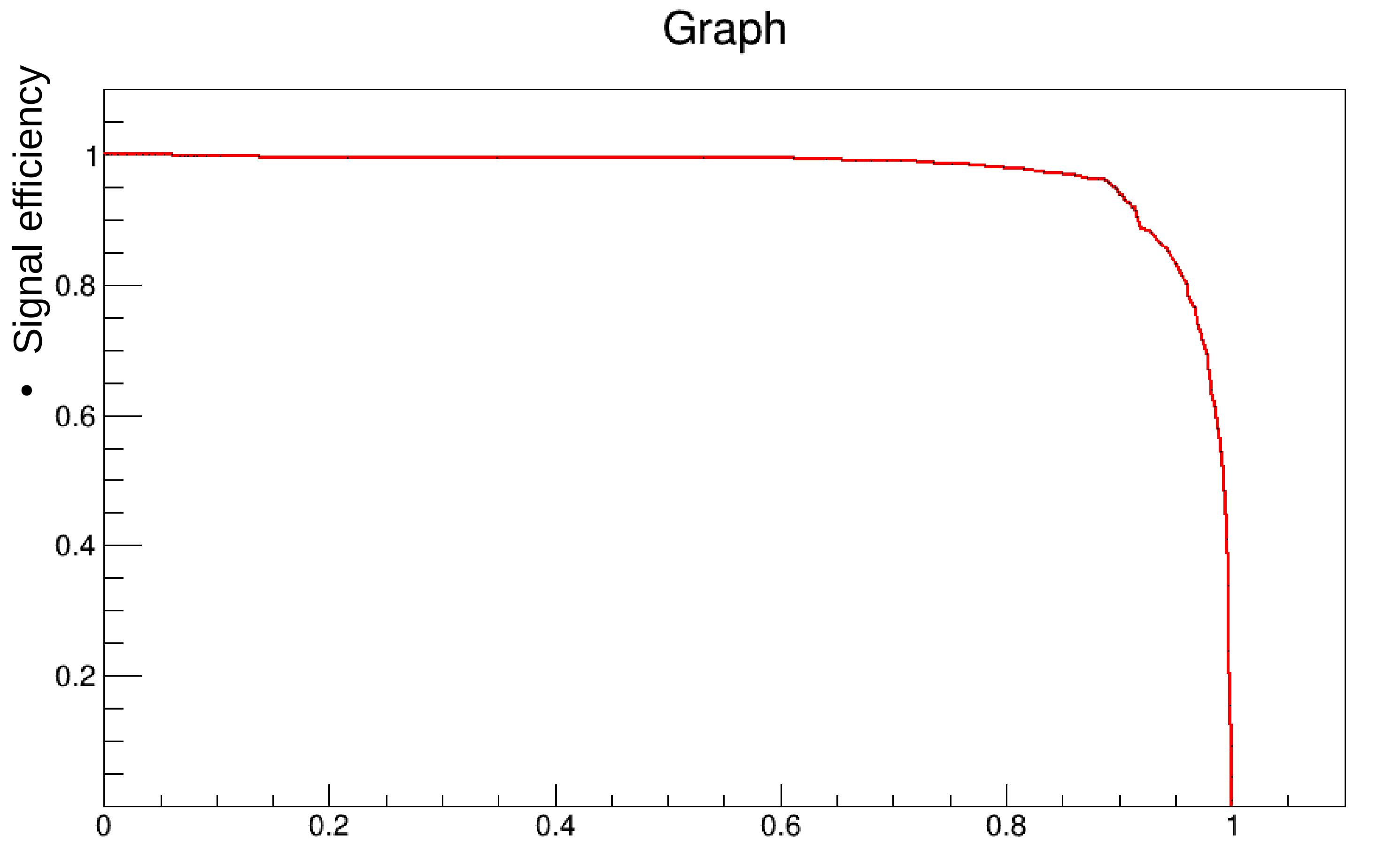
- One of the possible solutions is using custom NN API (c++ based in my case) that uses evolutionary algorithm for training.

