Istituto Nazionale di Fisica Nucleare
SEZIONE DI FIRENZE

Teaching Material on the KB

Lucio Anderlini, Vitaliano Ciulli, Matteo Turisini

- LHCb Masterclass, with Keras
- MNIST in a C header
- An introduction to classification with CMS data

Introduction

Improving the accessibility of learning opportunity is a fundamental part of our mission.

With the Covid-19 emergency, most of the learning opportunities were cancelled and we tried to enhance remote solutions.

At the last meeting, it was proposed to **share the teaching material** of the single institutions in order to provide help in offering teaching opportunities to small bunches of local students, according to the regulations of the various institutions.

In Florence, we collected **three "*tutorials*"** that were prepared to get some students started with data analysis and machine learning and we uploaded it to the KB.

The material is divided in two categories:
1. *material getting few students (1-2) started on a very specific topic*
2. *material for lectures and hands-on of a whole class.*

Among the contributions discussed here, there are examples of both categories, as we believe both may serve to the purpose.

1. LHCb Masterclass, with Keras

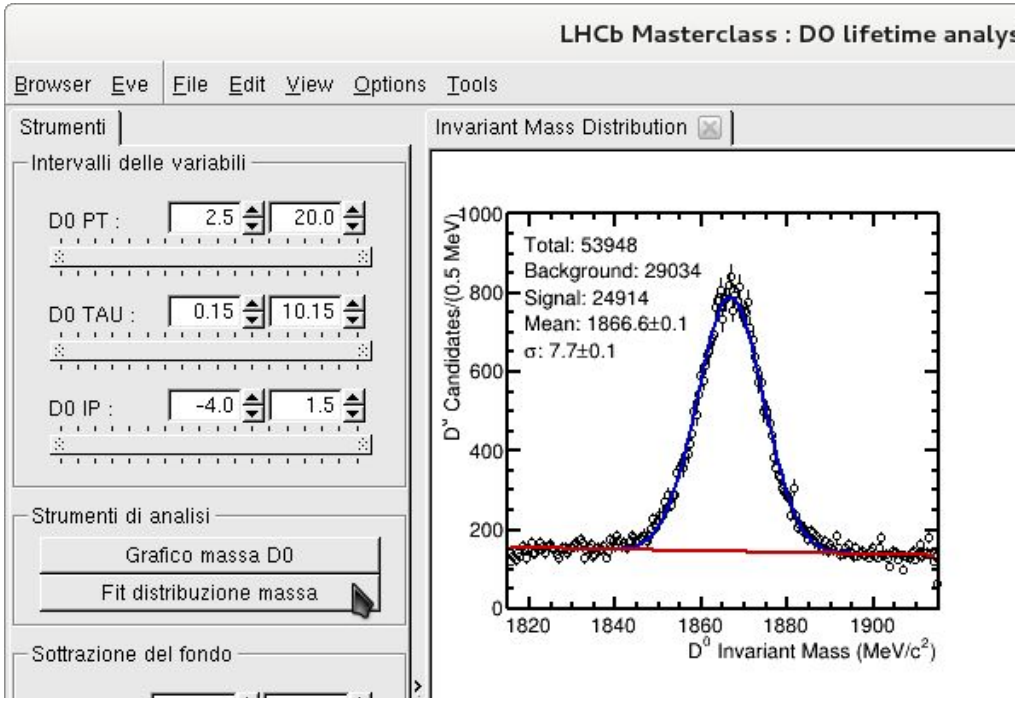
Difficulty: 🙂
Category: *generic introduction*
Target: *students approaching python for data analysis*

LHC masterclasses are 1-day classes offered to master students to get started to the data analysis of the LHC events.

After a few introductory talks, ATLAS, CMS, LHCb and Alice (and recently other non-LHC experiments) have designed exercises that the student can play with in front of a computer to familiarize with concepts such as signal, background, ...

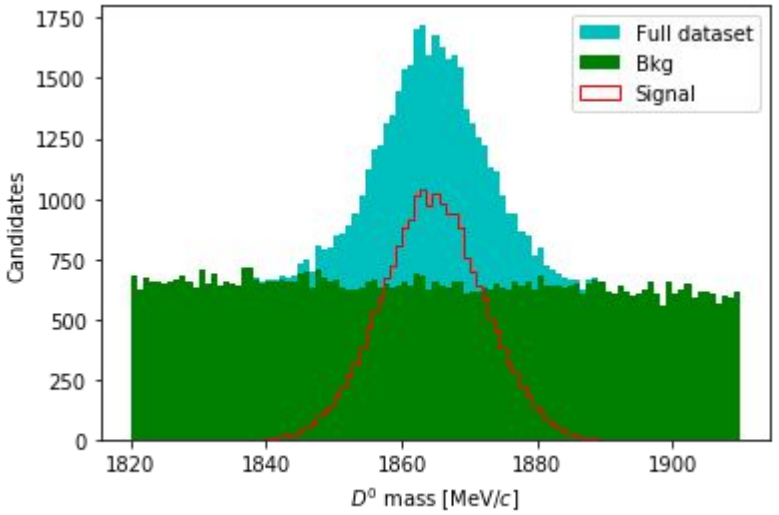
Behind the scenes,
the exercise proposed by LHCb make usage of advanced statistical tools (maximum likelihood fits, sPlot, statistical significance) that bachelor students are often not confident with.

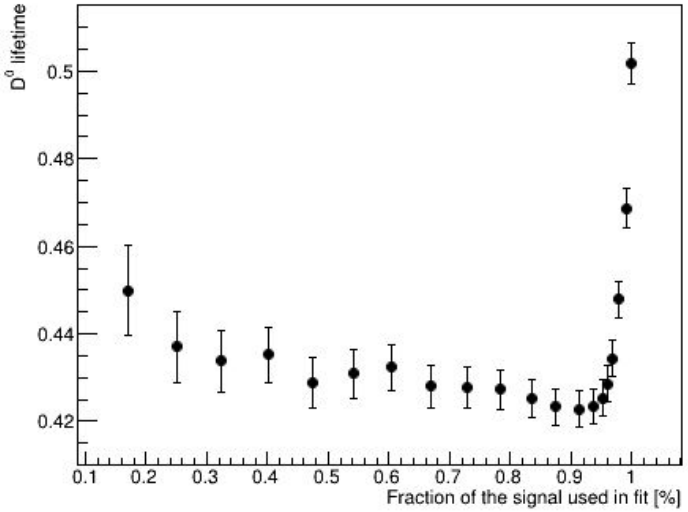
We tried to go behind the scenes in a Jupyter notebook.

