PhyStat Seminar, CERN (online) - June 15 2022

# *Sticking to the Roots of Machine Learning: Old-School Approaches to Physics Analysis*

**Tommaso Dorigo**

INFN, Sezione di Padova

Istituto Nazionale di Fisica Nucleare

# Abstract

*In this seminar I will take a step back from the recent trend of DNN-izing everything, by discussing the merits of two recent results produced using statistical learning approaches to supervised and unsupervised learning problems of interest to HEP.*
*The first is the use of a 66-million-parameter k-NN algorithm for deep regression[1]. The second is the identification of anomalies in HEP data in a fully unsupervised way, exploiting the random sampling of their copula space[2].*

[1] https://arxiv.org/abs/2203.02841
[2] https://arxiv.org/abs/2106.05747

# Contents

- Introduction: Where we come from
- Can a kNN learn? Deep regression with nearest neighbors
- The copula space: Where to be in anomaly searches

Note: as I am trying to squeeze two seminars inside a single slot, I will have to skip some detail. Slides skipped are marked with a * in the title. If you are interested, ask a question during the seminar or at the end!

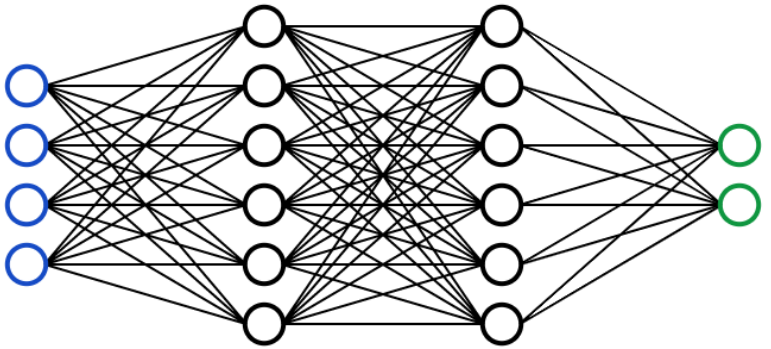
# Before 2012...

Long before most of us started thinking about MVA tools for HEP, statisticians developed and used a huge weaponry of statistical learning tools

- Linear discriminant analysis is a 1936 brainchild of R. Fisher
- k-NN was invented in 1951 (Fix, Hodges)
- Clustering dates back to 1938-39 (Zubin, Tryon)
- Cross-validation was invented in 1974 (Allen, Stone)
- Boosting comes from ideas of the late eighties, then Schapire 1990.

When machine learning boomed in the early 2000s, it was largely a computer-driven revolution rather than the discovery of new tools

# The rise of deep learning in HEP



DNNs use skyrocketed in HEP after 2012, when BDTs and NNs were used for the Higgs discovery (2012 is also the turning point in the imagenet challenge). A **true paradigm change**!

Further evidence of the benefit of ML tools for HEP was given by the Kaggle Higgs challenge [Kaggle 2014], with 1800 teams participating (physicists, statisticians, computer scientists).
Task: separate H→ττ decays from backgrounds in LHC simulated data

The most effective solution was based on a pool of DNNs, with emphasis on cross-validation

Alternative methods commonly used in HEP (xgboost, Bayesian NN, etc.) were beaten soundly

| Solution | Score |
|---|---|
| Gabor Melis (DNN pooling) | **3.806** |
| MultiBoost | 3.405 |
| TMVA boosted trees | 3.200 |
| Naive Bayesian classifier | 2.060 |
| 1D cut-based selection | 1.535 |

equiv. to 6 times more data!

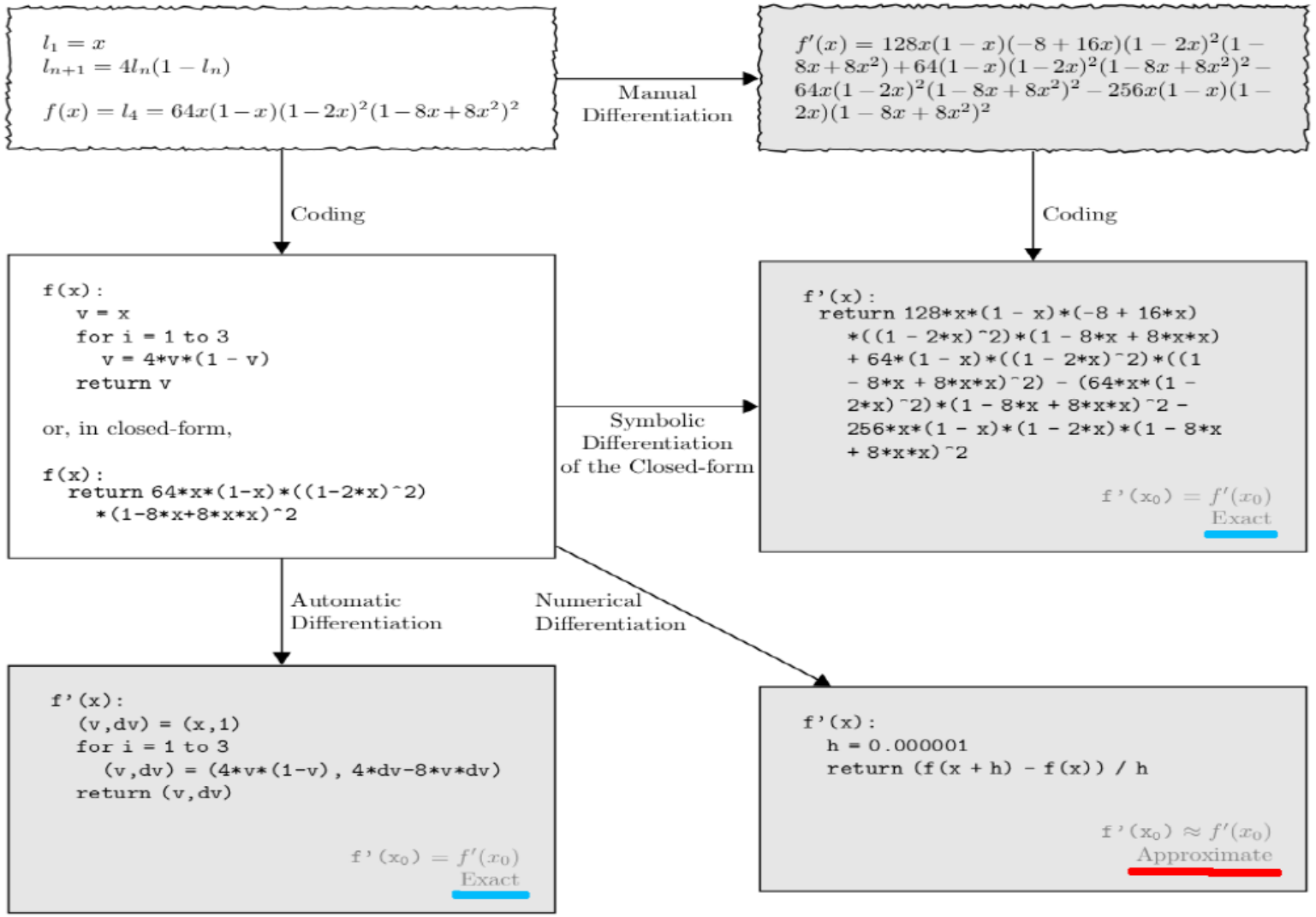
# * The weaponry of the 2020s

Current trends of DL applications in particle physics include

- DNNs for classification, particle ID, signal discrimination, deep regression
- CNNs for image-based classification (e.g. boosted jets)
- Graph NNs for event reconstruction
- Differentiable pipelines for incorporation of nuisance parameters in supervised learning tasks
- Implementation of NNs in FPGAs for online data acquisition
- VAEs for anomaly detection
- GANs for generative models

The above tools essentially use the same engine under the hood: the chain rule of differentiable calculus, which allows gradient descent

# * Automatic Differentiation




Wouldn't it be nice if you coded a problem and the dependence of variables in a program, and the language took care of figuring out how functions vary depending on parameters, and carry out the complicated task of propagating derivatives around?

Those of us who have done this manually can't be happier by seeing the rise of Pytorch, TF, etc.

# Manual differentiation

Automatic differentiation is great, and is speeding up progress... But manual differentiation also works!

In this talk I will discuss two results that I produced with unrefined c++ code, by manually implementing a calculation of a loss function and its derivatives in two different contexts

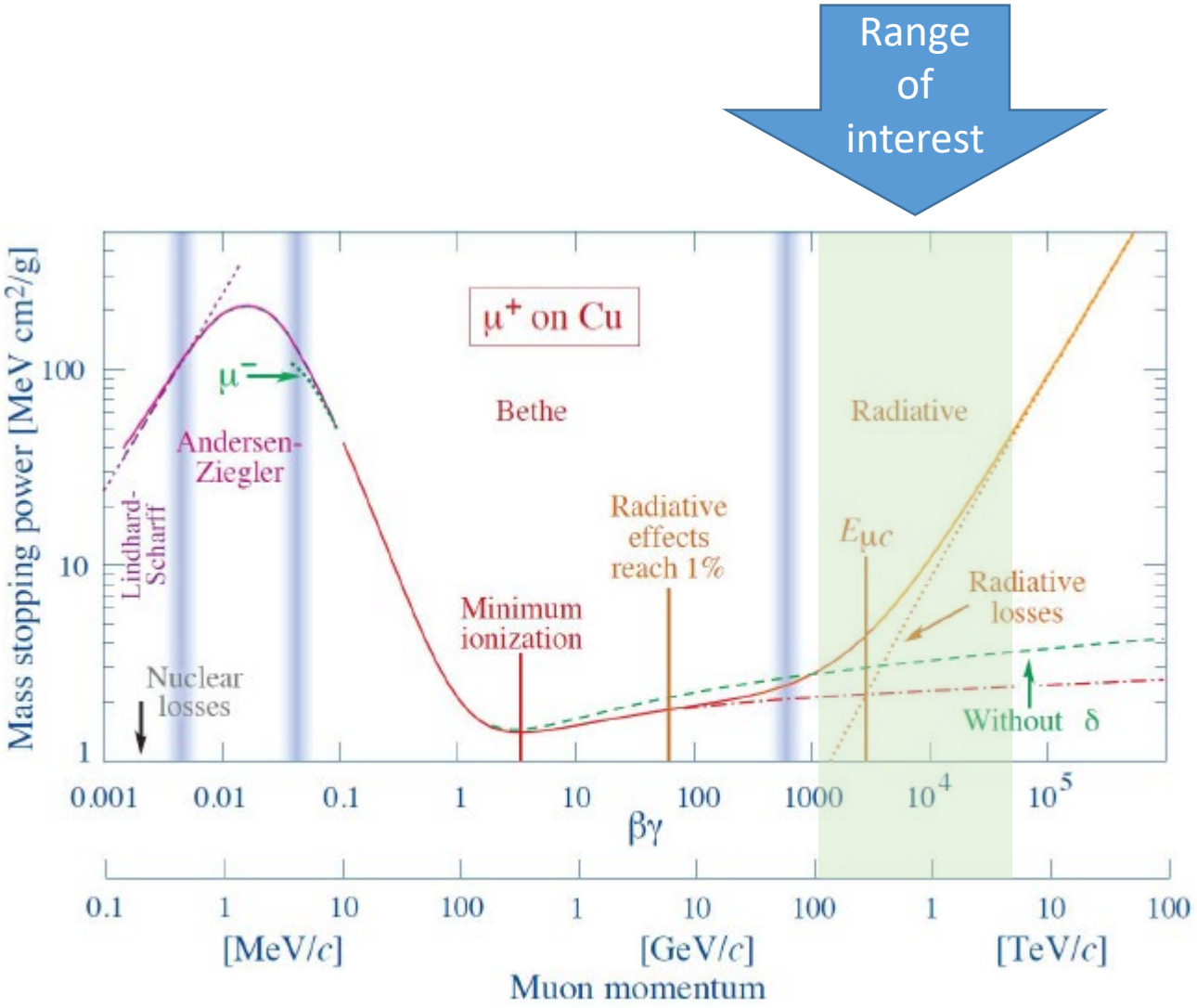
- Measuring TeV-muon energy with a granular calorimeter
- Finding overdensities in a multi-dimensional space

The message will be that what is really important is the statistical model, not the tools that we apply to it to get our answer...

# Part 1: Measuring muon energy in a calorimeter

Muons are minimum-ionizing particles, and they do not produce EM showers as they traverse dense materials

Their behavior changes above a few 100 GeVs, when they start to radiate soft photons in significant amounts – but still, even then the energy loss in a thick calorimeter is typically of the order of a few percent



# What to do above a few TeV?

In the past, muons were crucial to discover, e.g., the Upsilon, the W/Z, the top, the Higgs...

Future colliders of energy higher than that of the LHC might produce new particles whose signature involves decay to ultra-high-energy muons. But above a few TeV we cannot rely on magnetic bending to measure their momentum:

- CMS has $\delta E/E = (0.06 : 0.17)\ E/\text{TeV}$
- ATLAS has $\delta E/E = (0.08 : 0.20)\ E/\text{TeV}$

→ above 3-4 TeV muon energy becomes unmeasurable

Can we increase the bending power of our detectors? Hardly...

Can we use radiative losses? Not by ordinary means...

# Original observation, and two questions

While training DNNs to reconstruct EM signals in the HGCAL, J. Kieseler found that the networks could to some extent learn the energy of very high-energy muons

We decided to investigate the effect in detail, putting two questions on the table:

- Is there information in the spatial distribution of the energy deposits?
- To what level can multi-TeV muon energy be measured in a very high granularity calorimeter?

The study brought us to develop a very complex CNN model

# Simulation of a high-granularity calorimeter

We produced with GEANT4 high-statistic samples of muons interacting in a high-granular, homogeneous $PbWO_4$ calorimeter

total depth = 2032 mm = 10 $\lambda_0$

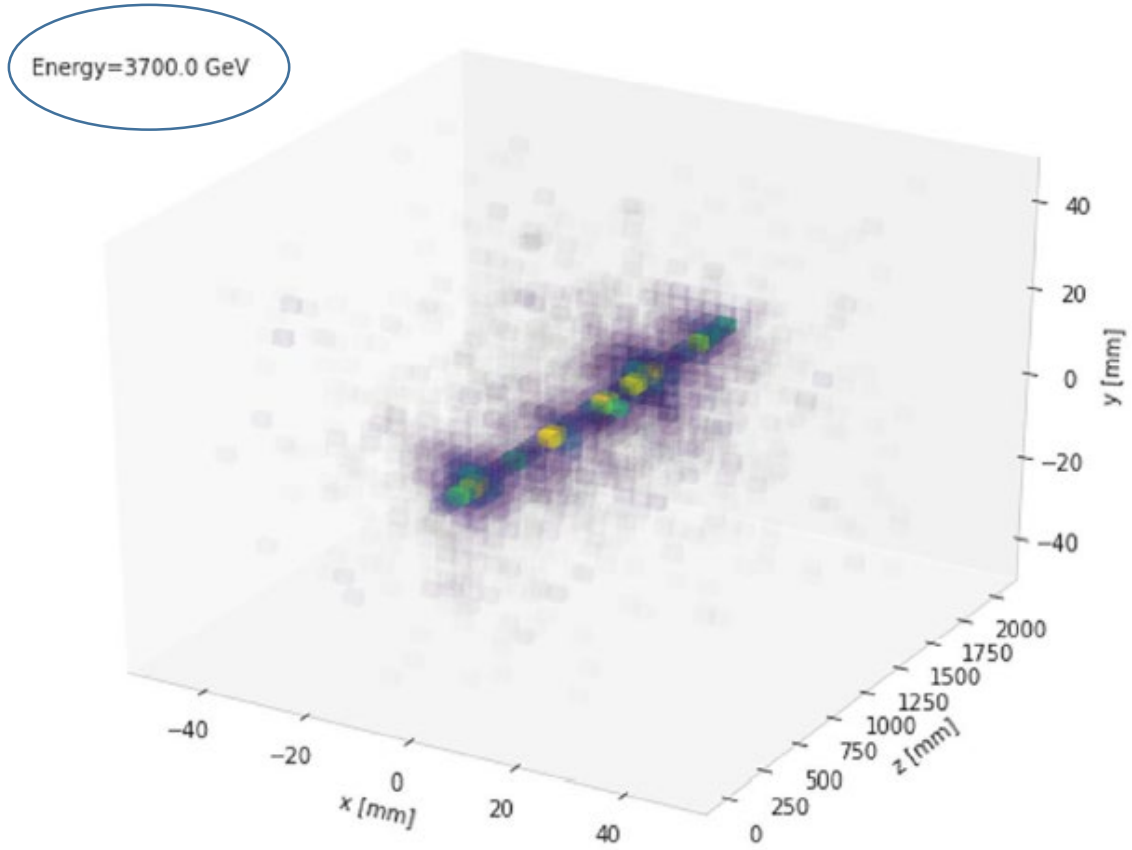
50 layers of 32x32 cells –

51,200 channels in total

cell size = 3.7 x 3.7 x 39.6 mm

For added realism, we assumed the calorimeter is embedded in a 2-Tesla B-field orthogonal to the muon incident direction. This has practically no effect for >1 TeV muon trajectories (<1mm deflection)

We simulate about 900k muons in the [0.1-8 TeV] range to train and validate our models, and some further fixed-energy samples for testing.

