Use of machine learning techniques to discriminate single particle clusters from various background physical and instrumental contributions.

Sacha Rejai, Stanislas Lambert

February 27^{th}, 2024

Faculté de **physique** et **ingénierie**
Université de Strasbourg

master PSA
Physique Subatomique
Astroparticules

Summary

- **Introduction**
  - Context
  - Program architecture
- **Data simulation**
  - Simulation parameters for the MIP
  - Analog to Digital Converter (ADC)
  - Signal and Background samples
- **Classification methods**
  - Simple classification methods
  - TMVA RNN methods
- **Results**
  - ROC curves
  - TMVA RNN Classifier outputs + cuts

- **Problem context**
    - Can RNN be used for particle detection?
    - Comparison of RNN performance with a conventional algorithm for their ability to separate signal from background noise.
    - To compare, we need to simulate data

Code architecture

- json file to set certain values
- data simulation used to create data file, processed by simple algorithm and RNN
- draw and results to write and print results

```
##################################
###         ARCHITECTURE       ###
##################################

config1.json
|
|_ data_simulation.py
        |
        |_data_file.py
        |         |
        |         |_simple_algorithm.py  ________________
        |         |                                      |
        |         |_RNN.py                               |_results.py
        |             |                                  |
        |             |_training.py _____________________|
        |
        |_draw.py
```

Data simulation

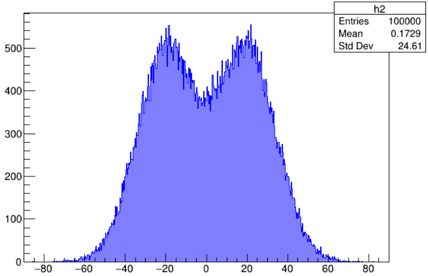
- In the data simulation file, we have a ClusterSimulator class, which simulates Minimum Ionizing Particles (MIPs).

- json file retrieves values for detector thickness and width, noise, digitization and signal cutting.

```
class ClusterSimulator:

    def __init__(self, config_file):
        self.config_file = config_file
        self.load_config(config_file)

        #Lists that compares who has the highest factor between MIP and 2MIP

        self.P=[]
        self.L=[]


    def load_config(self, config_file):
        with open(config_file) as f:
            config = json.load(f)
        self.b = config["b"]          # digitalization
        self.r = config["r"]          # signal cutting
        self.t = config["t"]          # thickness
        self.w = config["w"]          # width
        self.noise = config["noise"]  # noise
```

Angular distribution

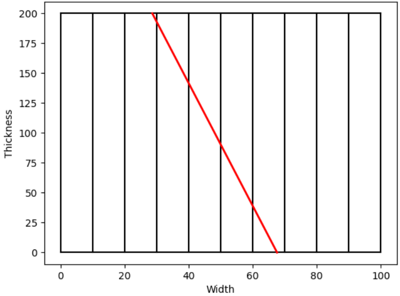
- the angle of incidence of the particle is randomly generated so that it is not too grazing, and not perfectly perpendicular either degrees



Theta distribution (in degrees)

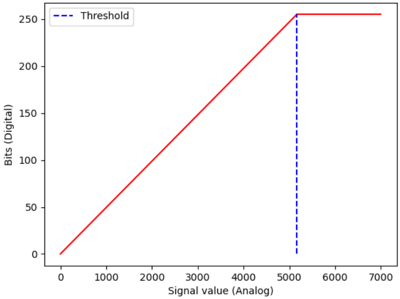
What does the particle's passage look like?

- random selection of initial position
- Initially, the charge deposited on a track is linearly dependent on the distance covered on this track.
- Then, we add a cross-talk effect on neighboring tracks, then the noise and a threshold



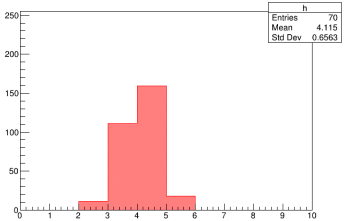
Analog to Digital Converter

- A crucial aspect of our simulation is the conversion of analog signals, generated by the passage of particles through the detector, into digital data via an Analog-to-Digital Converter (ADC).
- The number of bits is defined in the json file (8 bits here). We have calibrated our ADC to reflect the performance of real devices, taking noise and resolution into account.
- this ADC is used to return clusters with MIP and 2MIP functions