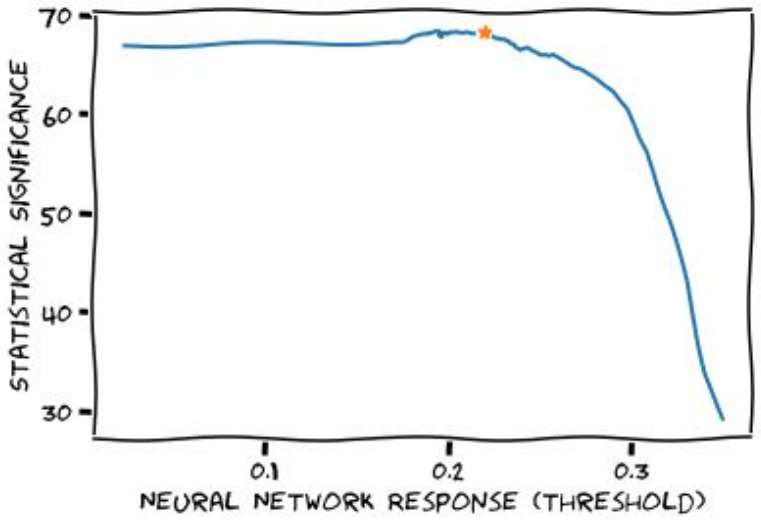


1. LHCb Masterclass, with Keras – *Items discussed*

- Accessing ROOT data with root_pandas
- Plotting with PyROOT and Matplotlib
- Fitting with RooFit via Python
- Modeling signal and background
- Statistical subtraction of the background
- Training a simple NN on sWeighted events
- Plotting and reading the ROC curve
- Optimize the selection based on some metric
- Fitting the D^0 lifetime
- Distinguish sources of uncertainties (stat & syst)







2. MNIST in a C header

Difficulty: 😎
Category: *very specific*
Target: *nerd students*

In the life of a physicist, the time arrives when you get disgusted by the software dependencies that evaluating a simple neural network requires and you want the plain, pure and clean function to be evaluated in a whatever context.

Examples:

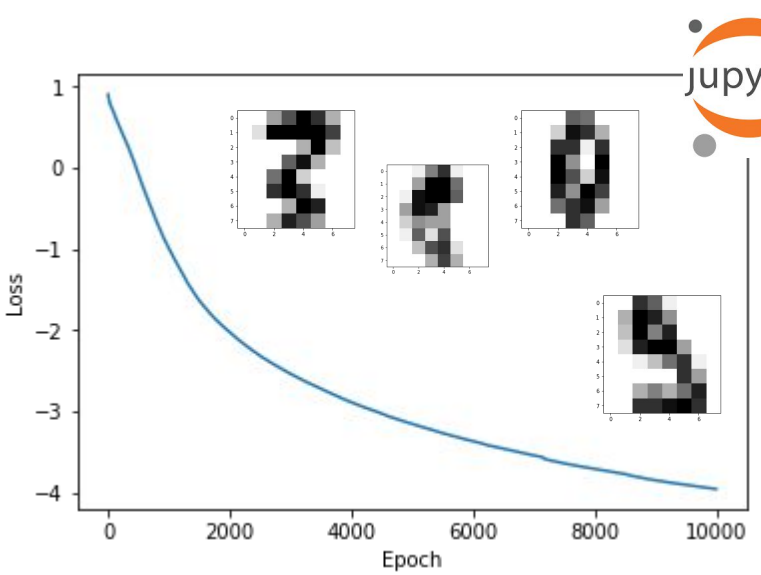
- run a simple NN on Arduino
- include in your "consolidated" C++ analysis program a NN
- generate a NN module for a FPGA device using HLS to convert C in HDL (Hardware Description Language)

We amused ourselves in developing a simple **conversion of a Forward, Dense MLP** trained to solve the *"handwritten digit classification" (MNIST)* problem into a **single C header including everything** (weights and functions).

Disclaimer. Do not try this at home: more robust tools exist to achieve similar results for specific applications in a much more elegant fashion. This is at most pedagogical.

2. MNIST in a C header – *Items discussed*

1. Training a MLP on reshaped 2D data (handwritten digits) using Keras
2. Reading back the structure and weights of the trained neural network
3. Parse the structure into a C-language function with hard-coded weights
4. Include the auto-generated C header in a C/C++ program
5. Compare the performance of the exported NN with the original python-based one



Ctrl+C
Ctrl+V

myNN.h C header

```
Some C workflow

/*your headers*/
#include "myNN.h"

int main ()
{
  /*your program*/
  myNN (yourinput);
  /*your program*/
}
```

3. An introduction to classification with CMS ~~data~~ simulation

Difficulty: *from* 🧐 *to* 🤯🤯🤯🤯
Category: *HEP-focus introduction*
Target: *python-proficient students, moving first steps in statistics*

Classification of signal and background is the most frequent problem in HEP data analysis.

We prepared an introductory set of Colab notebook to explore the problem of classification, taking as use-case the selection of the decay

$$H \rightarrow W\,W \quad \text{with both} \quad W \rightarrow l\,\nu$$

Introduction: what are LHC, CMS and the Higgs boson

🧐 Univariate classification: Hypothesis Testing, ROC

🤯 2D linear algorithms: Fisher, Logistic regression

🤯 Decision Trees and forests using scikit-learn

🤯🤯🤯🤯 Neural Networks using plain TensorFlow2 and then Keras

The aim of the notebooks is to illustrate the algorithms, while giving Python for granted. Consider preparing your students to this with a primer of Python.

The problem

We want to distinguish the WW events produced in a Higgs decay from the WW events produced via non-resonant production mechanisms in the *pp* collision.