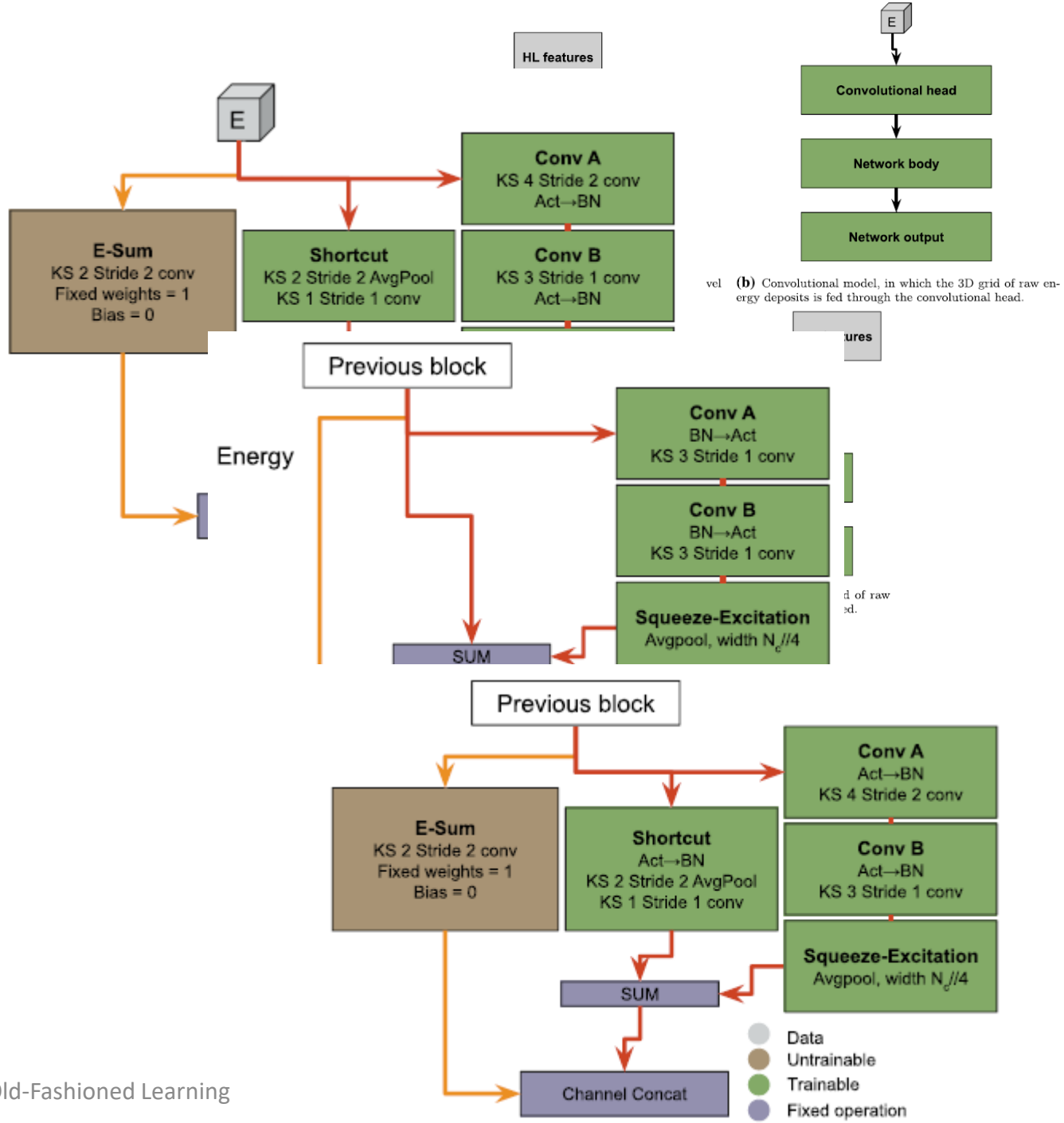



Note: this looks like a shower, but most photon deposits shown are of few MeV only

# * CNN model

The CNN we originally deployed [4] uses convolutional layers to reduce the dimensionality of the pattern of energy deposits, and employs 28 high-level features in a dense layer.

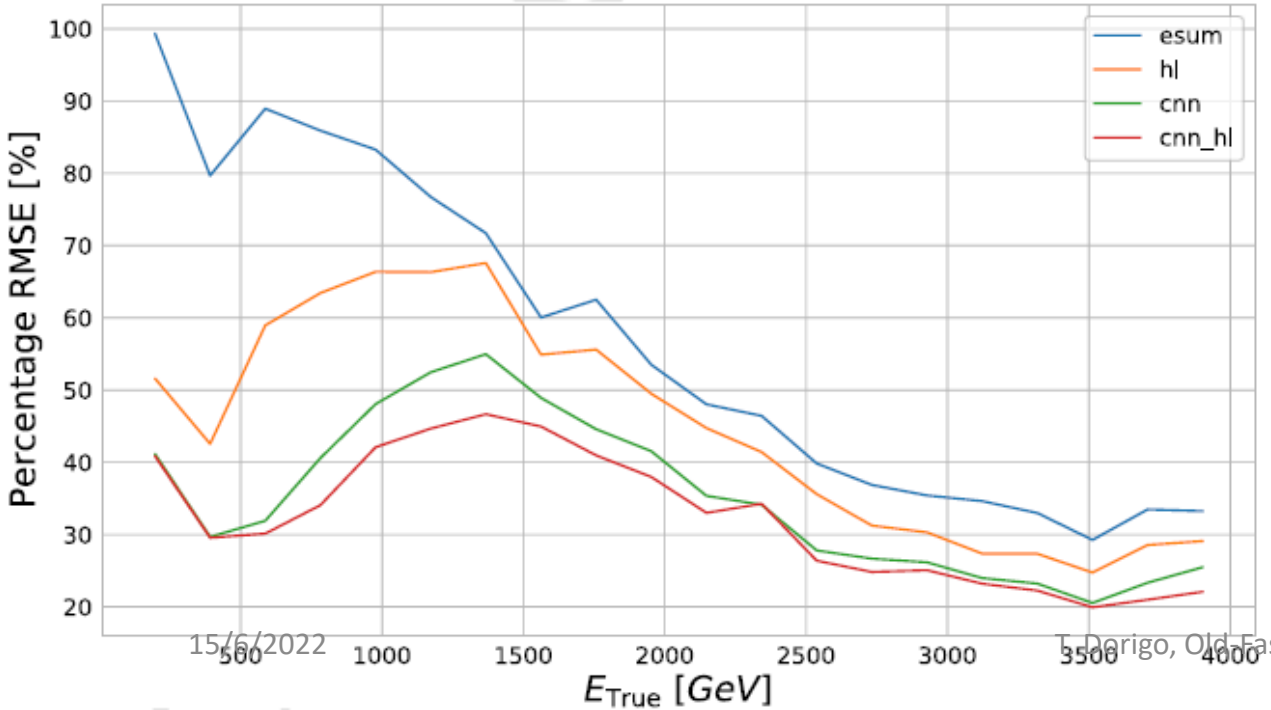
The 28 event features are the result of human-based «domain knowledge» about possible ways to aggregate the information on the spatial distribution of energy deposits (more later)

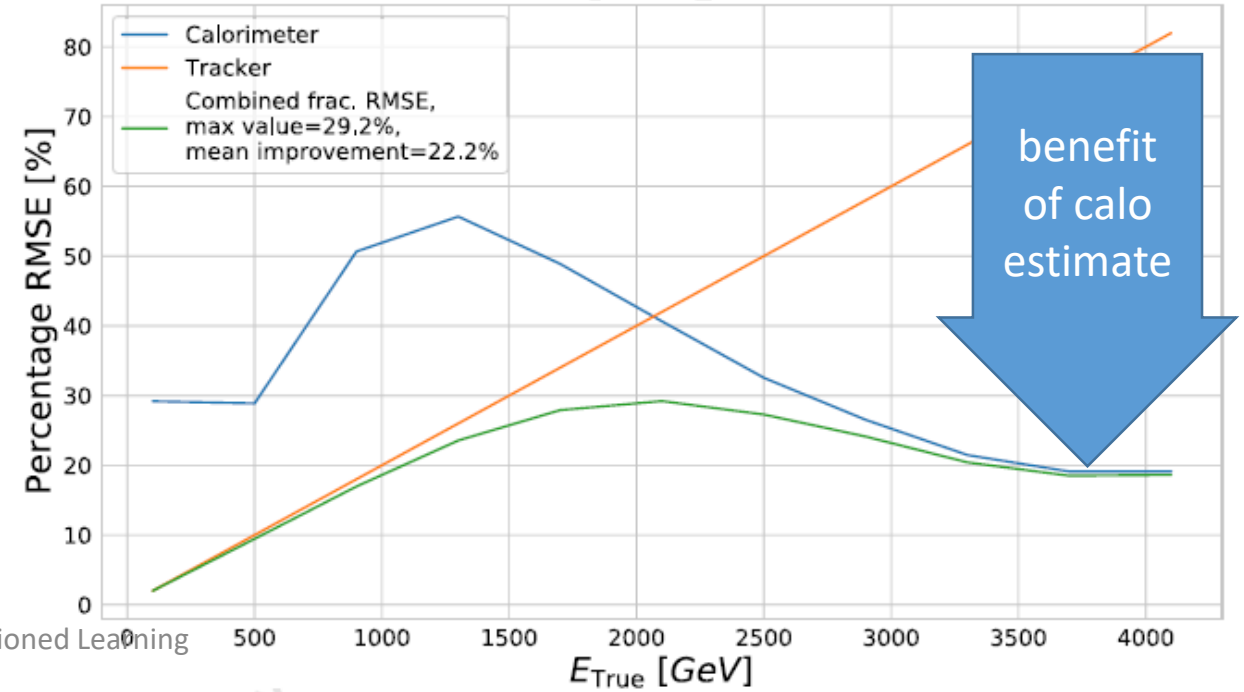


# * CNN results

The CNN model manages to recover 20% resolution for 4-TeV muons. It also demonstrates how there is information in the spatial distribution of photon deposits. A combination with a tracking measurement (δE/E=0.2E/TeV) is shown on the right.

Results are published in J. Kieseler et al., Eur. Phys. Journ. C82 (2022) 79, arXiv:2107.02119



# Why a k-NN then?

The CNN results shown were actually produced several months *after* we first tried a k-NN algorithm on the problem (in a restricted E<2 TeV range), because I had some code handy and needed to produce a preprint in time for a funding application...

The k-NN was eventually re-run on the data used for the CNN publication, after many improvements

Here I am describing the final version of the k-NN we developed.

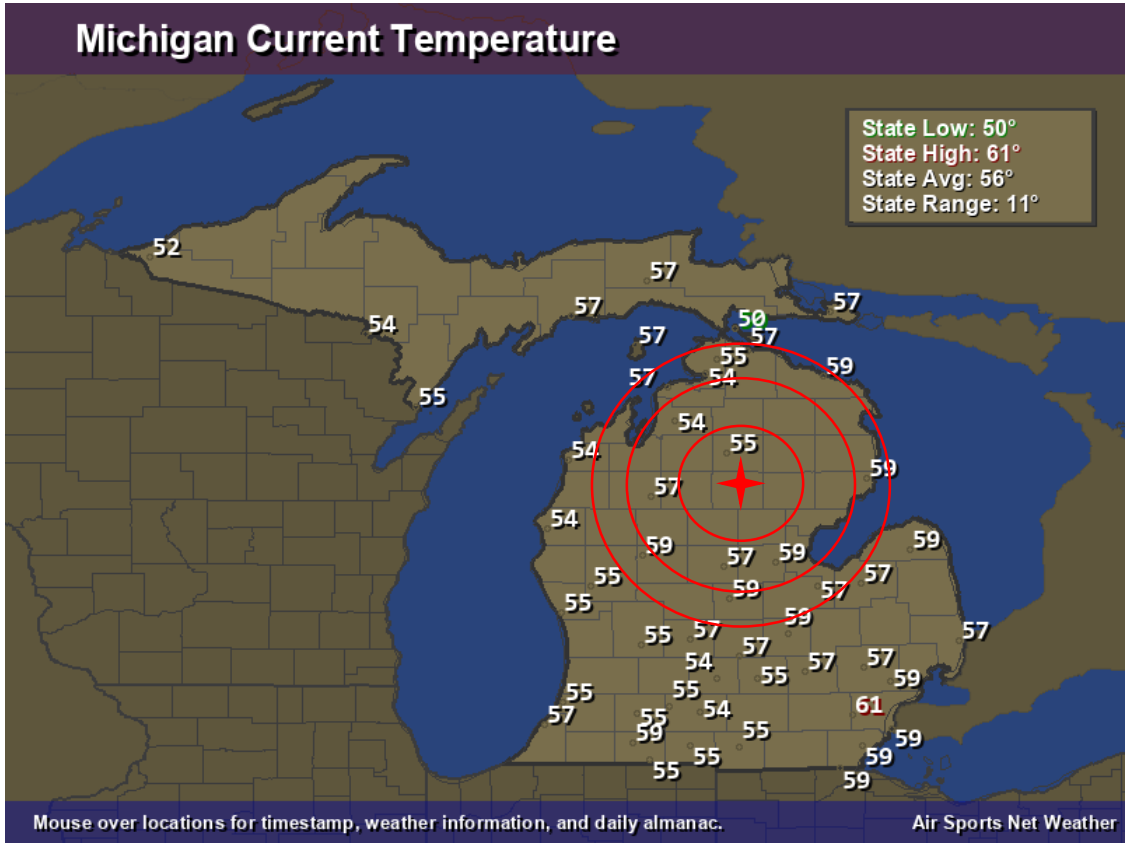
# * What is a k-NN?

It is entirely possible that you don't know what a k-NN is, so here is my one-slide explanation of a regressor k-NN.

Imagine you need to estimate the temperature in Roscommon co., Michigan (red star) based on the data on the right.

You might choose to take the state average of 56°F; that would be a low variance/high bias choice (although in the present instance you would do well).

A more principled choice is to check the temperature of neighboring counties.