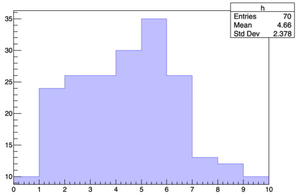


Clusters obtained

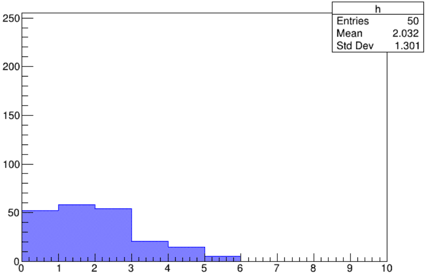
- We have a function that simulates a particle at MIP



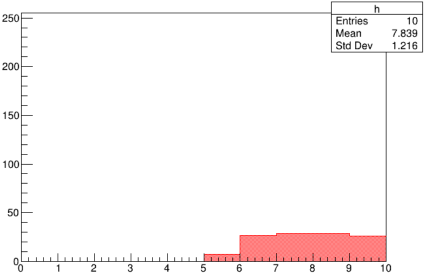
- We also have a function that simulates 2 MIPs



Some clusters are more complicated

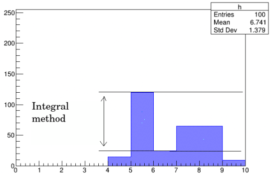


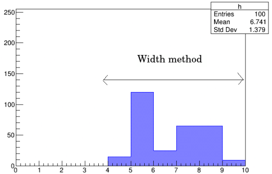
example for 2 MIP

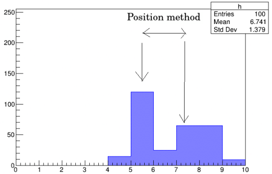


example for 1 MIP

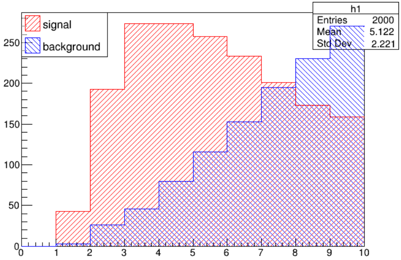
- **Simple classification methods:**
  - integral method
    $\longrightarrow$sum_Lebesgue(y_{max} : $max(y_{max}+1, y_{max}-1)$)
  - width method
    $\longrightarrow$abs($x_i - xf$)
  - position method
    $\longrightarrow$abs($x_{y_{max1}} - x_{y_{max2}}$)
  - charge method
    $\longrightarrow$sum_Riemann($x_i : x_f$)
  - ratio method
    $\longrightarrow$charge/width



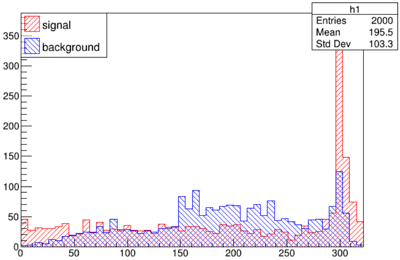




Classification methods



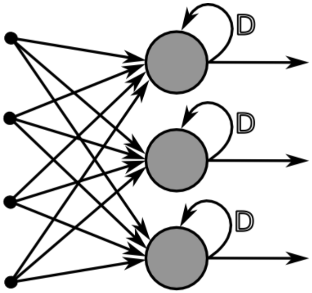
Hypothesis test for width



Hypothesis test for charge

Classification methods

- **Recurrent Neural Network (RNN)**
  - Use the output of a node as an entry
  - **How to use it**:
    Generate samples of signals and background.
    Take a % of samples to training and use the rest for the test.
    Training is optimize with the gradient descent algorithm.
  - **Hyperparameters:**
    epoch: number of training
    batch: subset of the training sample



Classification methods

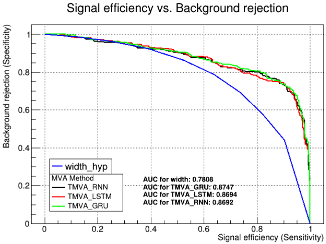
- **TMVA methods**
  Framework in ROOT to be used for classification and regression problems with various multivariate analysis (MVA) methods available.