An available simulated dataset allows to statistically describe:

- The signal (pp -> H -> WW -> l nu l nu)
- The background (pp -> WW -> l nu l nu)

The variables (preselected by CMS analysts) are, for example:

- The invariant mass of the two leptons
- The missing transverse mass
- The angle between the two leptons
- The number of jets in the event

Discriminant power (KS distance)

```
        label:   100.0%
          mll:    62.1%
          mTi:    51.8%
         drll:    48.6%
       dphill:    40.6%
          mth:    19.9%
     dphilmet:    15.9%
  metPfType1:    13.8%
         ptll:    11.0%
   dphillmet:     8.4%
         njet:     4.4%
  metPfType1:     1.5%
```

Processing ROOT data in Colab

[1st colab]

Colab is a Google service to run jupyter notebooks accessing data from Google Spreadsheet and Google Drive.

CMS data in nTuple ROOT format has been loaded on Google Drive and is accessed through *uproot.* The code has been written to avoid dependency on the Colab environment.

```
!pip install uproot
files = {
    "WWTo2L2Nu1.root": "1oix_LaKYaPgMPdOcG8O6_qwq3a0CD1EO",
    "GGToHToWWTo2L2Nu1.root": "1lD3JG12bP21oN94bEmDOZuLQBd4y2Dwk",
    "WWTo2L2Nu2.root":"1KTlRb66tSPajEpJJoLqmqisBQW-Pakjk",
    "GGToHToWWTo2L2Nu2.root": "1X72-i7qrYqQyA02jCESFFlSq33__x_tq"}
!rm -f *.root
import os
for kv in files.items():
  if not os.path.exists(kv[0]):
    a = os.system ( "wget -O %s --no-check-certificate
'https://docs.google.com/uc?export=download&id=%s'" % kv )
    if a: raise IOError ( "Could not download %s (%s)" % kv )
```

Non-standard **uproot** library is downloaded and installed at the first execution of the Jupyter runtime.
This may require to run this cell more than once after runtime reinitialization.

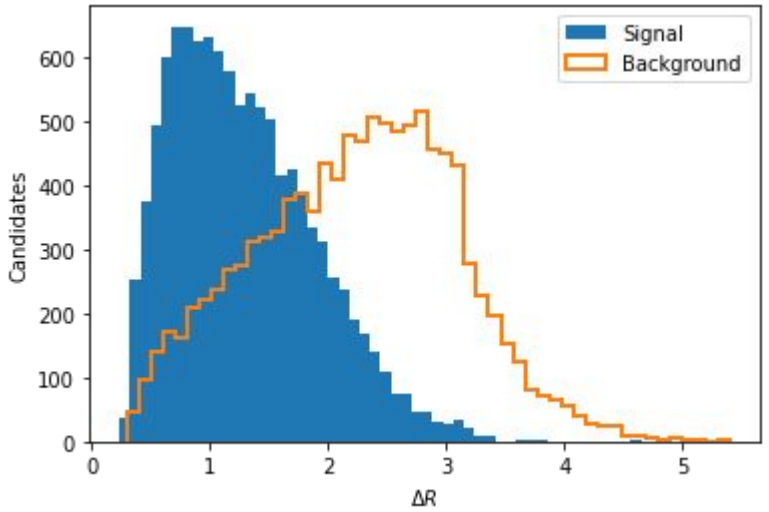
Linux wget command is used to download the data from Google Drive.

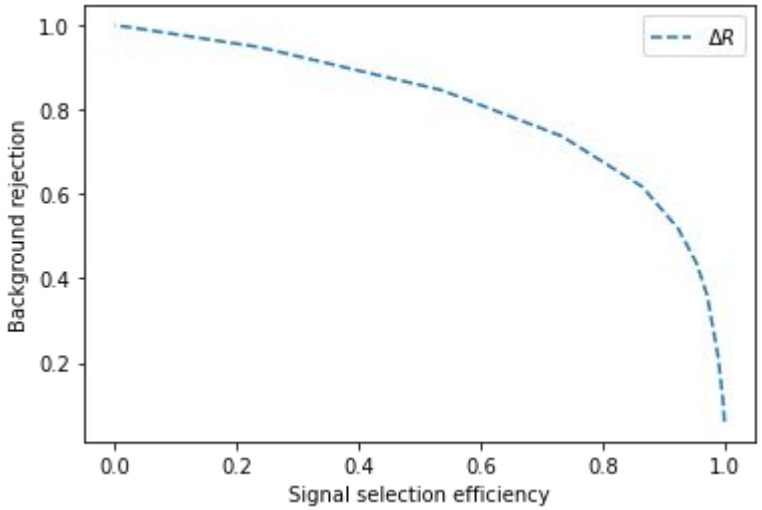
Test-statistic and ROC curve

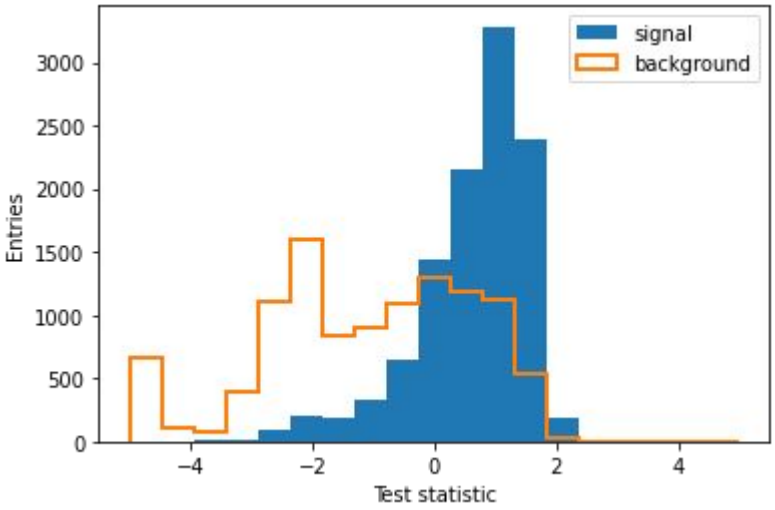
[1st colab]

The first notebook is devoted to play with the loaded data.