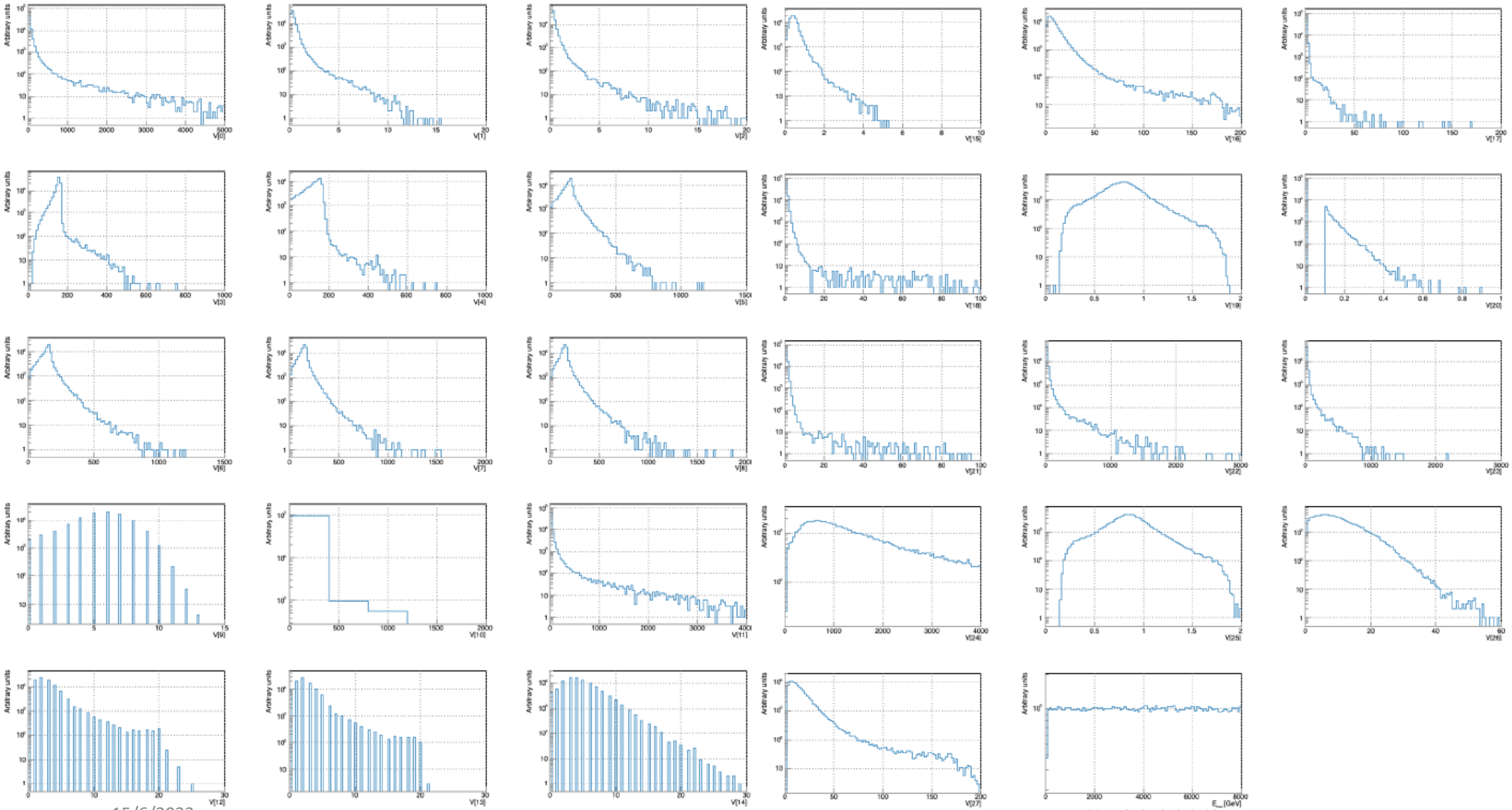

As temperature is connected to spatial distance, you may want to include/exclude data points based on their distance from Roscommon.
In k-NN, you specify k and the distance rule.

The more you restrict the range of counties which you allow in your average, the more «relevant» they become, but variance grows.

# High-Level Features

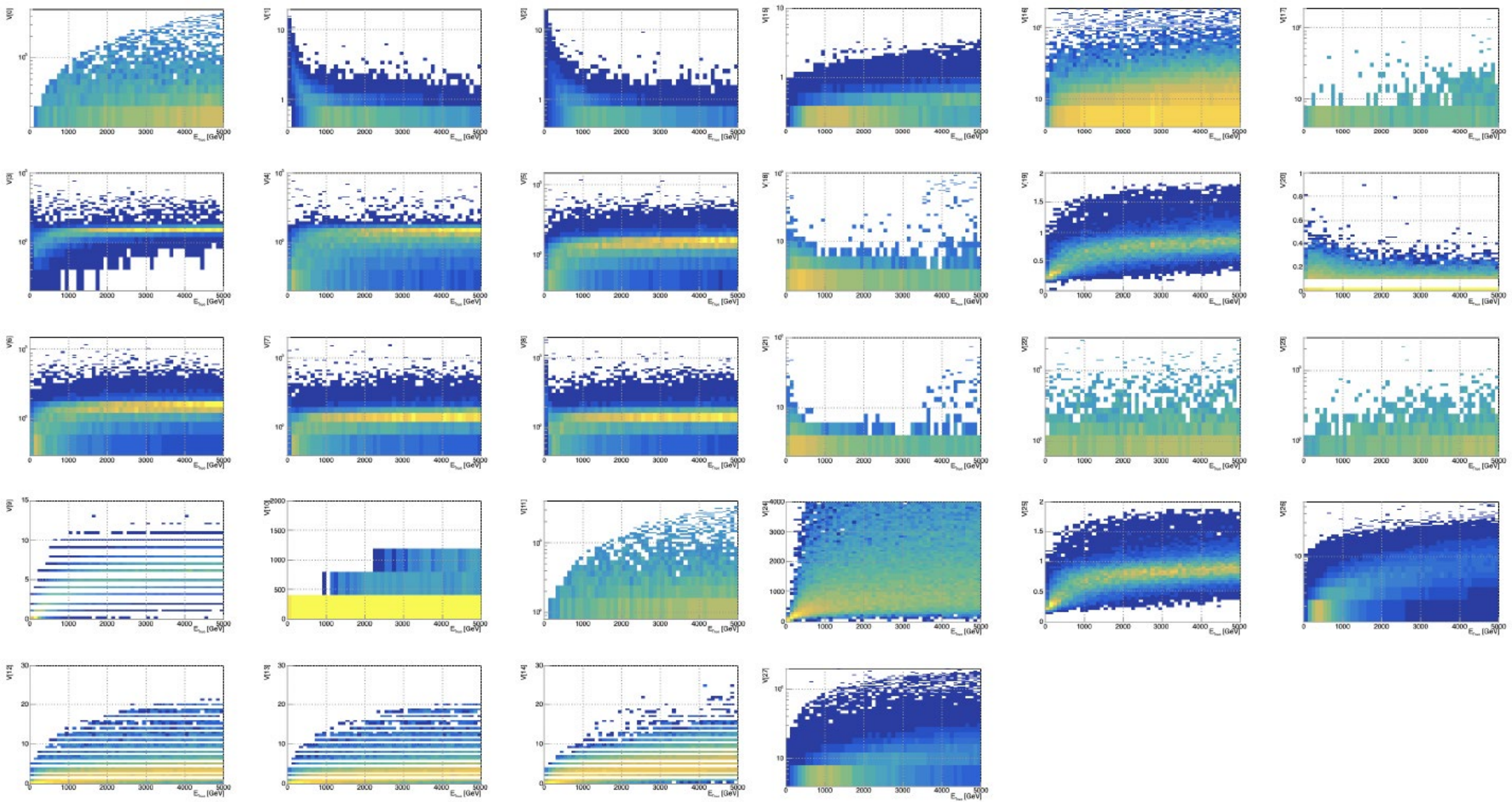
A k-NN cannot possibly make sense of a 51,200-dimensional space, so we cooked up 28 high-level features to aggregate information in various ways:

- total energy in cells for various values of min cell energy
- moments of energy distribution around track in xy space, in z slices
- number of clusters of cells (above energy threshold)
- number of towers in clusters
- energy of clusters
- imbalance in x and y of energy distribution
- fit to curvature from energy depositions (useful for E in few-100 GeV range)


Arbitrary units
V[0]
V[1]
V[2]
V[15]
V[16]
V[17]
V[3]
V[4]
V[5]
V[18]
V[19]
V[20]
V[6]
V[7]
V[8]
V[21]
V[22]
V[23]
V[9]
V[10]
V[11]
V[24]
V[25]
V[26]
V[12]
V[13]
V[14]
V[27]
E_μ [GeV]



muon energy



# kNN Construction

The kNN estimate is constructed as the average of several pools of weak learners, each pool trained independently on partially disjunct subsets of training data.

To address the curse of dimensionality, which makes even 28 dimensions too many for local averaging, we define the distance in subspaces by ignoring some of the features through an indicator function, I(d) = 1/0 if feature d is considered or not:

$$\Delta(i,j)^2 = \sum_{d=1}^{D} I(d)(x_i^{(d)} - x_j^{(d)})^2$$

Features in the definition above are standardized to have unit variance and zero mean.

The prediction for test event j can be written as

$$E(j) = \frac{\sum_{m=1}^{k} E(i_{\mathrm{kNN}}(m))}{k}$$

where $i_{kNN}(m)$ is the index of the m-th closest training event to j, according to the value of Δ(m,j).

# Pooling of weak learners

The mentioned solution of the curse of dimensionality brings a loss of information and turns the problem into one of identifying advantageous subspaces through a good choice of I(), or to combine them. To reduce information loss, we consider $N_{wl}$ weak learners, each performing a kNN average in a different subspace through different indicators $I_{wl}()$.

The regressors are then combined in a weighted average:

$$E(j) = \sum_{i=1}^{N_{wl}} W_{wl}(i) E_i(j)$$

Weights $W_{wl}(i)$ are optimized by gradient descent.

The loss of information remains, but its effect can be tamed by the added flexibility of the model, and optimized weights $W_{wl}$.

# Overparametrization

One of the aces in the sleeve of DNNs is overparametrization – it smoothens the loss function landscape and eases convergence to the real minimum. To inject overparametrization in a kNN one may only rely on training data.

Each training event affects the prediction by its position in space (fixed) and by the value of energy (also fixed). One can still inject flexibility by biasing the energy value, and altering the weight of the event in the averaging.

We introduce two sets of parameters b(wl,i), w(wl,i) for each weak learner:

$$E_{wl}(j) = \frac{\sum_{m=1}^{k} E_{wl}(i_{\mathrm{kNN},wl}(m))w(wl,m)}{\sum_{m=1}^{k} w(wl,m)} + \sum_{m=1}^{k} b(wl,m))$$

with O(600k) training events, O(10) weak learners per pool, and O(5) pools, this will boil down to about **66M free parameters** in the final model.

But can they be trained ??

# Gradient Descent

Once a loss function is defined, it is straightforward to compute derivatives and create a mechanism to navigate the parameter space in the direction of maximum negative gradient. But the kNN relies for the prediction on data which cannot be the same used to optimize the parameters.

So we split training data in a **training** and a **prediction** set. The former is used in batches to feed gradient descent and learn optimal parameters.

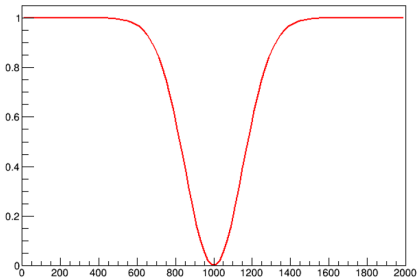
A first-guess loss for a regression task is the MSE: if T is true energy and P its prediction, we want to minimize

$$\alpha_0 \sum_{i=1}^{N_{batch}} \frac{(T_i - P_i)^2}{\sigma_0^2}$$

In our problem we have long non-Gaussian tails in many features. To reduce the effect of outliers, we write the loss as

$$L_1 = \frac{\alpha_0}{N_{batch}} \sum_{i=1}^{N_{batch}} \left[ 1 - \exp\left( \frac{(T_i - P_i)^2}{2\sigma_0^2} \right) \right]$$

and tune the sigma parameter to focus on acceptable resolution.



# Bias Penalization

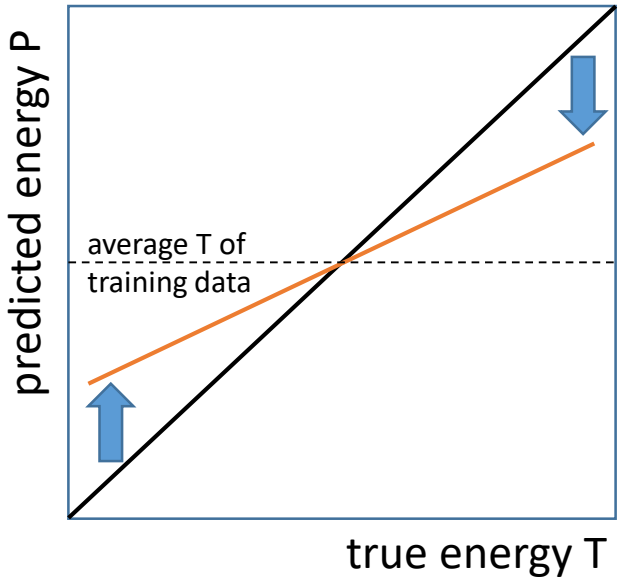
Since we average the energy of muons in a finite energy range ([0.05-8 TeV]), it is to be expected that our estimate will be biased toward the center of the range

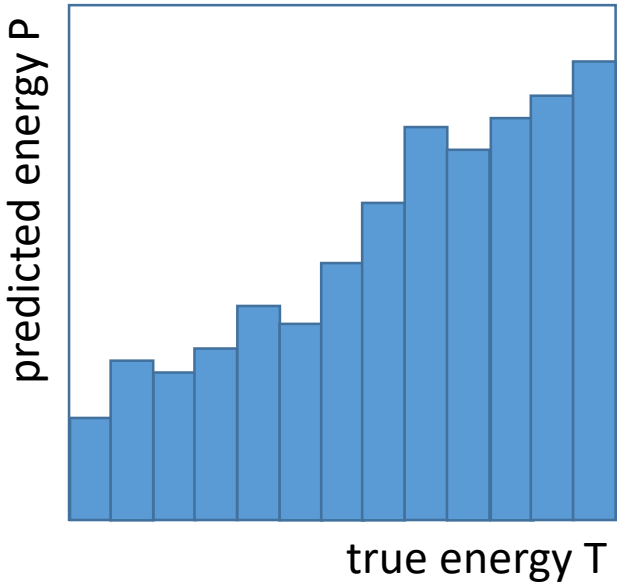
A way to reduce this effect is to penalize the loss with the following term, with m!=n:

$$L_2 = \alpha_1 \sum_{m=1}^{N_T} \sum_{n=1}^{N_T} [\hat{P}_m - \hat{P}_n - T_m + T_n]^2 / \sigma(m,n)^2$$

$N_T$ is the number of bins of tested energy; P and T denote predicted and true energy.

The sum considers all pairs of bins to enforce that predictions «line up» along the diagonal



# Bias Penalization

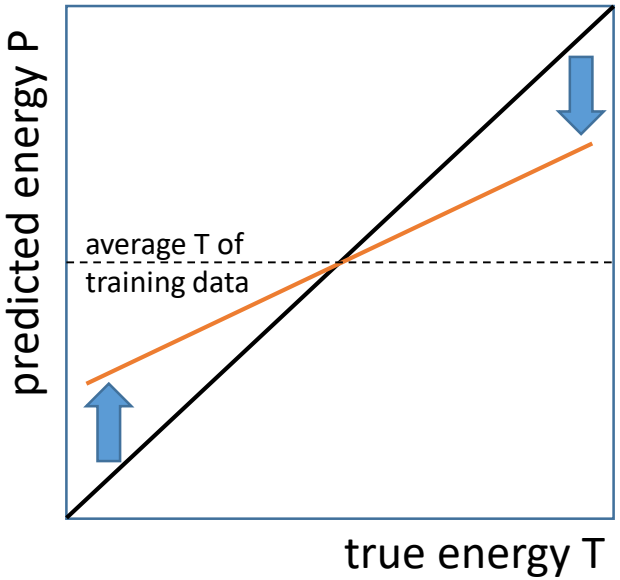
Since we average the energy of muons in a finite energy range ([0.05-8 TeV]), it is to be expected that our estimate will be biased toward the center of the range

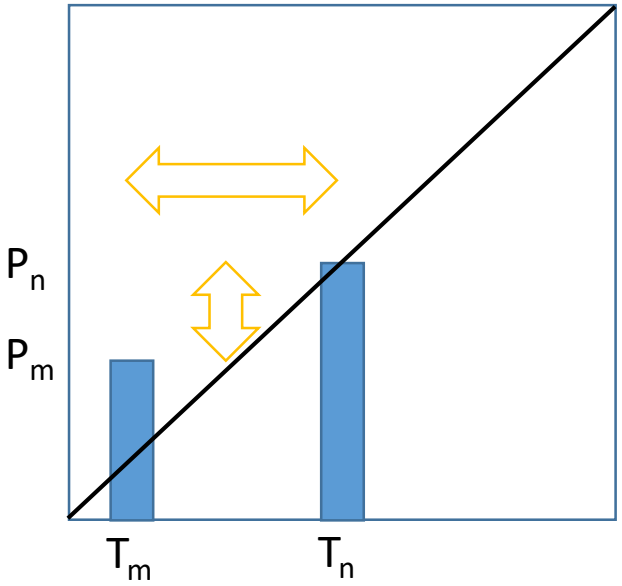
A way to reduce this effect is to penalize the loss with the following term, with m!=n:

$$L_2 = \alpha_1 \sum_{m=1}^{N_T} \sum_{n=1}^{N_T} [\hat{P}_m - \hat{P}_n - T_m + T_n]^2 / \sigma(m,n)^2$$

$N_T$ is the number of bins of tested energy; P and T denote predicted and true energy.