- Non-differentiable functions are allowed, since no gradients are computed.
- Uncertainties of the input values can be included and reflected to the uncertainty of the NN input, thus automatically accounting for different ‘importance’ of different input events.
- Input neurons can be ‘switched off’ for those events where some of inputs are not defined.
- Overtraining is controlled by comparing ROC-AUC, significance or NN output distributions between training and testing samples.
- Hyperparameters can be optimized alongside explicit parameters, c++ code allows simple and transparent memory management.

Evolutionary algorithm for NN training

- This custom NN API is applied to the pion-muon identification task.
- Just 3 input variables are used for the test purpose (track length, track length in RS, number of hits in RS)
- NN implementation is simple, involving classes for neuron layers and synapse connection layers.
- Deep NN with 2 hidden layers (15, 9 neurons) is constructed, containing 189 synapse connections (~380 explicit parameters)
- Population of 50 neural networks is created
- At each training step (generation) the one or few best performing NNs give rise to their children with random mutations of the parameters applied
- Overtraining is controlled by the difference between ROC-AUC for training sample and testing sample.

Track candidates: NN application. Muons and pions with 1.5GeV > pT > 2.1GeV

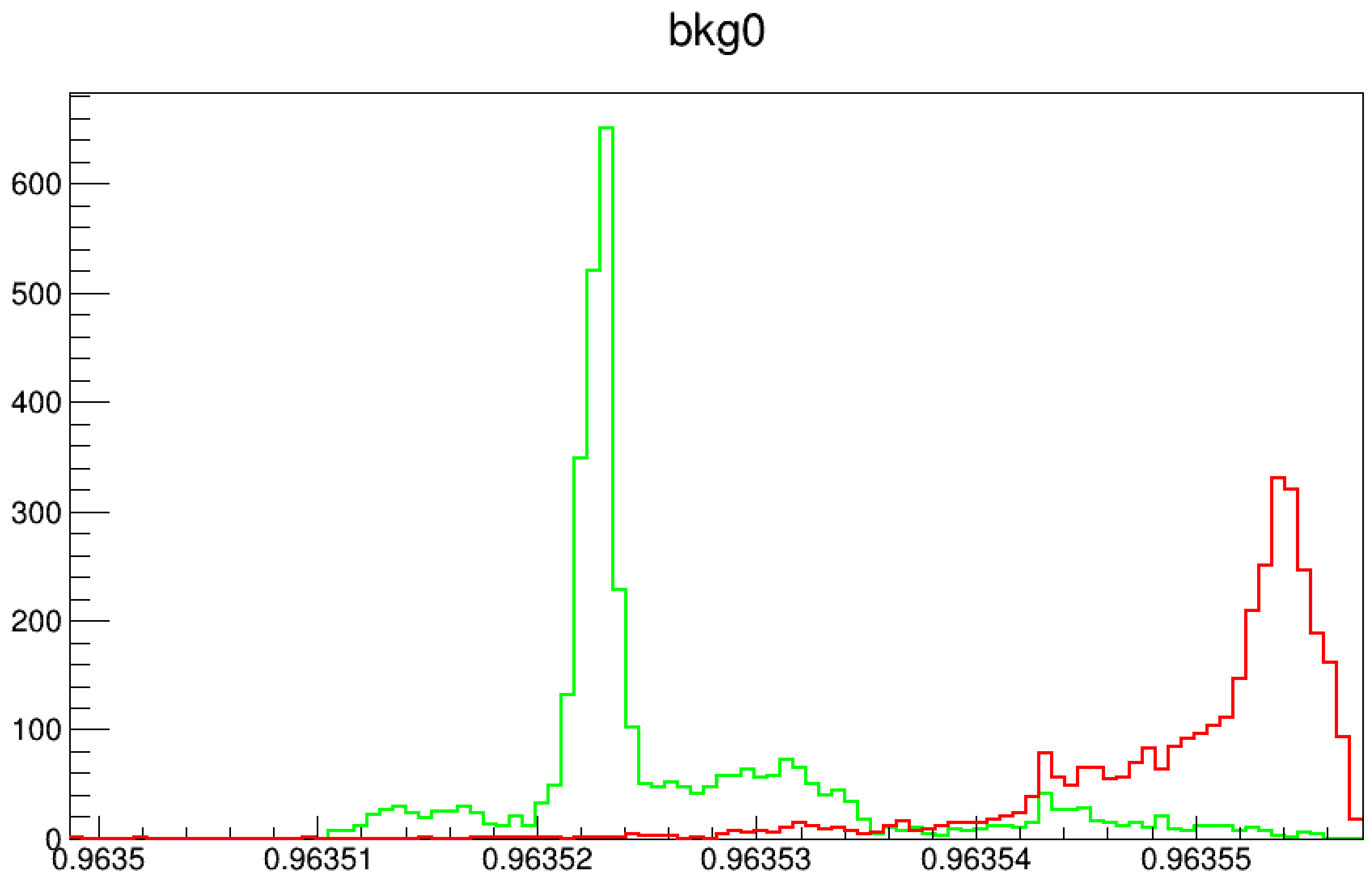


- Background suppression

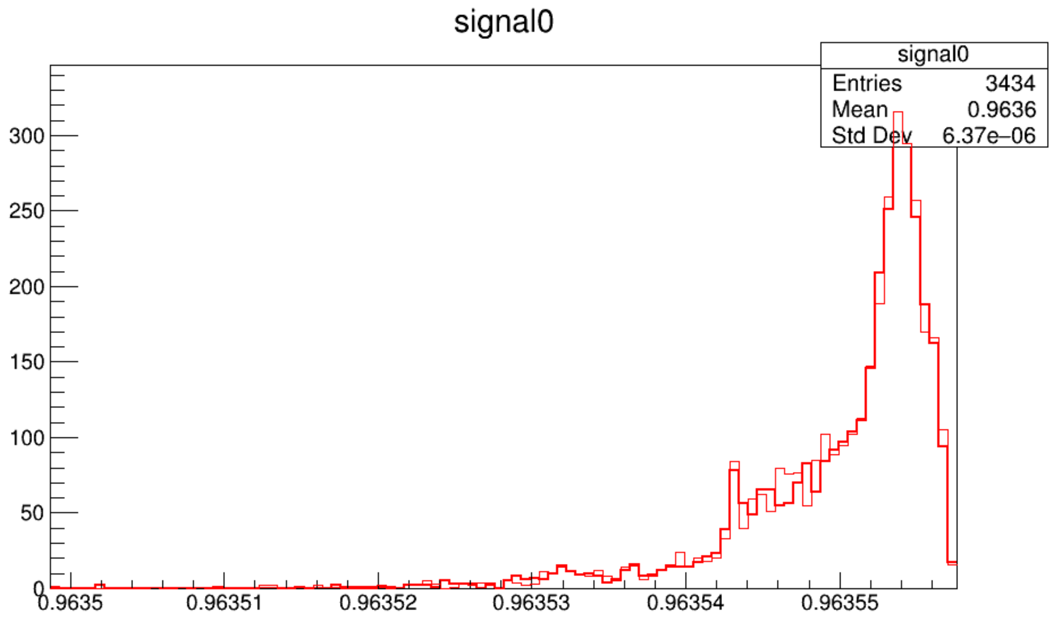
Possible WP:

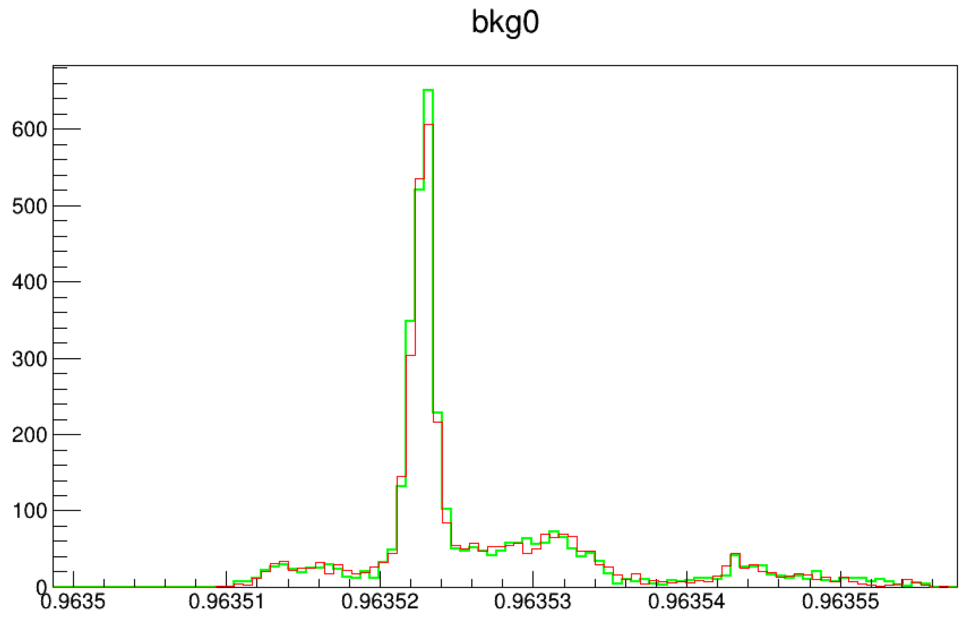
- Signal eff 9.902e-01 BG rejection 5.451e-01
- Signal eff 9.813e-01 BG rejection 6.407e-01
- Signal eff 6.938e-01 BG rejection 9.901e-01
- Signal eff 7.972e-01 BG rejection 9.802e-01