  The input parameter is the resolution (x-axis), the output is set to 1 and the nodes in the hidden layer are 8.

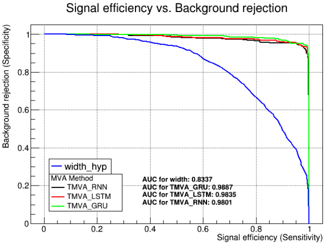
  - Vanilla RNN
  - Long short-term memory (LSTM) :
    $\longrightarrow$ Same principle as a RNN but avoids the vanishing gradient problem.
  - Gated Recurrent Unit (GRU) :
    $\longrightarrow$ Variant of the LSTM method

For the following results, we generated 1000 signal samples and 1000 background samples.
80% were used for training and the rest for testing.

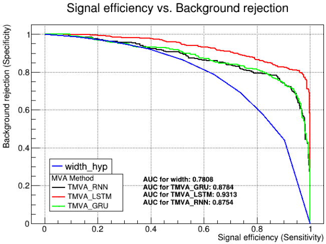
Results



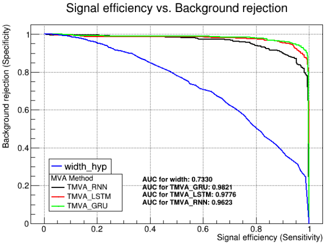
ROC curve (b=256,r=10)



ROC curve (b=256,r=100)

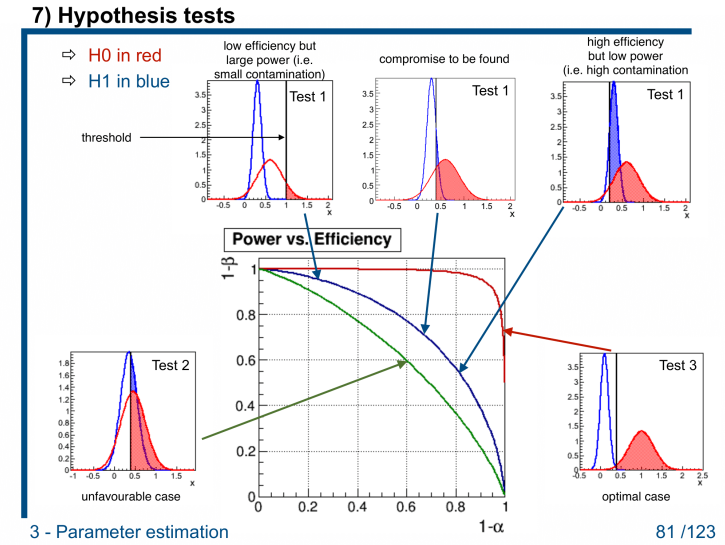


ROC curve (epoch & batch)

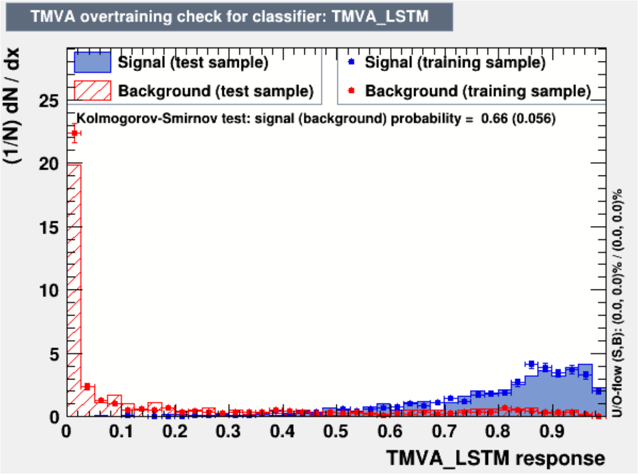


ROC curve (b=1024,r=100)