The sum considers all pairs of bins to enforce that predictions «line up» along the diagonal



# * The denominator in $L_2$

$$L_2 = \alpha_1 \sum_{m=1}^{N_T} \sum_{n=1}^{N_T} [\hat{P}_m - \hat{P}_n - T_m + T_n]^2 / \sigma(m,n)^2$$

The denominator is constructed by assuming that uncertainties in energy deposits scale with Poisson statistics, so that an estimate of the uncertainty in the difference at the numerator above as

$$\sigma^2_{\Delta_{mn,obs} - \Delta_{mn,exp}} \simeq \frac{T_m}{N_m} + \frac{T_n}{N_n}$$

with $N_m$ the number of events in bins m. This gives equal importance to deviations over a small energy difference $T_m$-$T_n$ and over a large one. A way to give more weight to larger differences is to add that term to the denominator. Our choice then is

$$\sigma(m,n)^2 = \frac{T_m N_n + T_n N_m}{N_m N_n (T_m - T_n)}$$

# Other figures of merit

The loss as defined above is not necessarily the best proxy to the success of our regression task, as we are mainly concerned with the high-end part of the spectrum, where a curvature measurement would be inadequate.

In addition, we assume we will combine the calorimetric measurement with a tracking one with a resolution δE/E = 0.2E/TeV.

The combination yields

$$RMS_{comb}(E) = \sqrt{\frac{RMS_{tr}(E)^2 RMS_{cal}(E)^2}{RMS_{tr}(E)^2 + RMS_{cal}(E)^2}}$$

and we define as a figure of merit the function $MaxRes = \max_{E=0.05-4\ \mathrm{TeV}} RMS_{comb}(E)$

Further, we gauge the discrimination power of predicted energies for 4 vs 2 TeV muons and 3 vs 1 TeV muons by computing the following proxies:

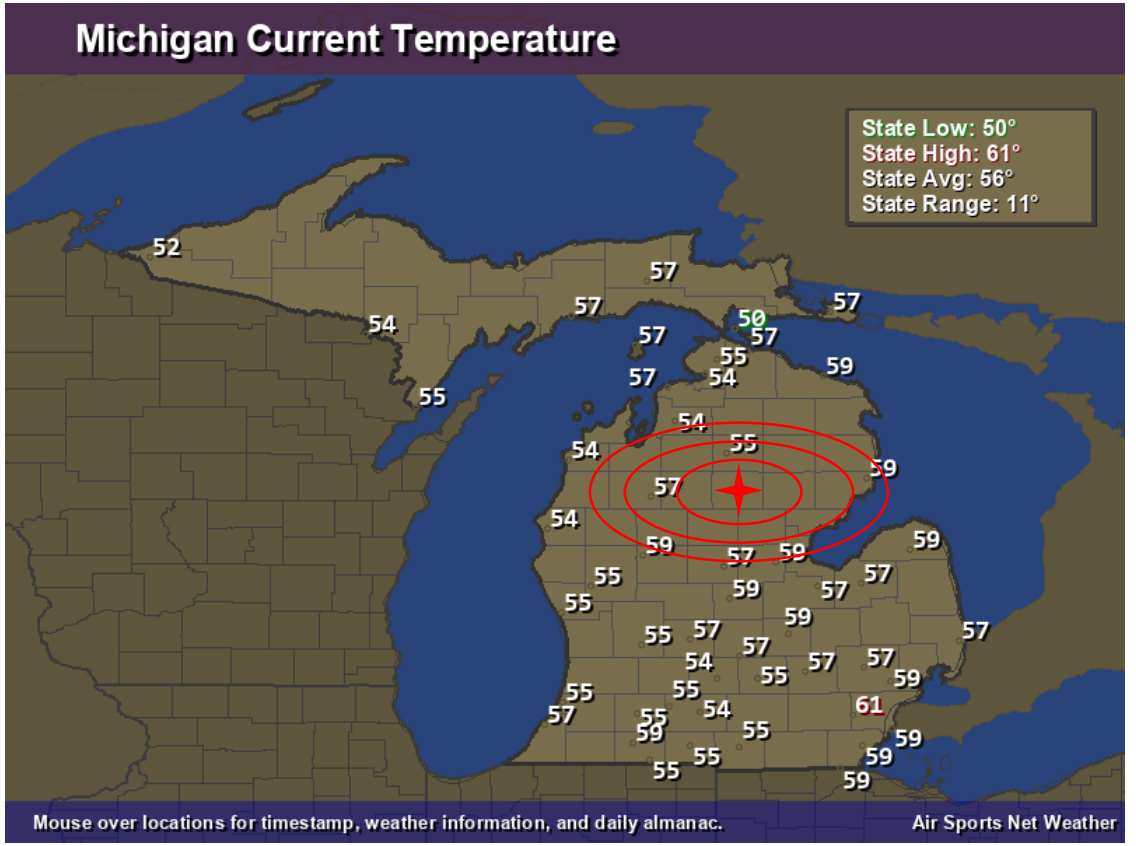
$$Discr_{24} = \frac{\hat{E}_p(4) - \hat{E}_p(2)}{\sqrt{\sigma_{E_p}(4)^2 + \sigma_{E_p}(2)^2}}$$

$$Discr_{13} = \frac{\hat{E}_p(3) - \hat{E}_p(1)}{\sqrt{\sigma_{E_p}(3)^2 + \sigma_{E_p}(1)^2}}$$

# *Not all features are equal

In the example of temperature averaging we made earlier, we could argue that climate models indicate that longitude is less important than latitude to impact ground temperature

We might use that information to get a less biased estimate, if we modified the distance calculation...




Above, the «balls» containing evaluation points are shrunk along the direction that is more relevant for the estimation.

However, note that a linear dependence of average temperature with latitude would not suffice for this to be a justified procedure, as a linear trend gets **averaged out** !

# * Hyperball optimization

An additional feature of the developed algorithm is an adaptive shape of the k-balls, by giving weights to the distance components proportionally to the variance exhibited, in the proximity of the test point, by the quantity to be estimated.

An estimate of the bias due to the variability of E as a function of x in the surrounding of the test point (x=0 here) is provided by fitting with a parabola the deviations from the mean.

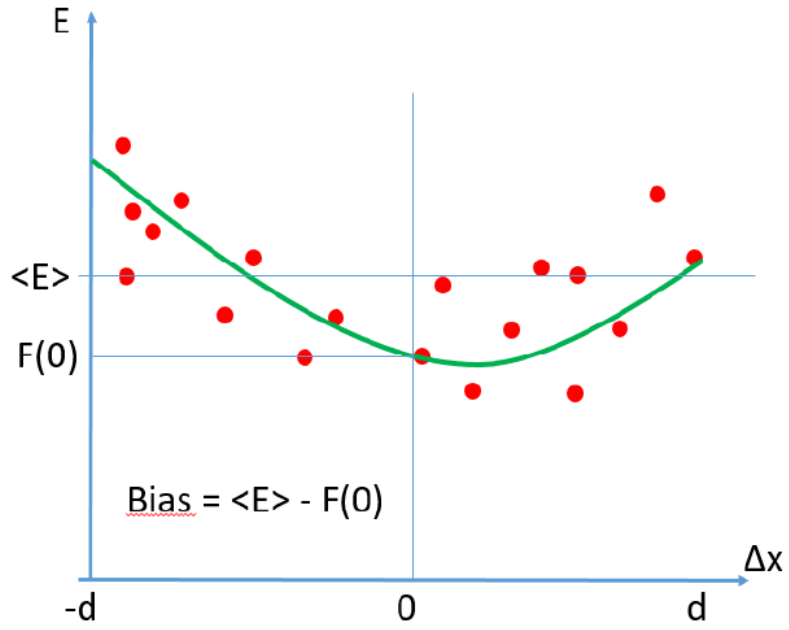
This corresponds to simply getting the average energy in a wide region [-d,d] and in a central region $E_0$=<E>([-d/3,d/3]), and computing the bias as

$$b = \langle E \rangle - E_0.$$

The biases can then be used to give more or less importance to the different features in the calculation of the distance:

$$d = \sum_{i=1}^{N_D} \frac{1}{b_i/b_{max} + \epsilon}(x_{i,j} - y_{i,j})^2$$

The calculation requires to study the vicinity of the test point with large k (3-5 times the normal value) and is thus very CPU expensive.



# Features pruning

Regardless of our dimensionality reduction based on feature subsampling, it is useful to identify and get rid of variables that are not useful for the regression task.

The identification is difficult, as the feature space is complex and the interdependencies non trivial.

We run 1639 independent regression tasks using randomly generated single weak learners on the same training data sample (50,000 events), with subspace dimension variable between 8 and 15, and compute the average value of figures of merit as a function of their inclusion or exclusion of each feature.

# * Pruning study results

| Variable | Δ Loss | Δ Max res. ($\times 10^4$) | Δ 2-4 TeV discr. | Δ 1-3 TeV discr. |
|---|---|---|---|---|
| 4 | $0.2939 \pm 0.0336$ | $\mathbf{-2.60 \pm 1.80}$ | $-0.00105 \pm 0.00138$ | $-0.01427 \pm 0.00200$ |
| 5 | $0.2531 \pm 0.0352$ | $6.91 \pm 1.76$ | $-0.00432 \pm 0.00123$ | $-0.01120 \pm 0.00219$ |
| 6 | $0.2299 \pm 0.0377$ | $11.91 \pm 1.73$ | $-0.00505 \pm 0.00125$ | $-0.00803 \pm 0.00222$ |
| 7 | $0.2034 \pm 0.0366$ | $17.80 \pm 1.77$ | $-0.00770 \pm 0.00126$ | $-0.01002 \pm 0.00219$ |
| 3 | $0.1646 \pm 0.0332$ | $4.40 \pm 1.82$ | $-0.00079 \pm 0.00140$ | $-0.00357 \pm 0.00207$ |
| 25 | $0.1575 \pm 0.0363$ | $0.67 \pm 1.86$ | $-0.00506 \pm 0.00139$ | $-0.00585 \pm 0.00215$ |
| 8 | $\mathbf{-0.0006 \pm 0.0365}$ | $3.40 \pm 1.81$ | $\mathbf{0.00303 \pm 0.00145}$ | $\mathbf{0.01068 \pm 0.00221}$ |
| 11 | $0.0910 \pm 0.0341$ | $2.93 \pm 1.74$ | $-0.00366 \pm 0.00116$ | $-0.00650 \pm 0.00197$ |
| 18 | $0.0898 \pm 0.0346$ | $\mathbf{-1.02 \pm 1.69}$ | $-0.00422 \pm 0.00116$ | $-0.00633 \pm 0.00198$ |
| 10 | $0.0985 \pm 0.0331$ | $2.61 \pm 1.73$ | $-0.00368 \pm 0.00118$ | $-0.00507 \pm 0.00193$ |
| 20 | $0.0997 \pm 0.0359$ | $\mathbf{-2.91 \pm 1.764}$ | $-0.00309 \pm 0.00116$ | $-0.01228 \pm 0.00203$ |
| | (larger is worse) | (larger is worse) | (smaller is worse) | (smaller is worse) |

Worst ΔLoss

Worst ΔMaxR

Worst D24

Worst D13

Listed here are variables that are among the six worst on any one of the four figures of merit.

The variables which do **not** worsen the performance of the corresponding figure of merit are highlighted in boldface.

We finally reject all listed variables except variable 8, which is only present in the list due to the (relatively noisy) max res.

# * Validation of pruning procedure

We validate the pruning procedure by running a regression on all 28 variables, and then removing gradually the worst features.

| Excluded variables | Loss | Max res. | 2-4 TeV discr. | 1-3 TeV discr. |
|---|---|---|---|---|
| None | 38.4141 | 0.330749 | 0.706453 | 1.12254 |
| 3, 4, 5, 6 | 37.5701 | 0.32778 | 0.709819 | 1.15602 |
| 3, 4, 5, 6, 7, 25 | 37.2649 | 0.327508 | 0.715047 | 1.1493 |
| 3, 4, 5, 6, 7, 25, 10, 18, 20 | 37.3583 | 0.325874 | 0.718947 | 1.15443 |

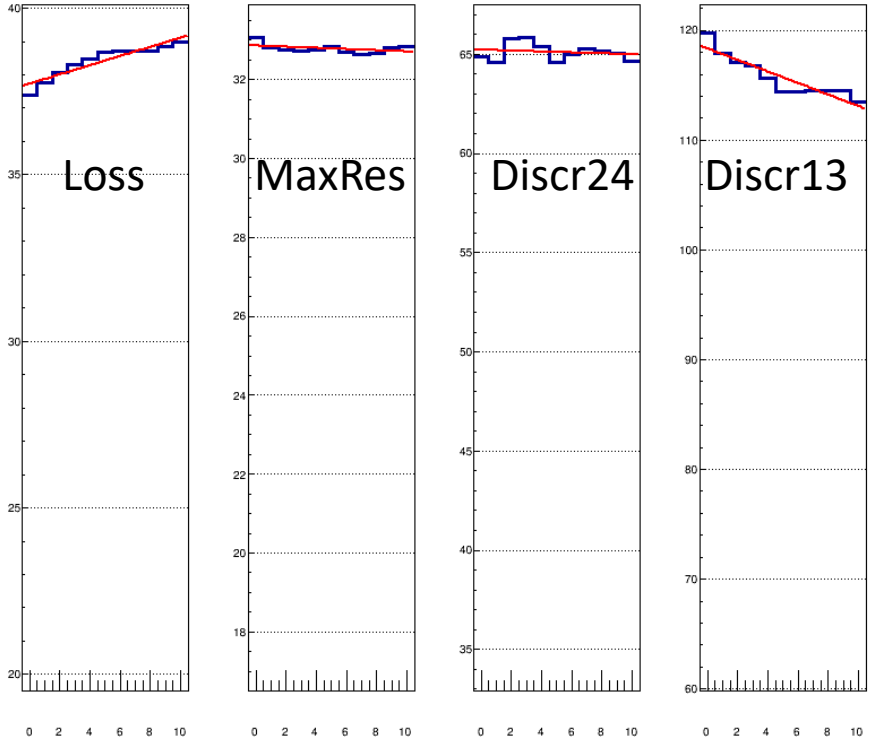
As expected, we observe a decrease of the loss and MaxRes, and an increase of the discrimination power

# * Validation of pruning/2

Further evidence of the validity of the choice comes from starting off from the best 5 features and adding the worst ones one by one. We see a clear increase of the loss and a decrease of Discr13; the other two FoMs remain basically untouched

| | (lower is better) | | (higher is better) | |
|---|---|---|---|---|
| Added variables | Loss | Max res. | 2-4 TeV discr. | 1-3 TeV discr. |
| 0 | 37.40 | 0.3306 | 0.6486 | 1.1972 |
| 1 | 37.77 | 0.3282 | 0.6458 | 1.1787 |
| 2 | 38.06 | 0.3276 | 0.6580 | 1.1714 |
| 3 | 38.31 | 0.3272 | 0.6583 | 1.1674 |
| 4 | 38.46 | 0.3276 | 0.6536 | 1.1567 |
| 5 | 38.69 | 0.3283 | 0.6459 | 1.1438 |
| 6 | 38.73 | 0.3270 | 0.6498 | 1.1444 |
| 7 | 38.70 | 0.3264 | 0.6526 | 1.1451 |
| 8 | 38.70 | 0.3266 | 0.6518 | 1.1451 |
| 9 | 38.85 | 0.3280 | 0.6503 | 1.1449 |
| 10 | 38.97 | 0.3285 | 0.6465 | 1.1384 |

Minimum value of graphs is 50% of maximum



# * Regressor details

- k is set to 100. This was not «properly» optimized but the regression performance was checked for a few values in [10,500]. CPU limitations make too large values impractical; smaller values start to make the loss very noisy (especially L2)
  - large k has benefit of giving more flexibility to predictions ($200*N_{wl}+N_{wl}$ parameters per test point)
- 300,000 / 400,000 events used for prediction
- 400,000 events are used for training (batch GD)
- 100,000 events are used for testing
- 18 variables used out of 28 (10 excluded by pruning)

# * Weak learners choice

We used 32 sets of weak learners: 8 sets of 5 learners, 8 sets of 10 learners, 8 sets of 15 learners, and 8 sets of 20 learners.