- Plotting histograms of the variables
- Computing 1D likelihood ratio to define a test-statistic
- Study the test-statistic to define a ROC curve

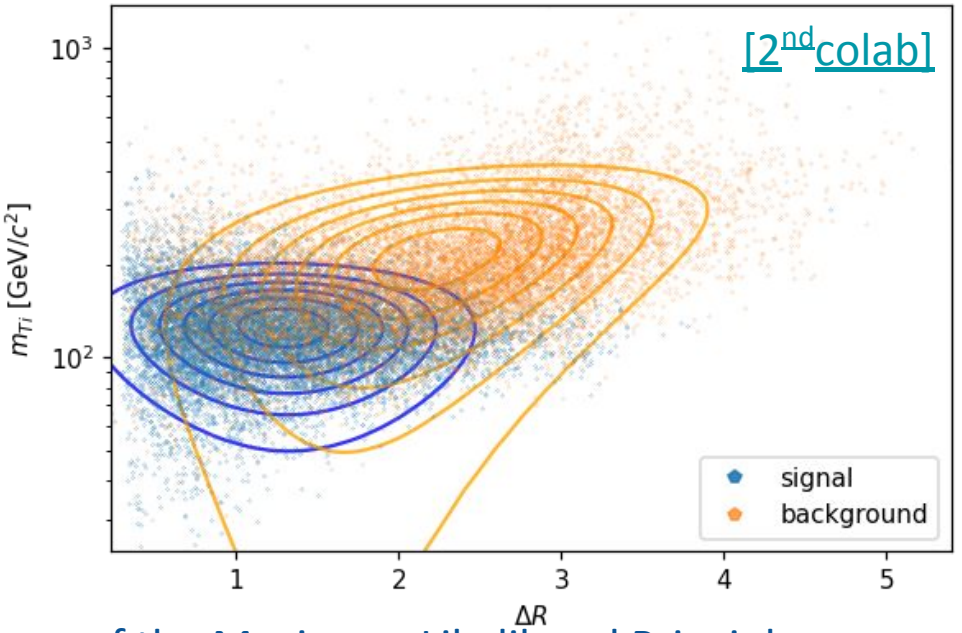




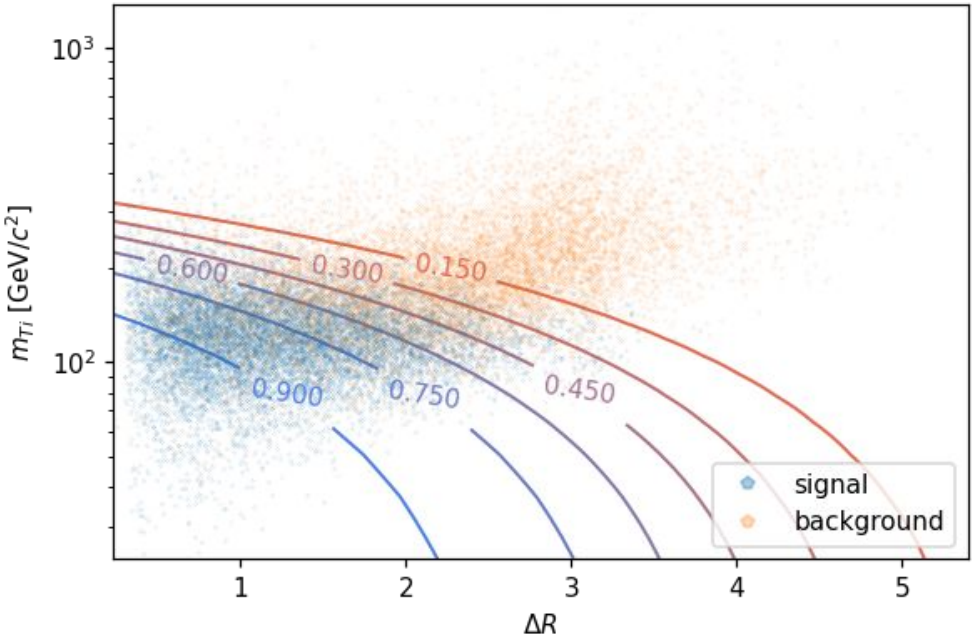


Fisher discriminant

The 2nd notebook is devoted to linear algorithms. The Fisher discriminant is introduced by fitting two multivariate Gaussians to the two categories and maximising both analytically and numerically their separation.



Logistic Regression as a consequence of the Maximum Likelihood Principle



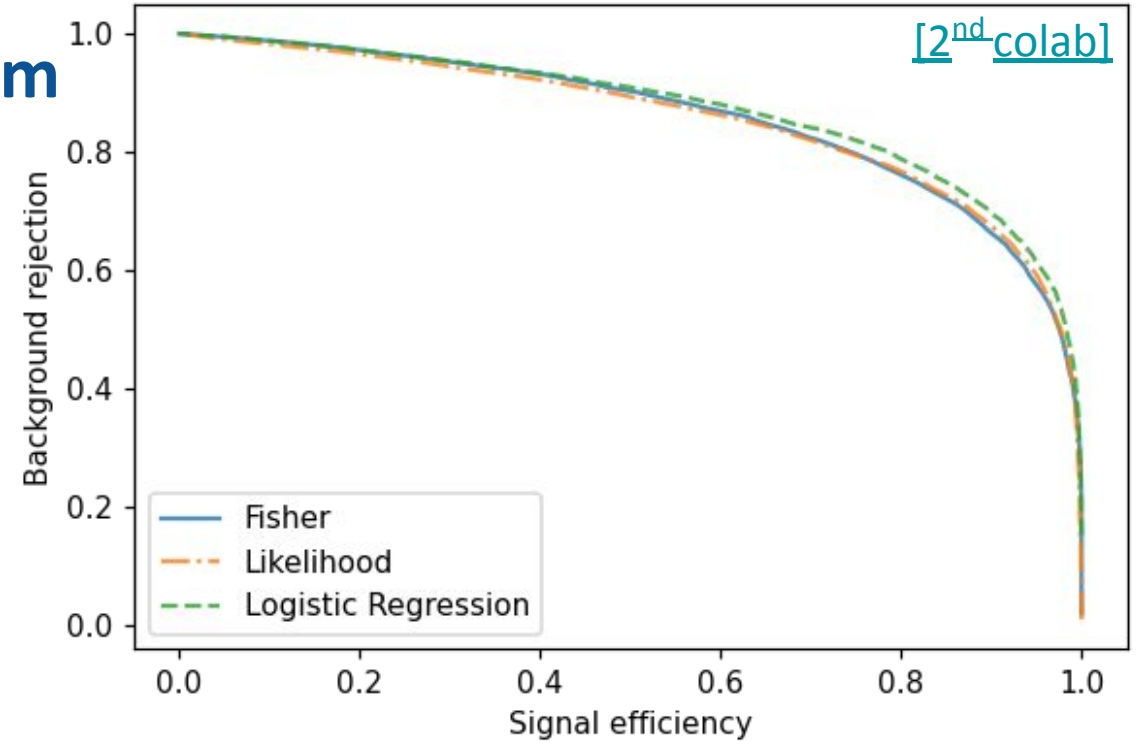
Dropping the hypothesis on the Gaussian nature of the two distributions and renouncing to analytical optimization, we introduce Logistic Regression applying the maximum likelihood principle to the Bernoulli distribution.