Track candidates: NN response to signal (red) and background (green)



Track candidates: NN response to signal (red) and background (green)





- Overtraining is controlled by comparing ROC-AUC for testing and training samples
- The observed difference is <0.4 permille
- Comparison of NN response to training and testing samples for signal and background are shown on the plots. No systematic deviations are seen.

conclusions and plans

- Custom NN trained by evolutionary algorithm showed better performance w.r.t. signal/background efficiency as compared to keras-based NN.
- It is possible to implement ROC-AUC value as loss without problems in training.

- Training takes ~2-5 min on a typical CPU. For complex networks it’s going to be slower...

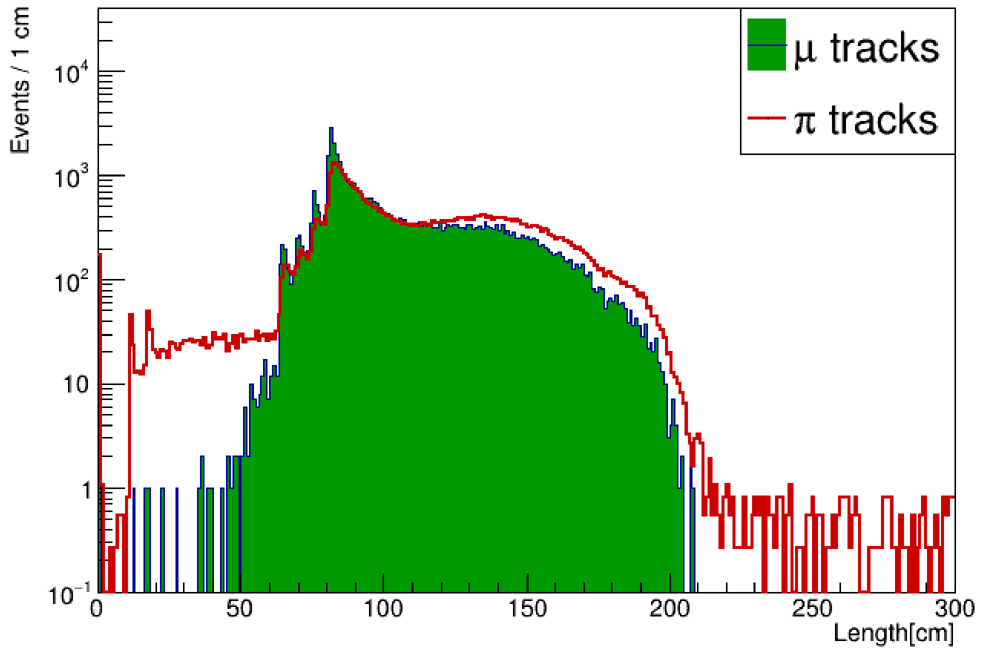
- Instead of ROC-AUC optimization one may try optimizing a specific working point performance, e.g., maximize signal efficiency at the point with background rejection of 99%.