Each set was separately determined by optimization searches running on cross-validation sets of training data, without an optimization of w() and b() parameters. The procedure was as follows:

1. Select by bootstrapping 10,000 training events
2. Define at random $N_{wl}$ weak learners, by selecting a variable fraction (from 30% to 80%) of the active features among the 18;
3. Iterate to modify active flags, by flipping some of the flags on or off, then performing the regression on 5000-event batches; compare loss, minimize it

| Flags of weak learner in pool (0:27) | Individual loss of WL |
|---|---|
| 1 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 0 0 0 1 1 0 1 0 | 37.22 |
| 0 1 1 0 0 0 0 0 1 1 0 0 0 1 1 1 0 0 0 0 0 1 0 0 1 0 1 1 | 37.37 |
| 1 1 1 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 1 0 0 0 0 1 1 | 37.34 |
| 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1 | 37.60 |
| 1 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 | 37.53 |
| Global loss for this batch | 37.02 |

*Table 5: Active subspace dimensions for five weak learners in a 5-WL set.*

The resulting sets of weak learners perform significantly better than the original random sets. Typical decreases of loss by 2% to 5% observed.

Other recipes, based on genetic breeding, did not produce good results.

# * Other hyperparameters

- Alpha parameters (loss multipliers) must be defined for the two loss components L1, L2. This is tricky and requires some fine-tuning
- The learning rate must be set to a value that allows smooth descent. Also tricky
- Learning rate is also updated during gradient descent via additional scheduling parameters, depending on the steepness of the descent
- $N_{batch}$ is the number of training events used to evaluate the loss during learning – must be large enough for L2 to be meaningful (40 bins, want O(100 GeV) uncertainty on predictions per bin with RMS of O(1 TeV) → 100 events per bin → 5000 events per batch)

# Weights and biases initialization and learning

W() and b() need to be initialized. b() is set to zero, W() to a value that is =1.0 below 5 TeV (4 TeV is the upper limit of the regression range), and smoothly decays to 0 for E=8 TeV with a sigmoid (same approach was used by CNN paper).

During training, the weights and biases get tweaked by the learning process. For W() you still see some remainder of the initial trend, but with large spread

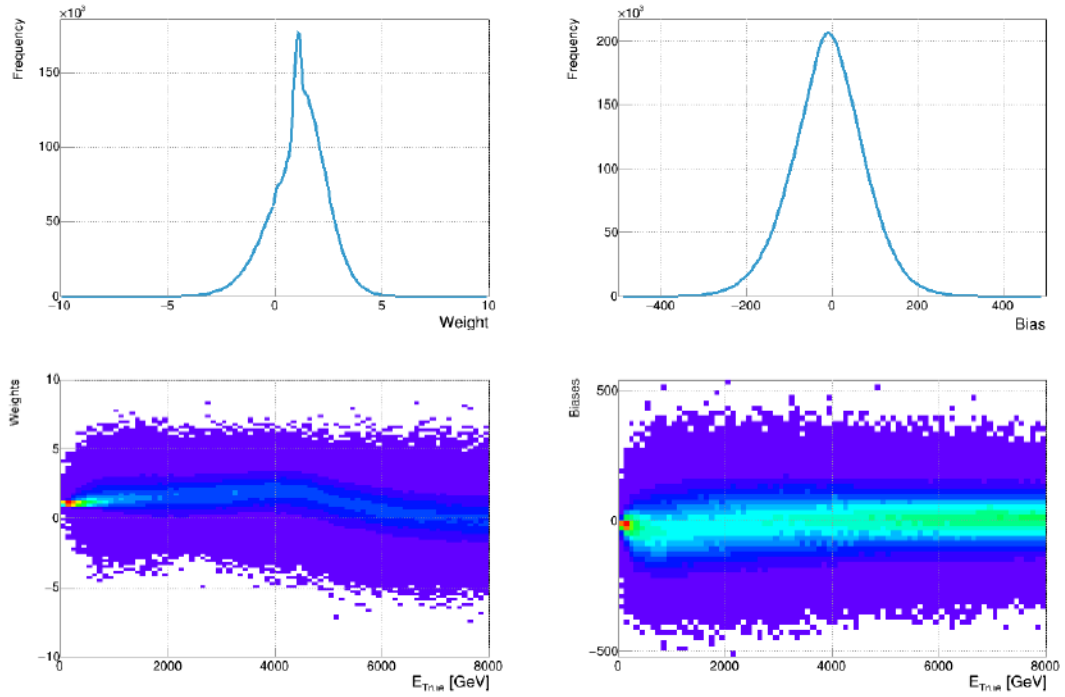



*Figure 7: Example of the distribution of weights obtained by an optimization run with 400,000 training events and 5 learners. Top left: distribution of event weights; top right: distribution of event biases (in GeV). Bottom left: weights versus true muon energy; bottom right: biases (in GeV) versus true muon energy.*

# Final run

For each of the four sets of 8 regressors with $N_{wl}$=5,10,15,20, two are chosen based on the FoMs already discussed, and using cross validation. Finally, we run the eight sets on 80,000 test events and 400,000 or 300,000 training events.

The total number of parameters used in the final regression is

$N_{weights}$ = (2*5 + 2*10)*400,000 + (2*15+2*20)*300,000 and the same number of biases, plus 100 global weak learner weights. The total is thus (12M+21M)*2 +100 = 66,000,100 parameters.

Each job needs about one week to complete the training.

The final loss from the averaging of the 8 regressors is 33.734, MaxRes is 0.3372, Discr24 is 1.3318, and Discr13 is 0.7025. These numbers are better than those of the 8 inputs.

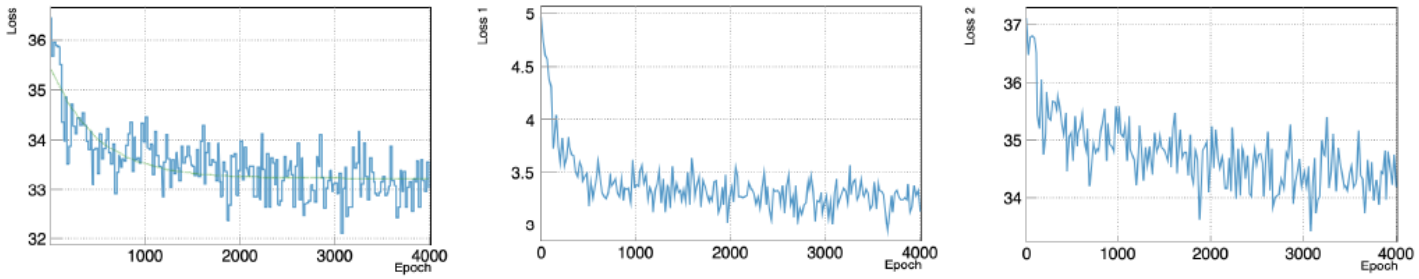



*Figure 8: Evolution of the loss function and its components with the gradient descent iterations (epochs). Left: total loss function (L); center: non-linearity penalization term ($L_2$), right: modified MSE term ($L_1$). The two terms on the center and right do not add up to the one on the left because of a rescaling factor.*

# Results

The regression shows an almost linear response, indicating that the bias correction penalization helped. The MSE reaches down to 22%E at 4 TeV.

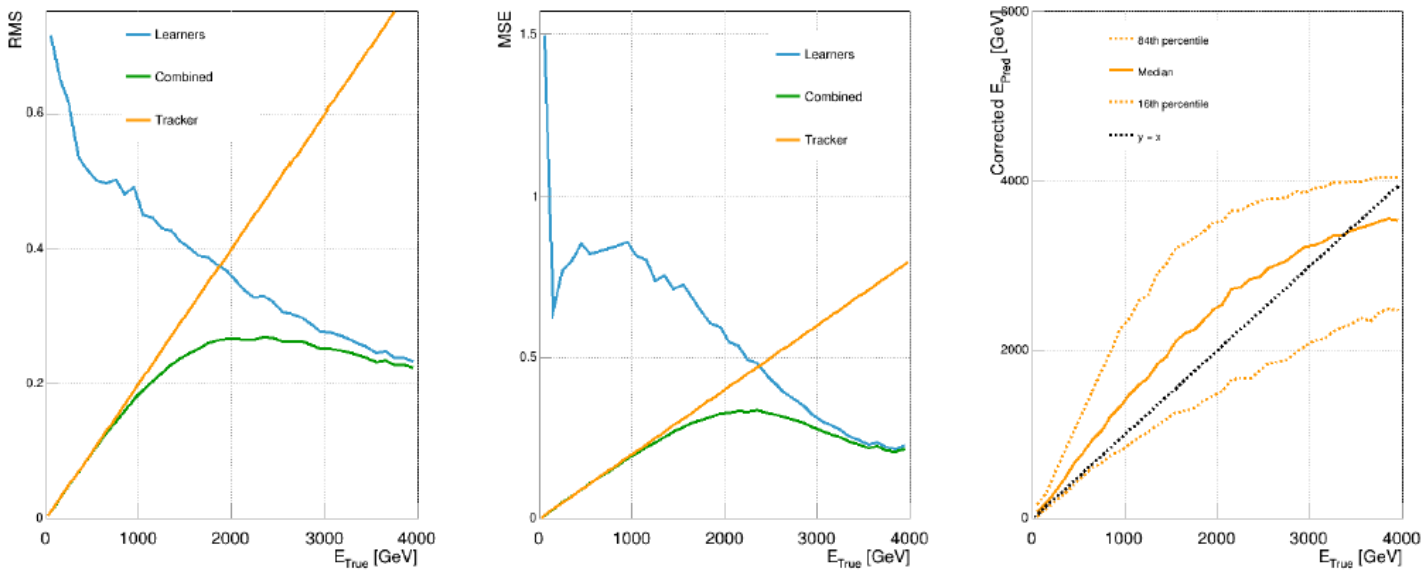
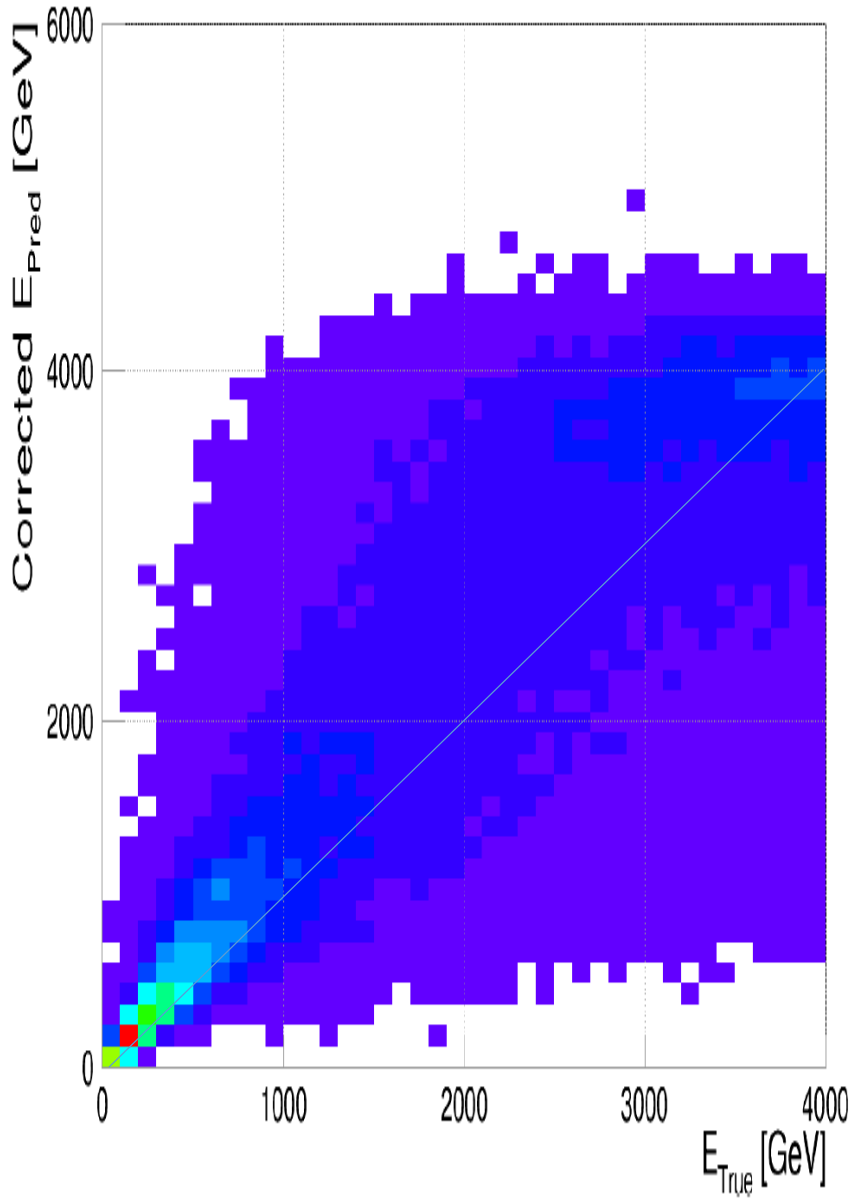



*Figure 11: Left: RMS resulting from learners-only, tracker-only and combined predictions, against true muon energy; center: MSE resulting from learners-only, tracker-only and combined predictions, against true muon energy; right: median, $84^{th}$, and $16^{th}$ percentiles of corrected predicted energy against true muon energy.*



# Results/2

For a comparison, we look at results of a NN (orange), a default kNN (pink), and Xboost. The results of our k-NN outperform the standard k-NN and are overall similar to those of the ML methods, and are slightly better at high E.

But the CPU and analysis load is non comparable!