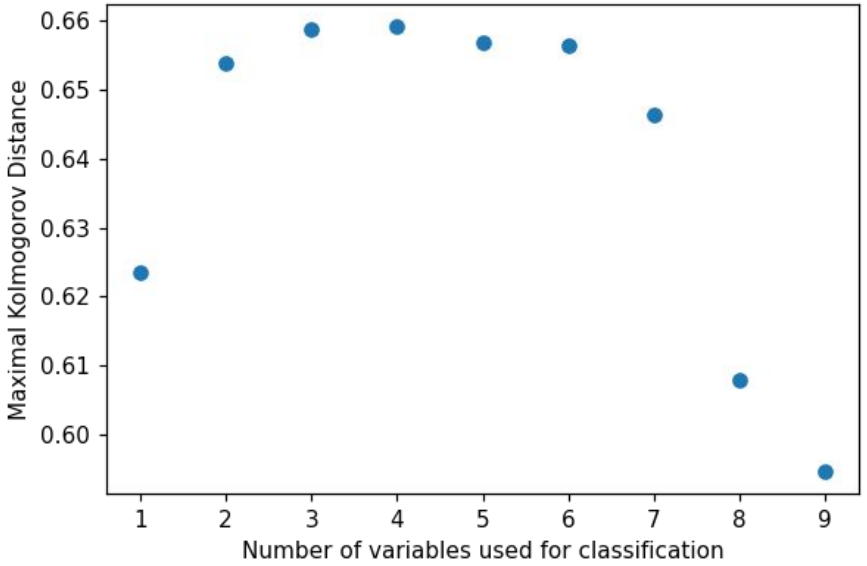
No-Free-Lunch Theorem

a. k. a. hypotheses matter

Comparing the approximated likelihood ratio, the Fisher and the Logistic regression we observe that the Gaussianity-hypothesis (common to Likelihood and Fisher) is detrimental.

[2nd colab]





The curse of dimensionality

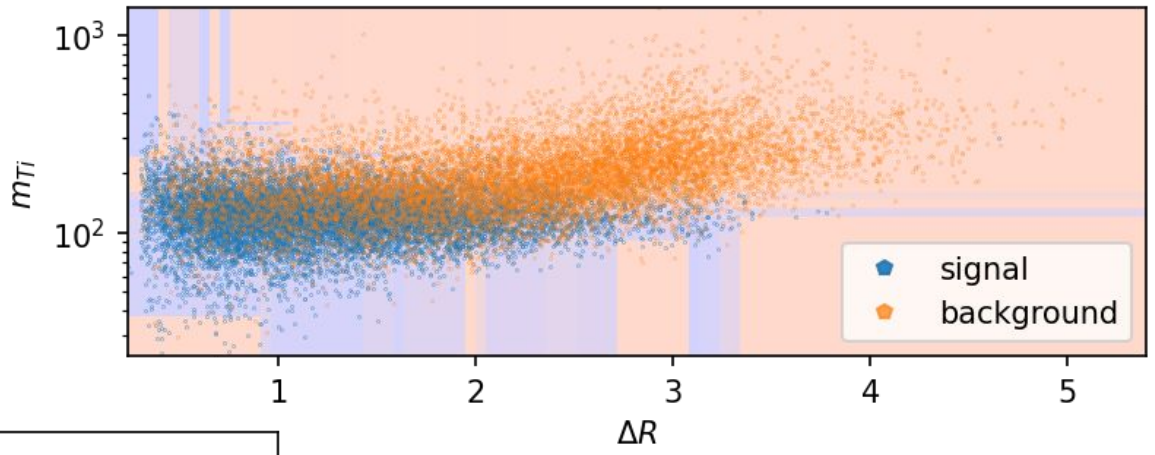
For each "number of variables in combination", we train a Fisher for all the possible combinations of variables.