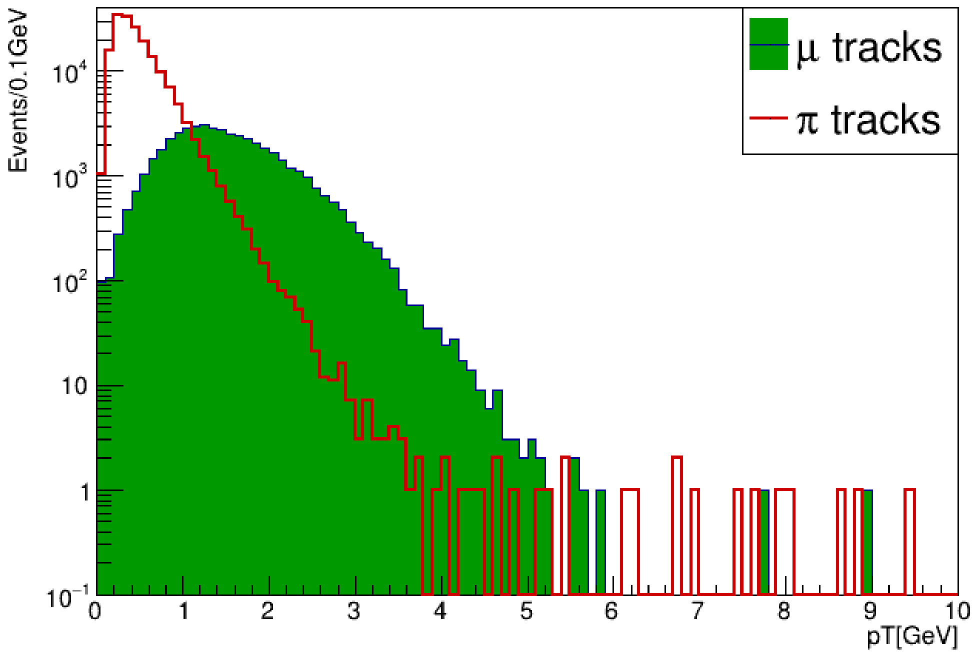
- Add more variables (like pt)

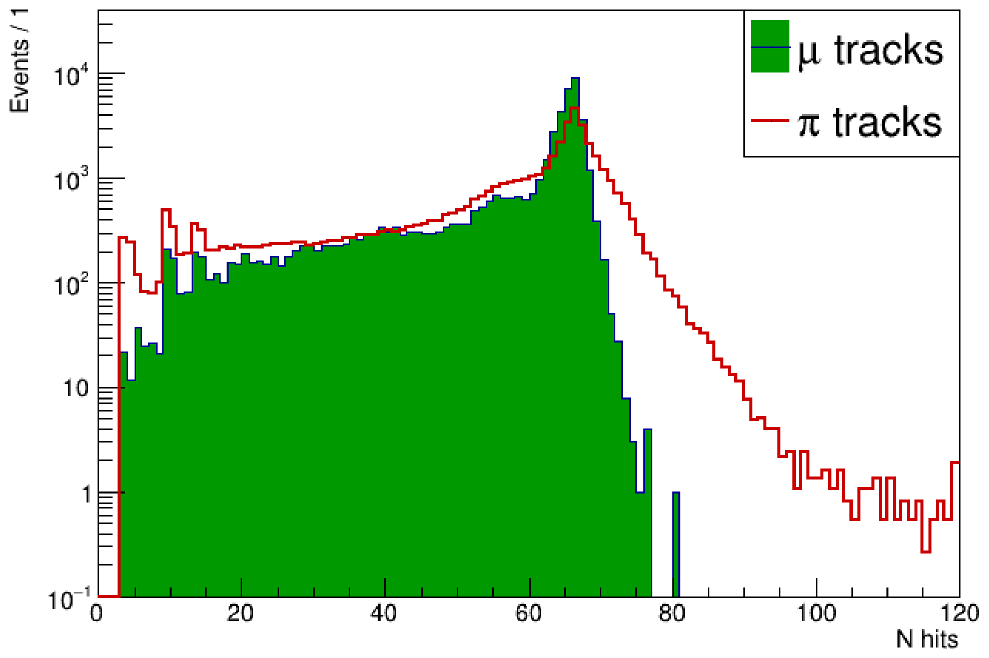
- Using this NN output plus requirement of two opposite charge muons in the event, plus information about common vertex, charmonia decaying to muon pairs selection algorithm can be implemented (in progress).