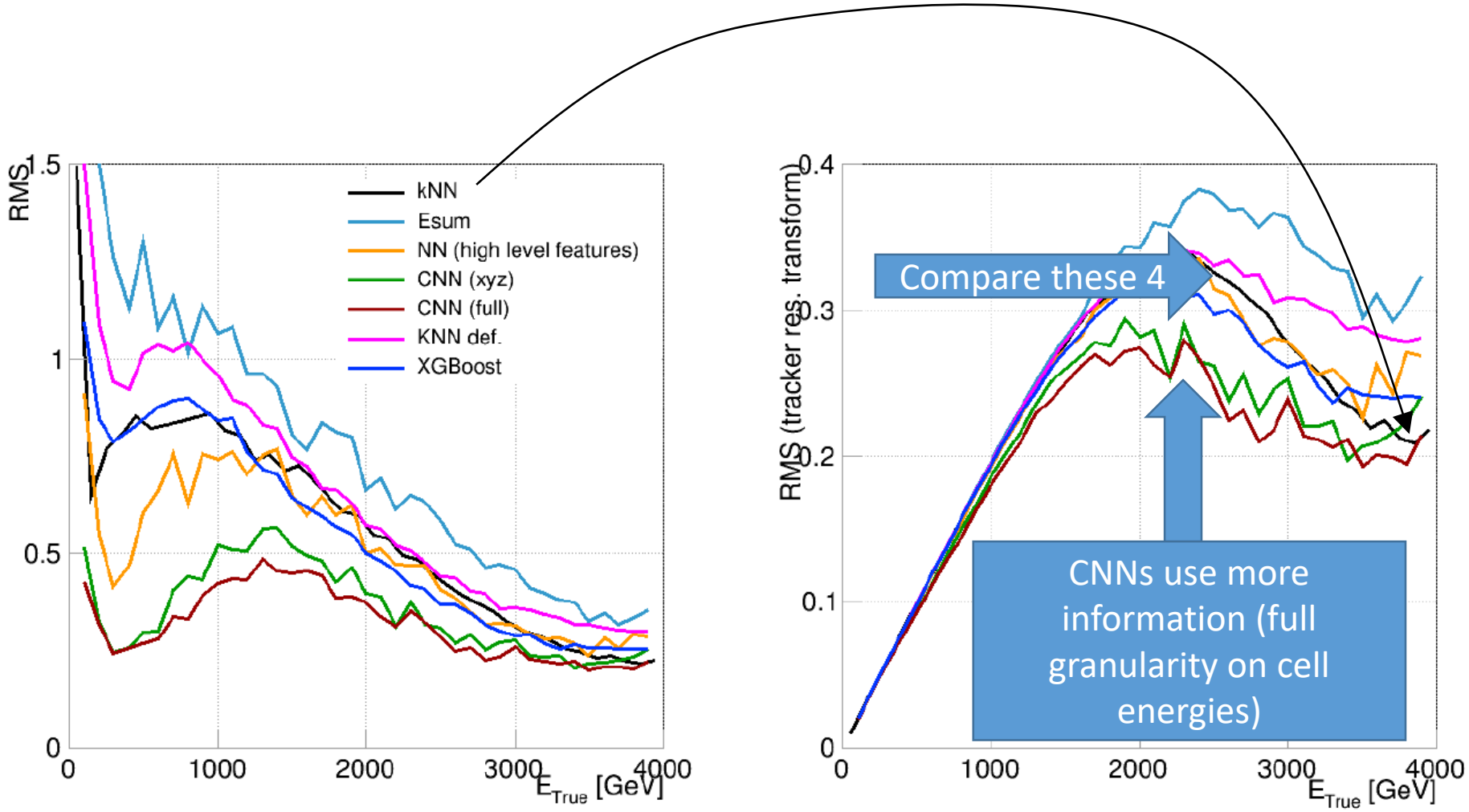



*Figure 13: Left: comparison of the mean squared error of predictions of different algorithms employing the same training and test data. Right: comparisons of the mean squared error of the combinations of tracker and calorimeter regressed predictions. Black: deep regression kNN (described in this article); light blue: energy sum model; orange: neural network with high level features; green: CNN with spatial features; red: full CNN; magenta: classical kNN; blue: XGBoost.*

# Part 2: Anomaly detection in the copula space

In general, we are interested in identifying localized areas of phase space with anomalous accumulation of events in collider data.

If we consider the problem in a fully unsupervised way, however, we must realize that «anomalous» is not a well defined concept

We can concern ourselves with tasking a unsupervised algorithm with identification of overdensities in a multi-dimensional space. This requires some pre-processing steps, to make it easier to the machine.

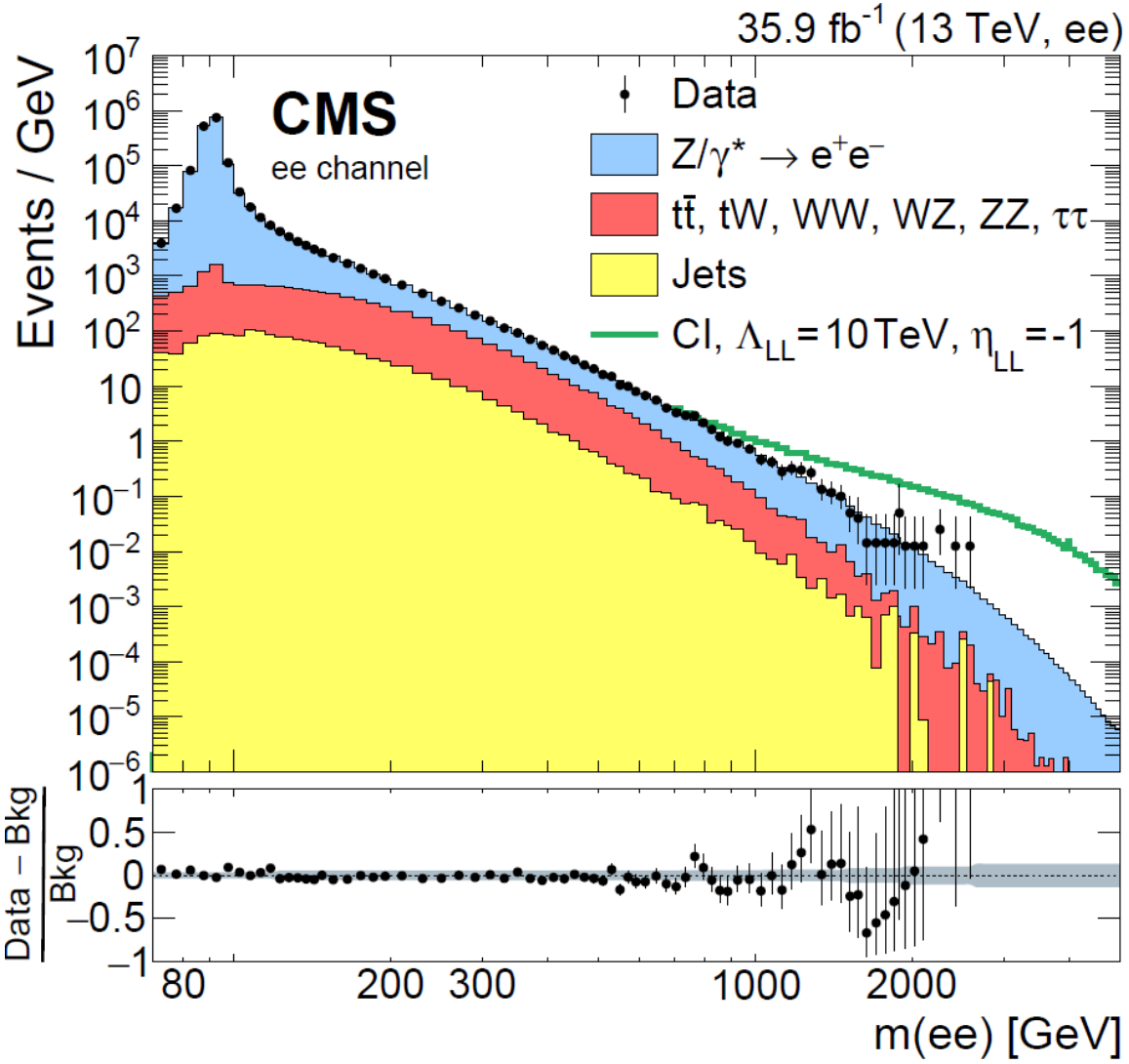
# Uneven marginals

Anomaly detection, like many other ML tasks, benefits from defining a metric in feature space

What we are looking for are overdensities in a complicated, high-D space, whose directions are populated in a very disuniform and sparsified way by discrete MC examples (a.k.a. events)

- e.g. think at $p_T$ distributions, or invariant masses: an exponentially falling behaviour is commonplace

If you ask a unsupervised algorithm to locate an overdensity in a space spanned by similar variables, it will point at low $p_T$, low M

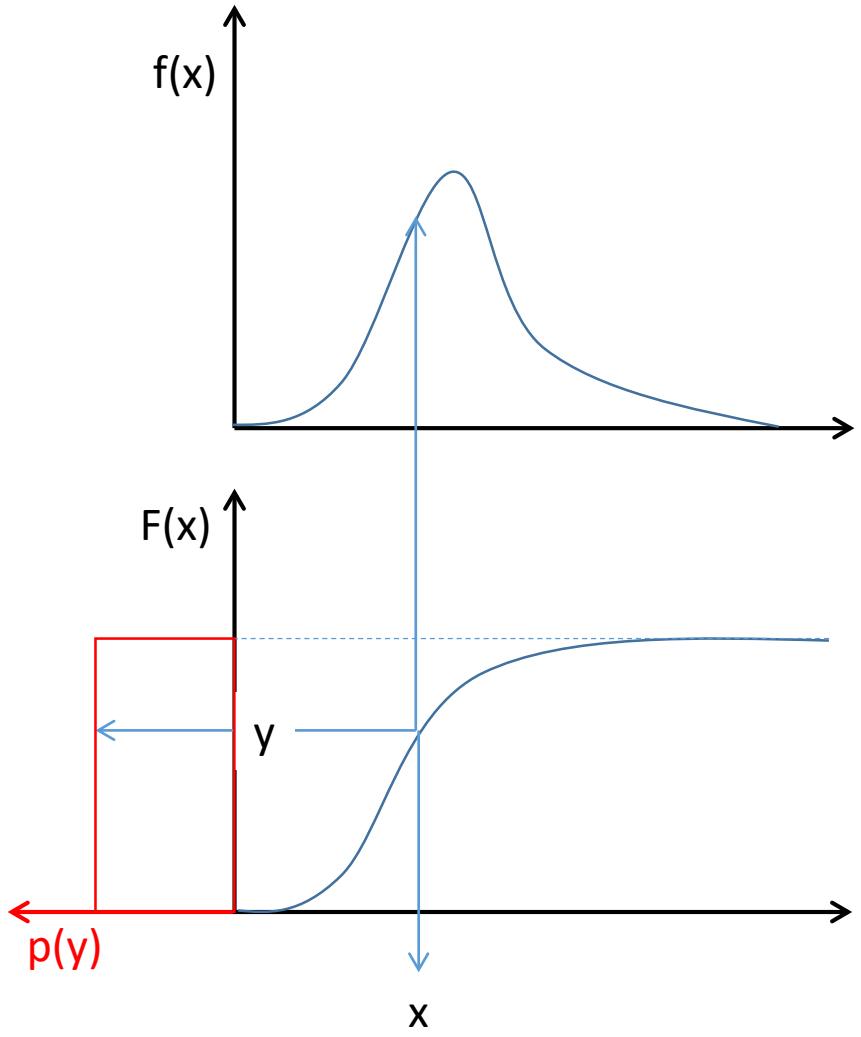


# The probability integral transform

Given a normalized f(x), compute the cumulative distribution function of f:

$$\mathrm{F(x)} = \int_{-\infty}^{x} f(t)dt$$

Now y=F(x) is uniformly distributed in [0,1]. There is a 1-to-1 map connecting x and y, so *the transformation is invertible*.

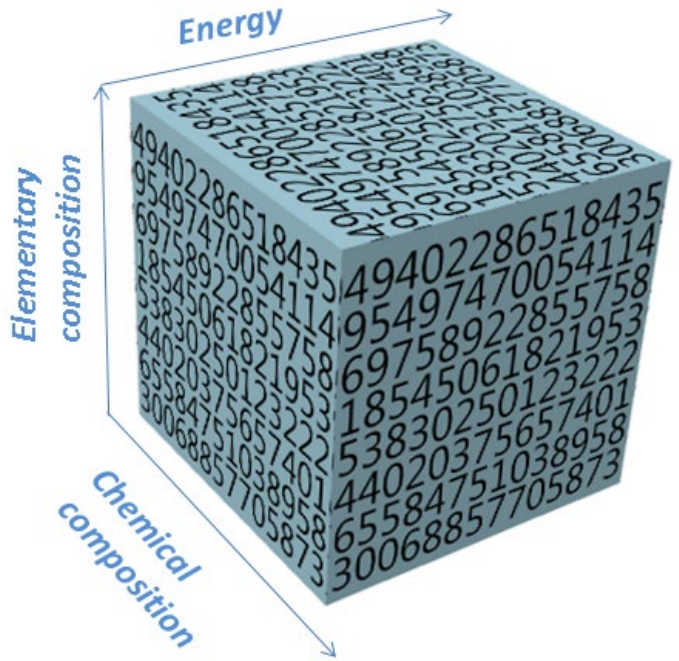


# The Copula

In Statistics a useful concept is that of the **copula**: from Sklar's theorem, any multivariate joint distribution is decomposable in univariate, uniform marginals and a **uniquely defined copula** which describes the dependence of the features

If you apply a **probability integral transform** to each coordinate, such that each feature has a flat marginal, all the information on data structure rushes to hide in the interconnection of the features

- As now each feature fits within a [0,1] interval, one gets a D-dimensional cube
- Watched from each side, the cube has flat marginals – all the information is still there, conveniently packed in the correlation of the features



# Advantages

Once your data fit in a cube, they are easy to represent - No more hassle with dimensionality of the features (you divided it out); discreteness looks less of an issue; boundaries are identical

The main advantage of standardizing data as described, when one searches for an overdensity, is that the very idea of overdensity emerges more clearly

- In the "bump in an invariant mass distribution" idealized picture, what you have is a featureless background which falls exponentially. Hidden within, there is a signal which can only be put in evidence if many other variables are considered. Background usually peaks at low M or $p_T$, and that is the most striking overdensity

- But if you take that away, by flattening the M and pT marginals with integral transforms, you may become sensitive to a **localized** overdensity present in a D'<D subspace of relevant features defining the signal phase space

# High correlations → trouble

In the typical feature space of data passing some trigger, or even preselected with basic cuts ("*two photons plus anything*", "*four or more jets*", etc.) one finds **huge correlations** between informative features

- the basic reason is that the momenta of observed objects are related by physical conservation laws: **matrix element + selection biases** do it

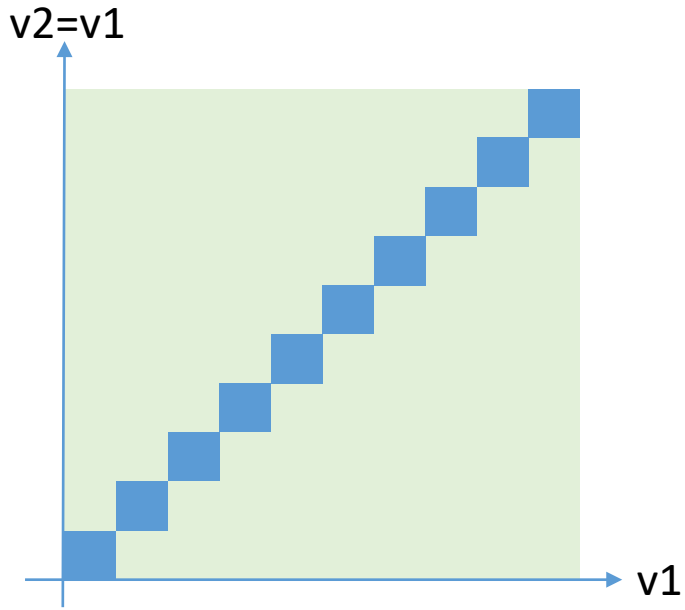
How can we even hope to get sensitive to those extra hidden correlations between some of the features (and **we do not know which**) that a signal may have added to the mix?

→We may **rotate the data such that we get uncorrelated features**, by doing a Principal Component Analysis (PCA)

→Alternatively, we may search for correlated variables and remove them (see later)

Imagine you search in a space with D>10 where a variable is repeated 10 times ($\rho$=1).Any $[x,x+0.1]^{10}$ interval on those features will have a volume of $10^{-10}$, but will contain 10% of the data! → $10^9$ underestimate...