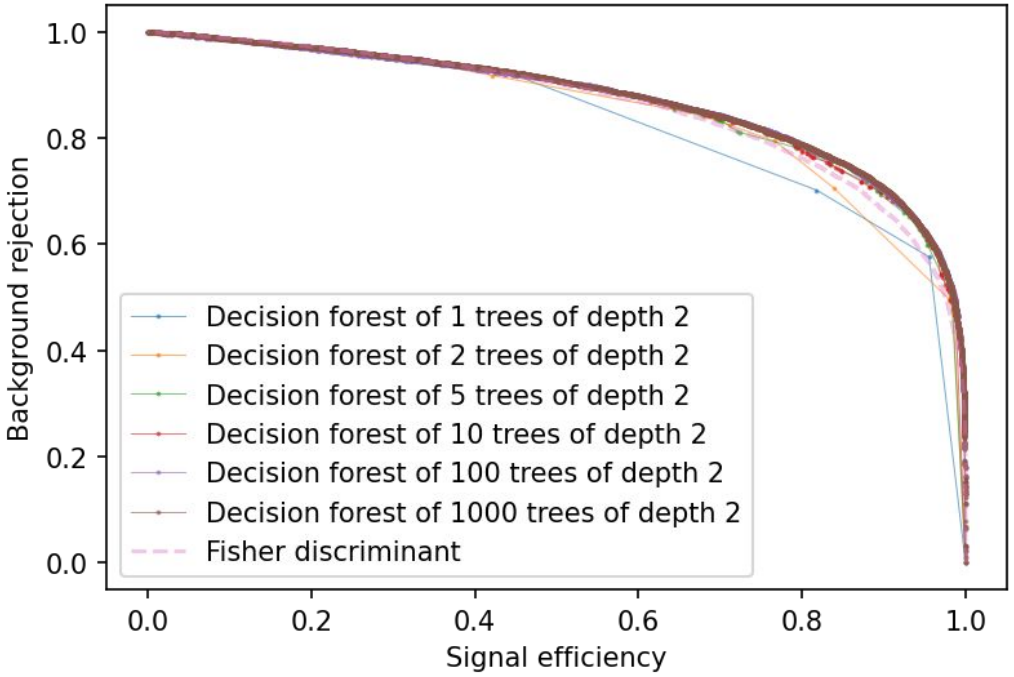
Adding more than 3-4 variables worsen the performance of the algorithm.

Boosted Decision Trees and Random Forests

[3rd colab]

In notebook 3 we start discussing single decision trees on 2 variables to visualize the decision boundaries.





Then we introduce ensemble learning by averaging the response of up to 10^3 random decision trees, trained independently one from another.

Already with 5 trees of depth 2 we equal the Fisher discriminant.

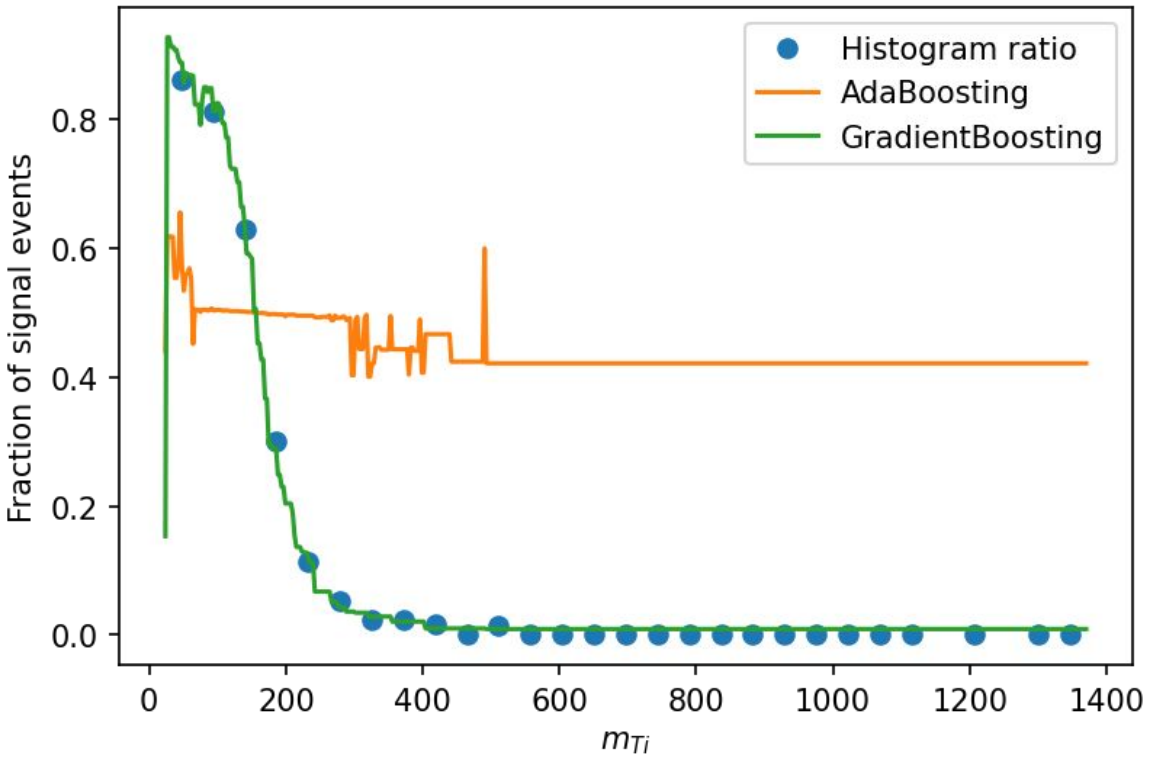
Boosting Decision Forests

[3rd colab]

Then we discuss techniques to combine several decision trees in a smarter way than randomizing and averaging: the *Boosting*.

We introduce:

- *AdaBoosting*, based on event weighting
- *Gradient Boosting*, based on the Maximum Likelihood Principle



The two boosting techniques result in similar performance, but different meaning of the BDT response.