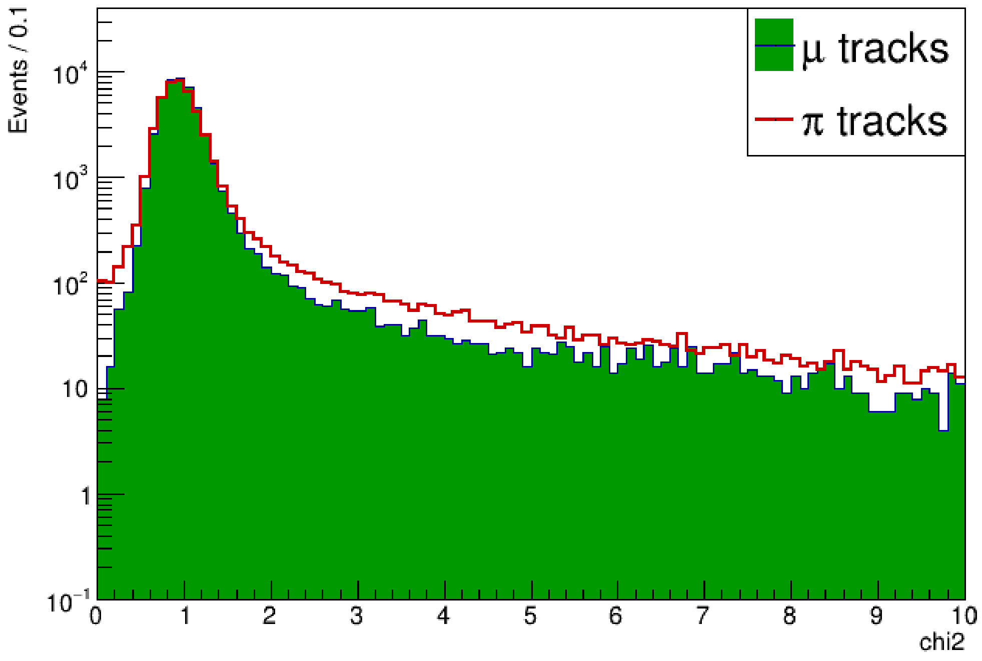
back-up

Track candidates: input variables

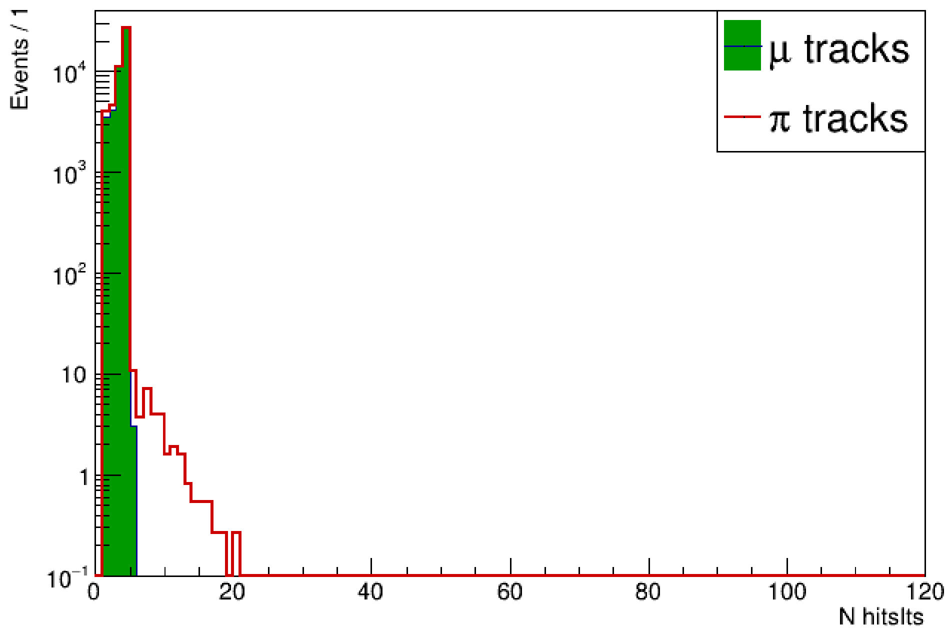


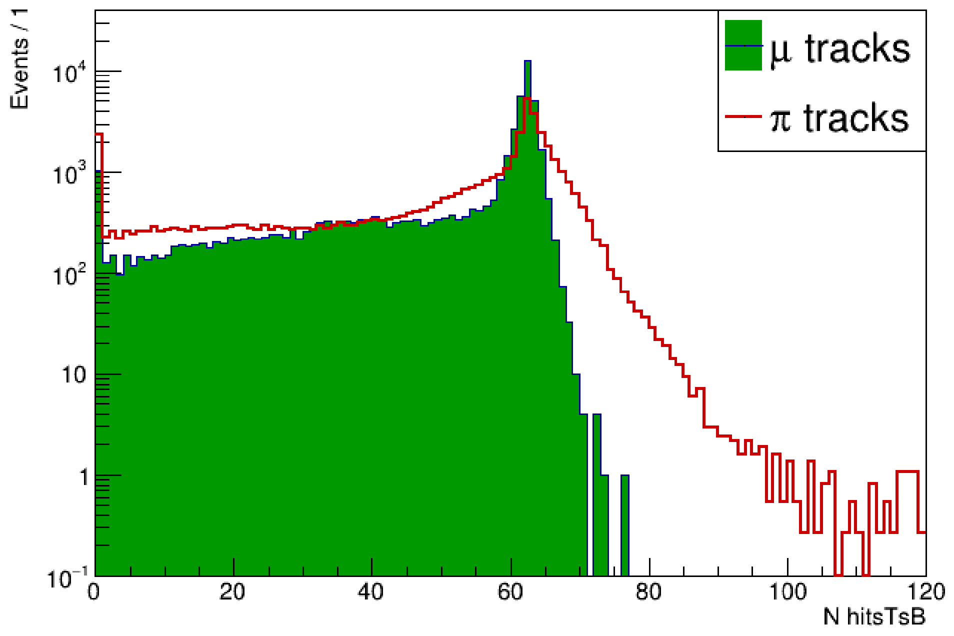