Below, a 2D example. A $0.1^2$ box will contain 10% of the data but have a volume of 0.01



# * Principal Component Analysis

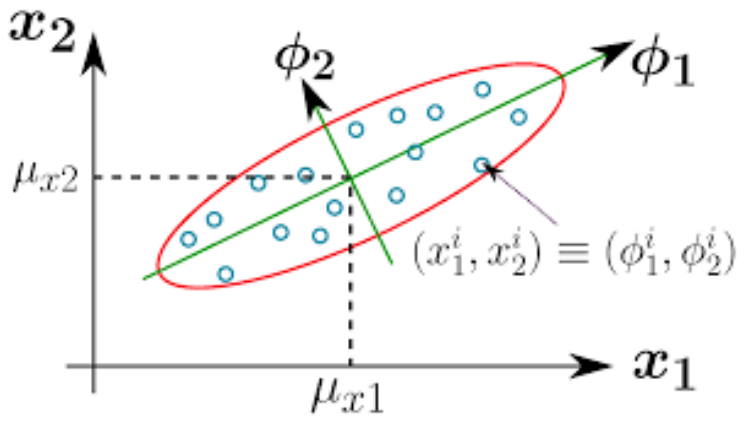
PCA applies an orthogonal transformation to a D-dimensional space, obtaining a new D'<=D –dimensional space where each of the new features (aptly called "principal components") are **linearly uncorrelated**

- The first PC is the one that accounts for the maximum amount of variance in the data; each successive PC accounts for most of the residual variance, while living in a subspace orthogonal to the previous one(s)

What PCA does is a hyperellipsoid fit to the data.
The ellipsoid is narrow along uninformative directions

One may thus only consider the first D'<D principal components, killing two birds with one stone:

1) dimensionality reduction
2) the new variables are uncorrelated, so they simplify the problem of finding pockets of overdensity in the copula



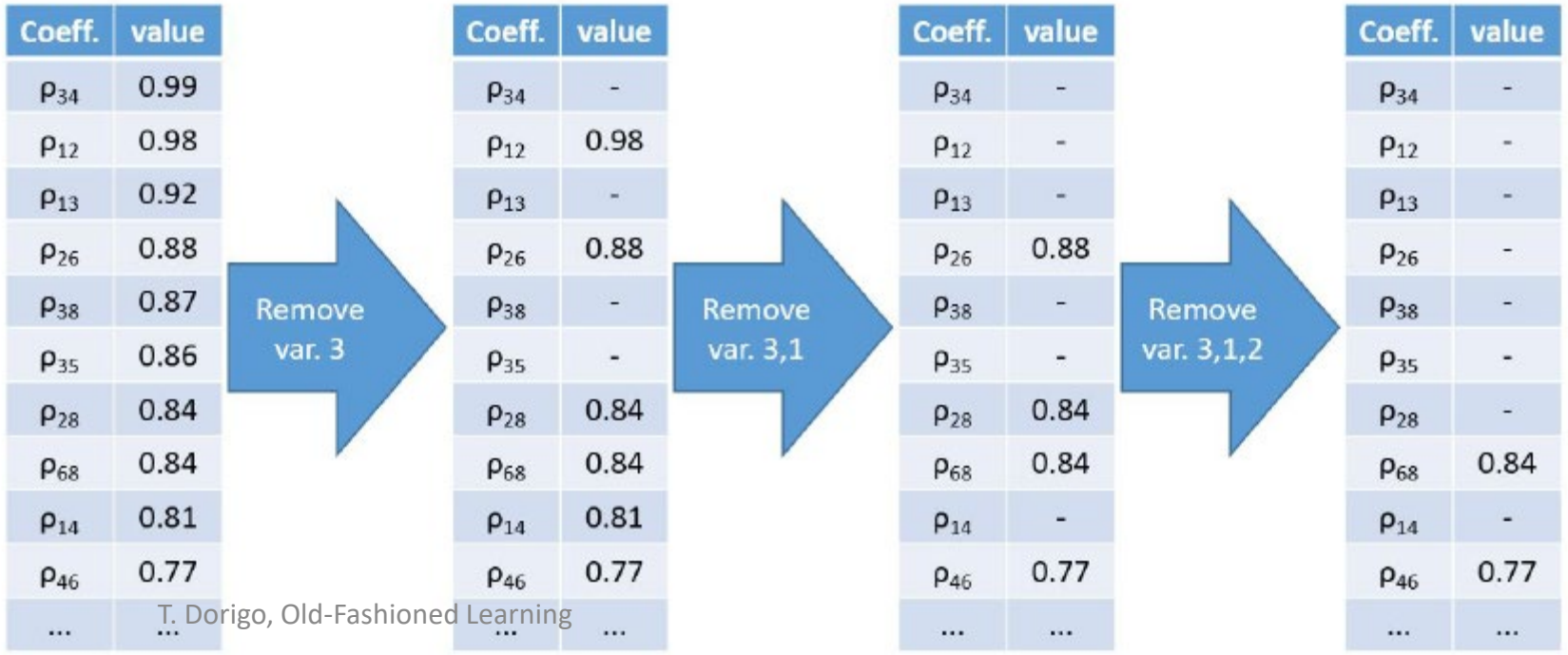
# Correlated variables removal (CVR) procedure

PCA may not be a good idea when variables that are very signal-sensitive have little variance → we risk removing them

A more careful procedure is the following:

1. Choose M, number of variables to be removed (out of N)
2. Compute Nx(N-1) pairwise correlation coefficients
3. List them in decreasing order
4. Find set of M variables that, once excluded from list, minimizes highest remaining correlation coefficient

Below, the choice of removing variables 1,2,3 minimizes the surviving correlation (algorithm is not iterative – it finds best combination by brute force search, so graph is a bit deceiving...)



# Summarizing: Standardization recipe

The standardization recipe we need for anomaly detection emerges as follows:

1) rescale features x such that they have zero mean and unit variance
2) (optionally) apply PCA or (better) CVR, if there are large correlations
3) Compute cumulative distributions of marginals; create new variables x' = $\int_{-\infty}^{x} f(t)dt$
4) **Study D'-dimensional space of x'**

# What now?

We now have our data in a nice, unit-volume D'-dim cube, and no pair of features have a large linear correlation. Any disuniformity in this space is now potentially interesting

**Physics** creates disuniformities.

- An unsupervised anomaly detection algorithm may spot them and assess their departure from a uniform density hypothesis
  - It may be able to find surprising structures, or only stick to completely predictable ones
  - eventually a human intervention is required to scan and assess the results

RanBox is tasked with find overdense regions of discrete data sparsely populating a high-D' copula space

# RanBox

The curse of dimensionality affects any discrete set of data thrown in large-D' spaces.

- But there are ways to measure local densities - e.g. kernels

Our task is to identify interesting overdense regions in subspaces of D'

- we have the preconception that some, if not most, of the features of NP events will not be "anomalous" → hence we need to **focus on subspaces**

We approach this problem by stochastic means:

- we randomly pick a subset of the features
- we scan the resulting space with a D"<D'-dim parallelepiped (a "box"), looking for box boundaries which maximize some "pseudo-significance" of the clustering of data captured within the box

Since our full data (N points) are contained in a space of volume=1, it is very simple to compute how many data points are expected (in the fully uniform hypothesis) in an arbitrary box:

$$N_{exp} = (N * \text{box volume})$$

A better recipe consists in generating a D''-dimensional sideband box around the tested volume, and predict

$$N_{exp} = \tau N_{sb}$$

# * $Z_{PL}$ definition

We need to define a measure of the "pseudo-significance" of observing $N_{box}$ data inside a box, when we expect $N_{exp}$ from a sideband. This is the **so-called "on-off" problem** often encountered in astrophysics searches

A handy definition was proposed by Li and Ma [T. Li, Y. Ma, Astrophysical Journal 272 (1983) 317] in the form of the "Profile Likelihood" $Z_{Pl}$ obtained from:

$$\mathcal{L}_{\mathrm{P}} = \frac{\left(\mu_{\mathrm{s}} + \mu_{\mathrm{b}}\right)^{n_{\mathrm{on}}}}{n_{\mathrm{on}}!} e^{-\left(\mu_{\mathrm{s}}+\mu_{\mathrm{b}}\right)} \frac{\left(\tau\mu_{\mathrm{b}}\right)^{n_{\mathrm{off}}}}{n_{\mathrm{off}}!} e^{-\tau\mu_{\mathrm{b}}}$$

(tau is ratio of time spent looking off and on source)

whence one gets

$$\Lambda(\mu_{\mathrm{s}}) = \frac{\mathcal{L}\left(\mu_{\mathrm{s}}, \tilde{\tilde{\mu}}_{\mathrm{b}}\left(\mu_{\mathrm{s}}\right)\right)}{\mathcal{L}\left(\tilde{\mu}_{\mathrm{s}}, \tilde{\mu}_{\mathrm{b}}\right)}$$

and

$$Z_{\mathrm{PL}} = \sqrt{-2 \ln \Lambda\left(\mu_{\mathrm{s}} = 0\right)}$$

- R.D. Cousins, J. Linnemann and J. Tucker examine that and other definitions [Arxiv:0702156]. Bottomline, **$Z_{PL}$ is quite okay for fast calculations**

# $R_{reg}$

In many cases, large-scale correlations suggest RanBox to focus on large areas, where even small (in percentage) overdensities make $Z_{PL}$ become large

A convenient alternative is $R_{reg} = N_{obs}/(N_{exp}+1)$

In case the signal is small and localized, this density ratio works better

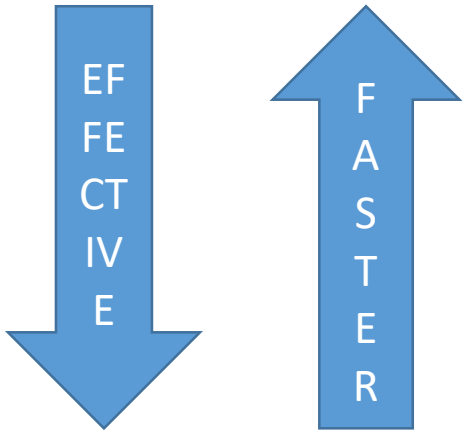
# Other nuts and bolts

The search for the highest-significance box is complicated, because data are sparse. There are a huge number of local maxima wherein a maximization routine can get stuck.

One thus needs a smart seeding to inform the choice of initial box boundaries, easing the convergence task

Three strategies considered:

- **random initial box** boundaries
  - fastest but... random
- window around region of **maximum kernel density**
  - slow, but good unless high-D sparsity makes it ineffective
- area containing **cluster of close points**
  - best for high-D search, slowest



# * Seeding recipe

1) Random box: choose two numbers in [0,1] per each dimension → initial volume is $V=(1/3)^N$. For high-dim searches it is useful to enforce larger total volume.

2) Kernel density: Substitute each data point with a (wide) multi-D Gaussian density, compute sum of densities, get

$$\arg max_{x_j} \sum_{i=1}^{N} G(x_j - x_i)$$

then pick initial box boundaries as $[x_j\text{-}k, x_j\text{+}k]$
- k=0.2 reasonable choice for the cases explored so far

3) Find box boundaries with clustering routine

→next slide

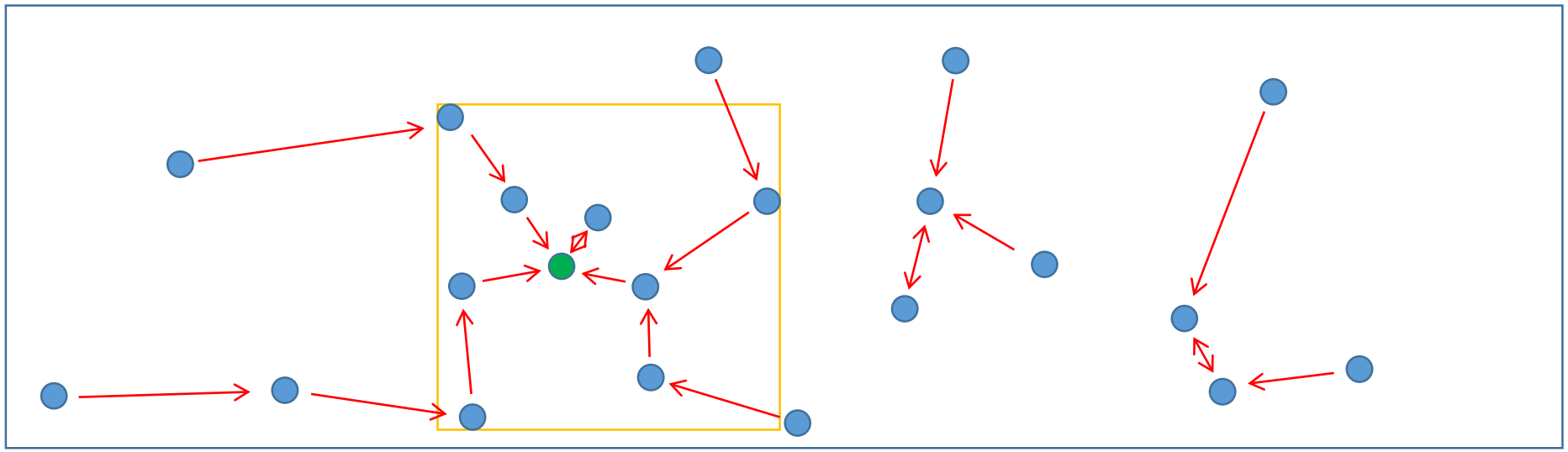
# * Clustering for seeding

We are searching for a local overdensity, so it looks reasonable to seed the box search with the point where several data events are close in space.

Routine:

1) for each data event i, find list of other data events $j_1...j_N$ that have i as closest (relatives); **identify event i*** which has maximum number of relatives
2) for each relative of i* $j_1...j_N$, find list of data events that have $j_k$ as closest (relatives of $j_k$)
3) Draw **minimum box in multi-D** which includes all second-order relatives of i*

Many variants investigated – some perform similarly to this one



# The full RanBox routine

- **Standardize** data boundaries in D-dim space
- **Do PCA or corr. var. removal**, select D‘ features
- Apply **integral transform** to all features
- **Loop** on subspaces - repeat many times:
  - Choose at random subspace D" of D' of workable dimensionality (e.g. D"=6-10)
  - **Find seed box** in D"
  - Maximize $Z_{PL}$ or $R_1$ by **gradient descent**

The procedure produces an ordered list of the boxes of best approximated pseudo-significance or density ratio, in the corresponding most promising subspaces

# * Maximization: gradient descent

Being autharchic, I did not rely on prepackaged minimization routines, and wrote my own gradient descent routine. "The problem has distinct features that make it amenable to specialized solutions"

Not interesting to discuss details here, except that indeed:

- discreteness and sparsity of data in multi-D create large number of minima, noisy convergence to good ones
  - useful to schedule periodic variation of learning rate
- we are **not interested in generalization**, as we want to minimize on data at hand
  - stochastic GD useless
- Acceleration not really crucial but it does improve speed

# RanBoxIter

The random scan of subspaces may be ineffective if the dimensionality of retained features is very large, and/or if the size of data is not large

RanBoxIter solves the dimensionality problem in a different way:

1. Look in all 2D subspaces, find best box in each
2. Select most promising 2D subspaces (e.g. 20)
3. Look in all 3D subspaces which include the 2D ones found above, find best box in each
4. Add dimensions and iterate until a given max D is reached

The procedure may in principle fail to focus on the subspaces spanned by signal-sensitive variables, but it is in practice roughly as effective as the random scan, and even a bit faster

# Tests on synthetic data

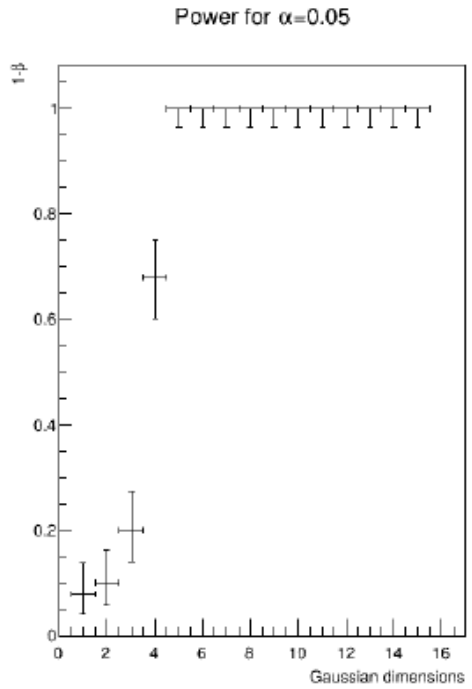
To test the algorithms, we generate a perfectly flat background in N dimensions, and add to it a signal component with M<N features distributed as a multivariate Gaussian (e.g. sigma = 0.1, random mean, random correlations)