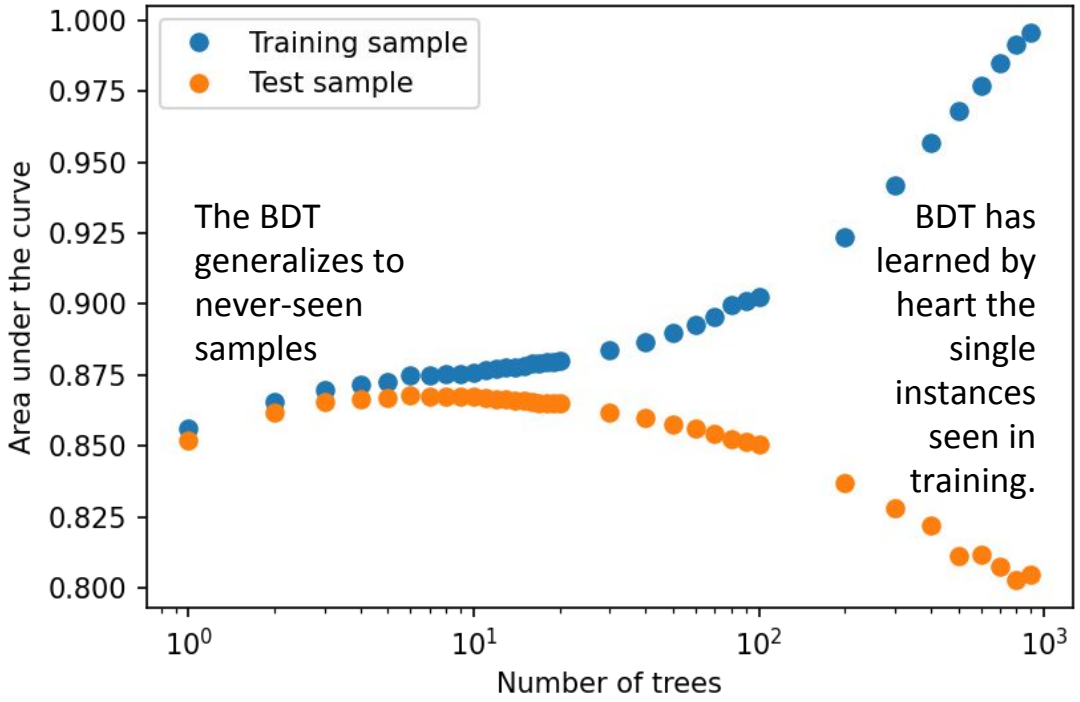
Beware of Overtraining

[3rd colab]

Boosted decision tree are very prone to overtraining.

To monitor for overtraining we split the dataset in test and train sample and observe how the discriminant power of the BDT response evolves with the number of trees contributing to the decision.

Once the discriminant power starts decreasing increasing the number of trees, that's overtraining.

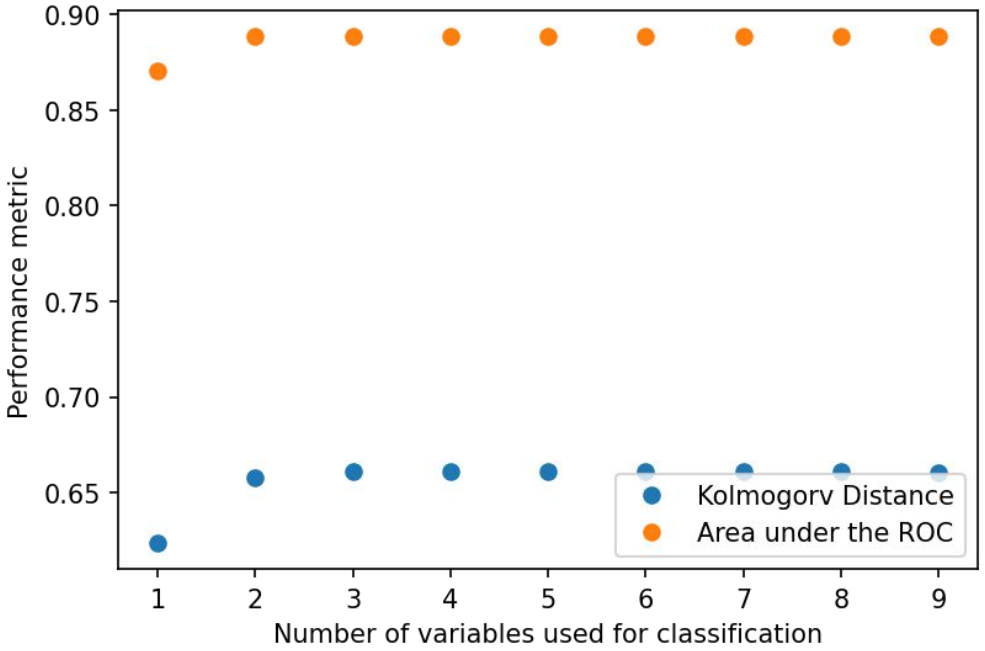


Robustness against the curse of dimensionality

[3rd colab]

We discuss the robustness of the decision trees to the curse of dimensionality and we observe they are much more robust than the Fisher discriminant (because of the weaker hypotheses).

We perform the exercise comparing two different metrics and ensuring that neither worsen adding more than the optimal number of variables.

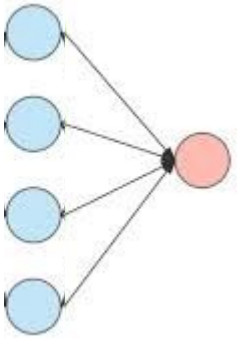


Neural Networks

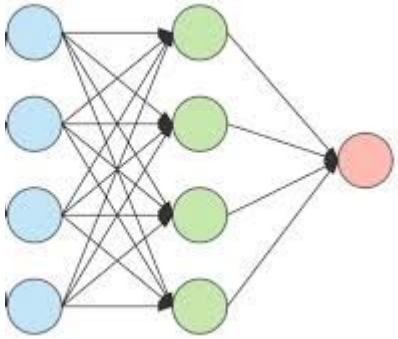
[3rd colab]

- Implementing the Logistic Regression in Plain Tensorflow 2
- Extending it to a Neural Network by learning a better representation