Quickly reminder



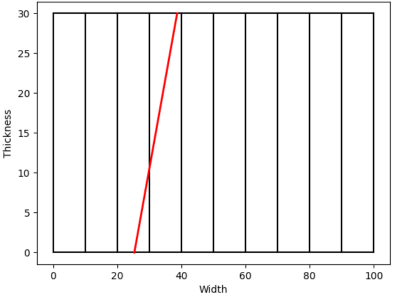
Results



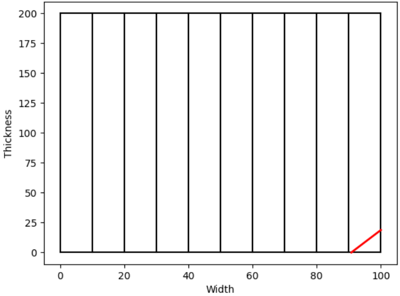
- Kolmogorov-Smirnov test higher than 0.01 $\longrightarrow$ no overtraining !

Some references

- https://indico.fnal.gov/event/17409/contributions/42949/attachments/26558/32939/Conley_2018June20_SNBMeeting.pdf
- https://root.cern.ch/download/doc/tmva/TMVAUsersGuide.pdf

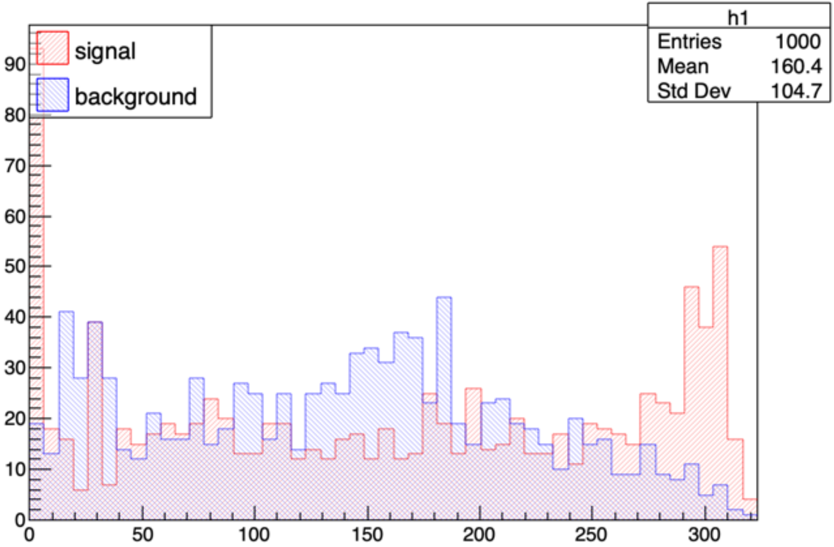


example of trajectory

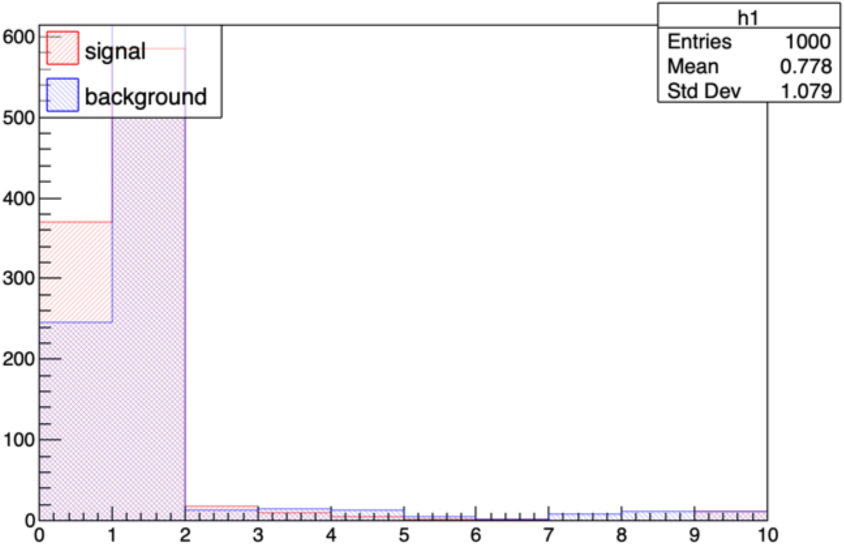


example of another trajectory

hypothesis test integral



hypothesis test position



hypothesis test ratio