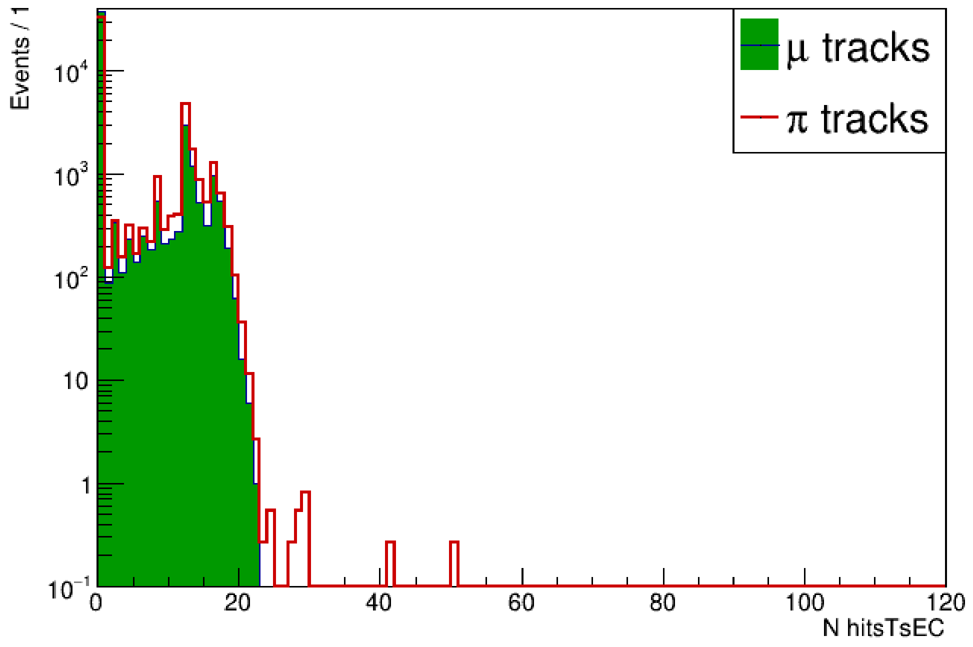




Track candidates: input variables


Events / 1
μ tracks
π tracks
N hitsIts


Events / 1
μ tracks
π tracks
N hitsTsB


Events / 1
μ tracks
π tracks
N hitsTsEC