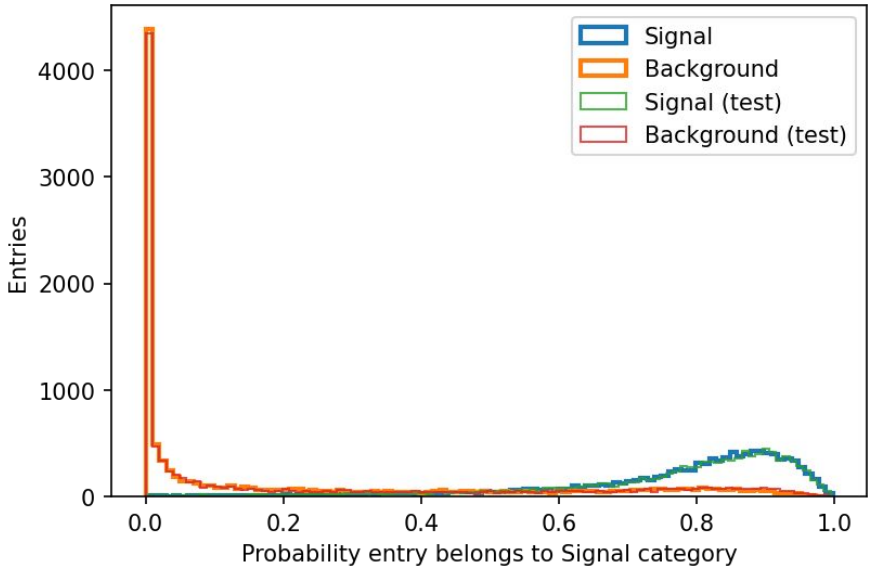
Logistic Regression

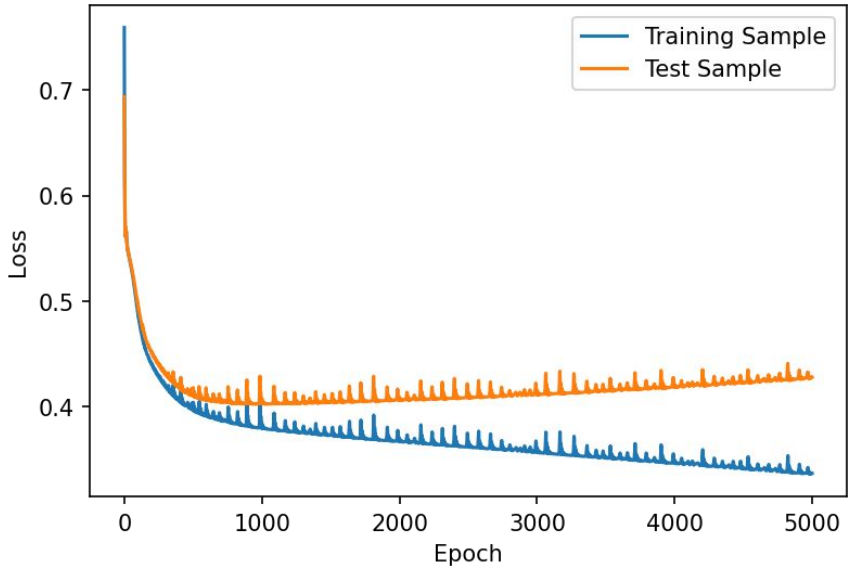


Neural Network



Understand the meaning of the prediction and beware of overfitting.





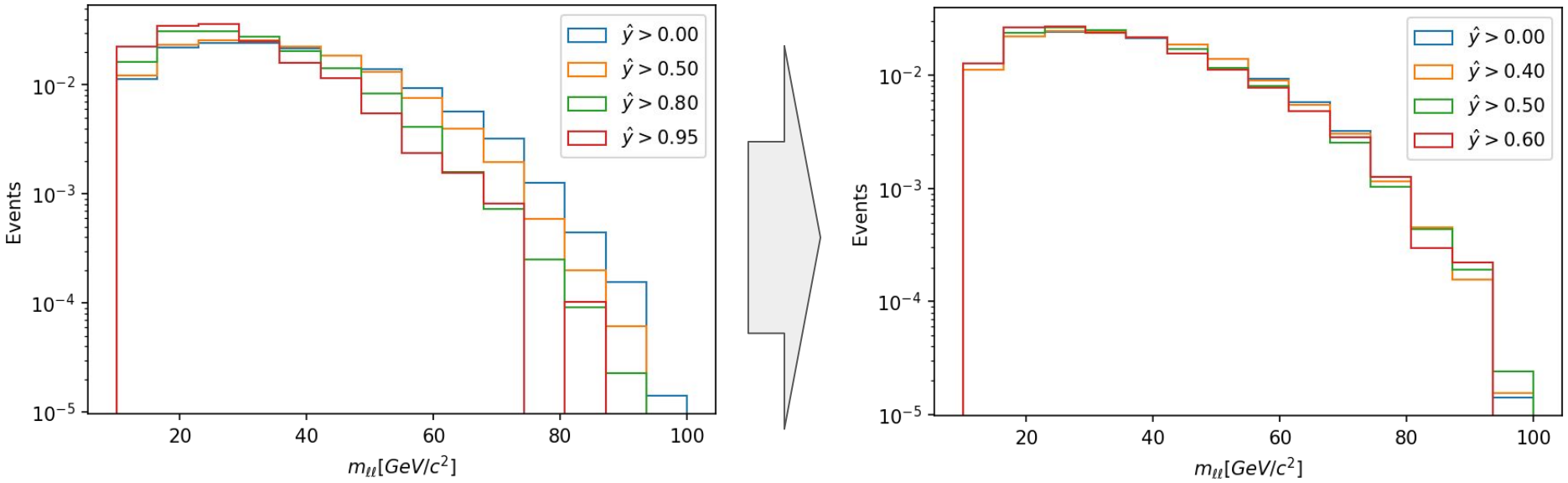
Adversarial Training for Domain Adaptation

[3rd colab]

Finally we try to go one step further training **systems of neural networks** taking as an example the domain adaptation.

We try to train a classifier requiring it to explicitly ignore the dependence on a given variable (for example the invariant mass of the two leptons).

To do this we train a second NN to guess the value of m_{ll} from the representation used for classification, and reward the classifier if m_{ll} is hard to guess.



Conclusion and feedback from students

We hope this material may help in the teaching and tutoring activities.

Room for improvement after testing ~~on~~ *with* students:

1. *LHCb Masterclass, with Keras*. Requires too advanced Statistics for students learning Python and Keras.
2. *MNIST in a C header.* Should be extended to be more generic.
3. *Classification with CMS data.* Requires too advanced Python for students approaching such foundations of statistics.
   Suggested timing: 2h (introduction) + 3 × 2h (lectures) + 3 × 2 (coding)

In summary, tuning the pre-requirements in terms of Python and Statistics experience to the specific audience is a bit challenging.

Consider integrating this material with an introduction to better tune the lectures to your audience.