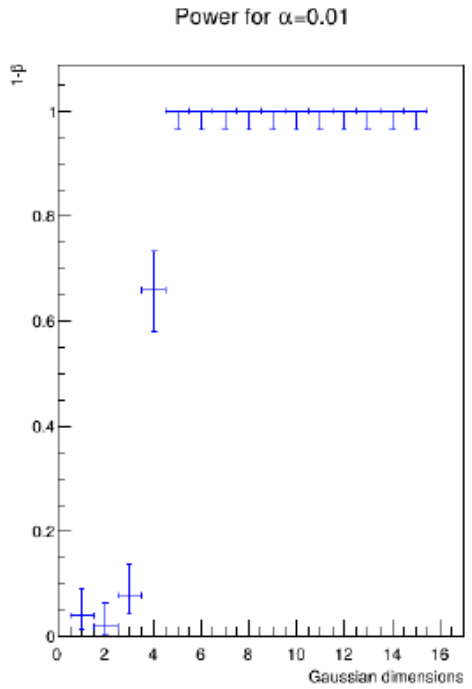
Other details:

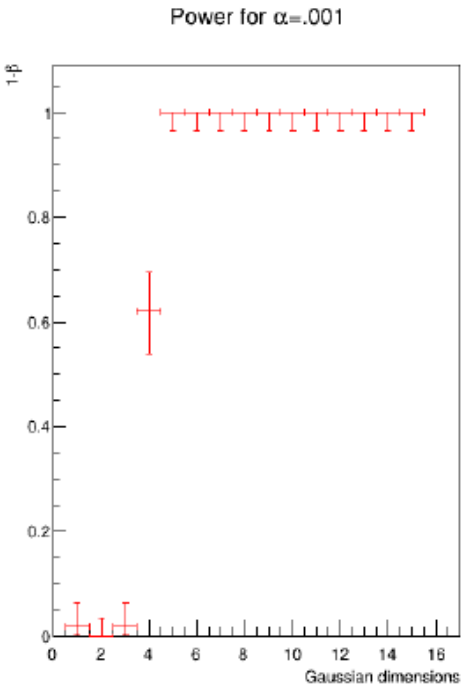
- use 5000 events, vary signal fraction,
- use 5000 events with signal fraction = 0.01, vary number of Gaussian-distributed features
- Determine TS distribution from runs with no injected signal
- Dimensions of space: D=20; RanBox does 1000 trials, D'=6.

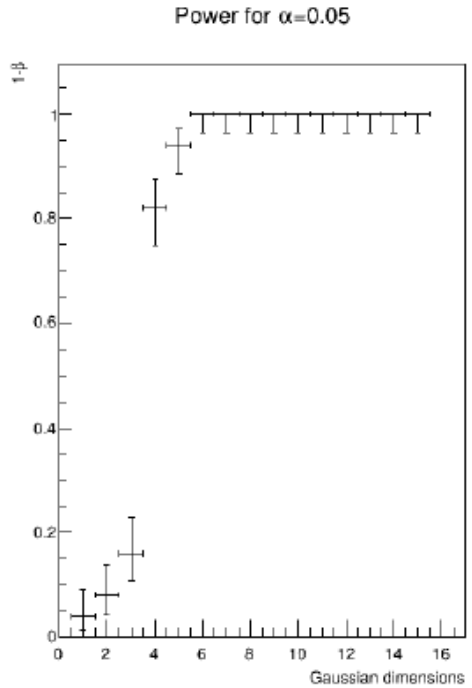
# Power results

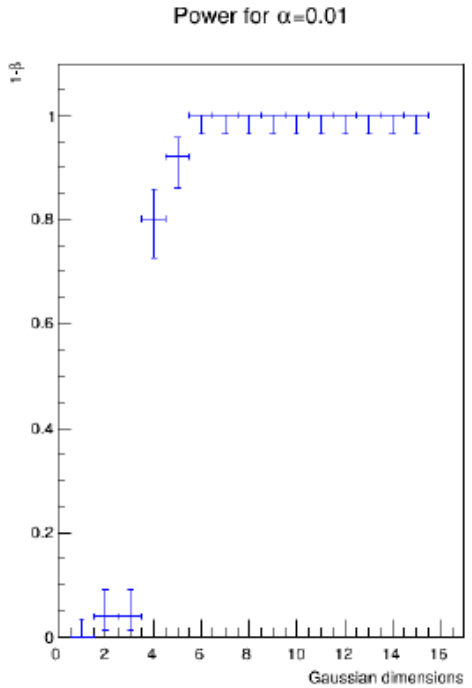
RanBox (top) and RanBoxIter (bottom) both consistently find a 50-event signal in 5000 events, until the number of Gaussian-distributed features goes down to 4 or less

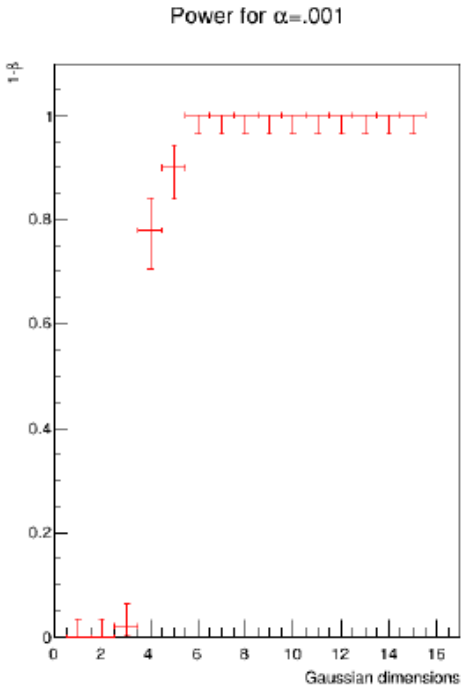


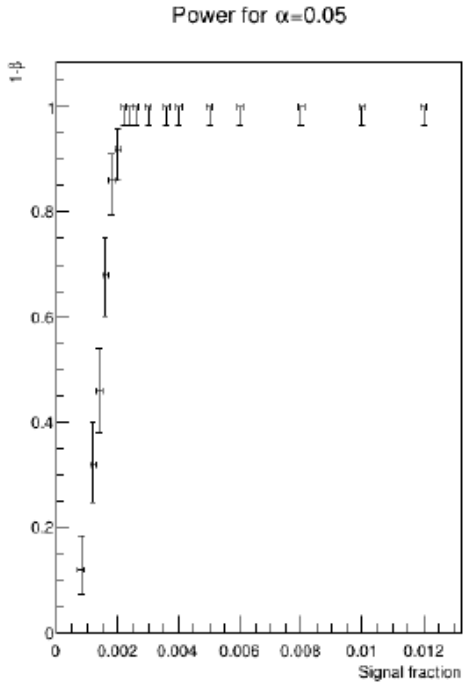



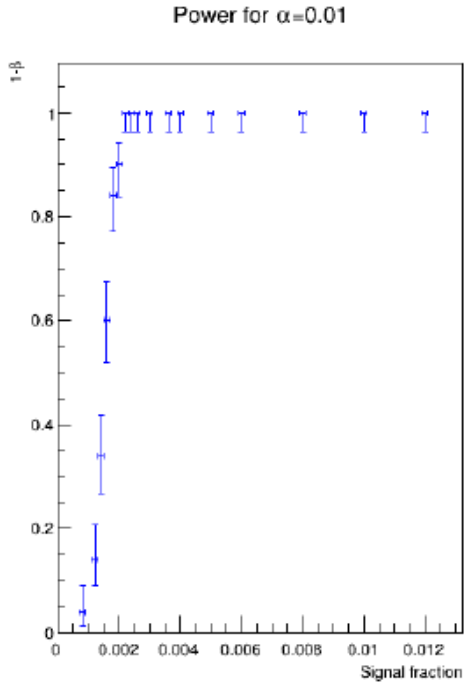



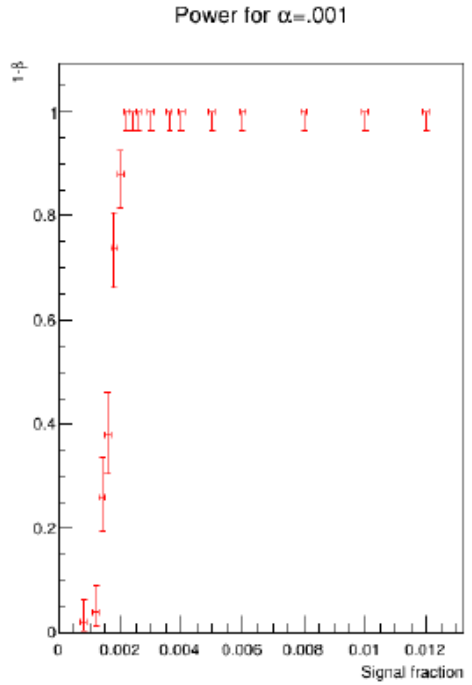



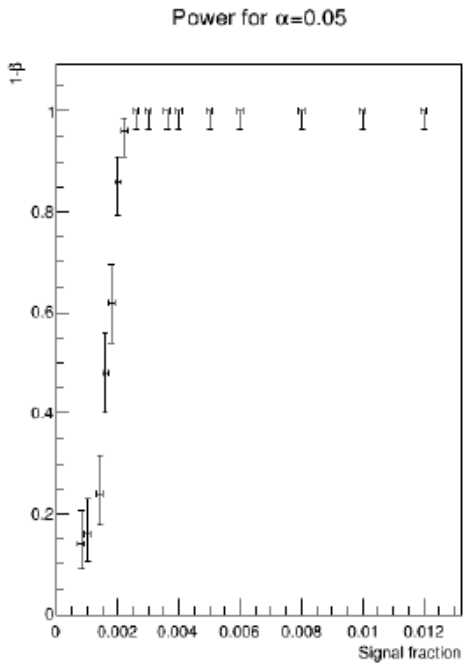
# Power results/2

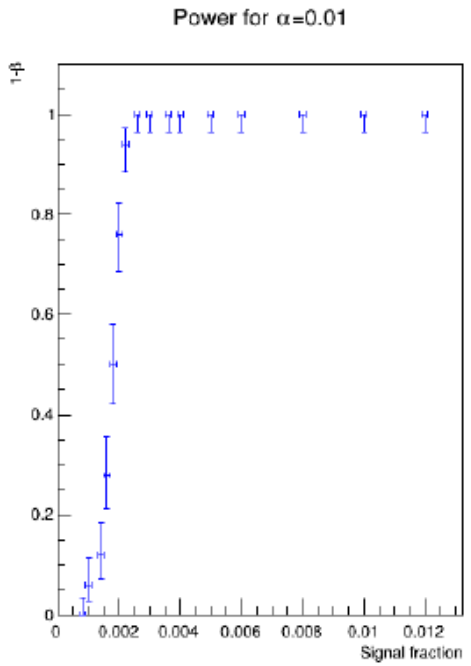
With a distinguishable signal (15 Gaussian dimensions) the anomaly can be spotted for signal fractions down to less than 10 events out of 5000, IFF the background is sampled from a Uniform distribution

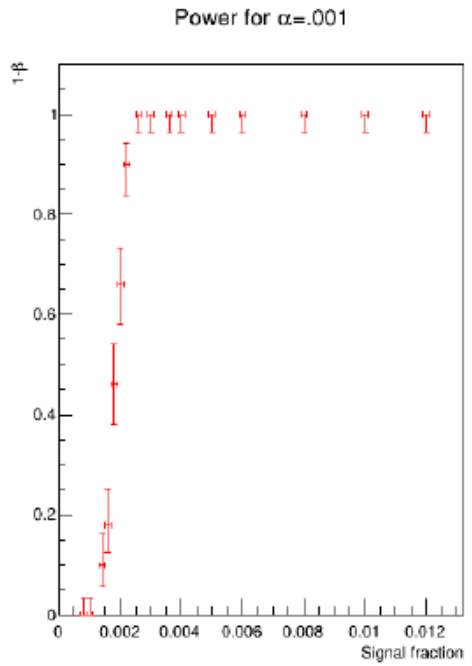


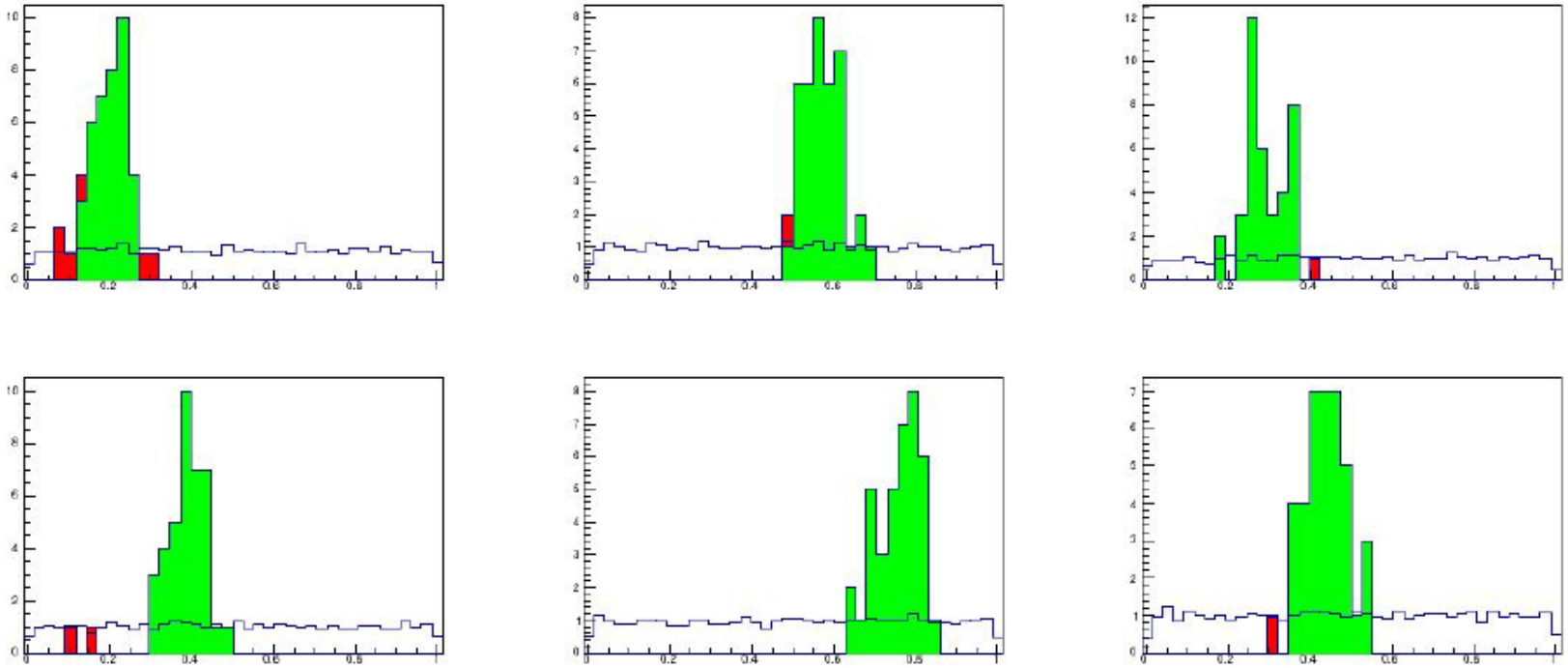



# Data in the copula space

The six features identified by a RanBox run on 50 signal/4950 background event dataset with 11 Gaussian features show that the algorithm correctly focuses on the overdensity in the copula space

In blue is the original data distribution (5000 events);
in green are the 38 events selected in the most significant box;
in red are events that are not included in the box only because of the value of the feature shown



# * 2D plots

The view in 2D gives more info on what the algorithm is looking at, and what it converges to

For each pair of plots, the one on the left shows the totality of the data, and the one on the right events selected in the box in all but the two shown features (in green the box events, in red those failing the selection on the shown features).

Left to right, top to bottom:

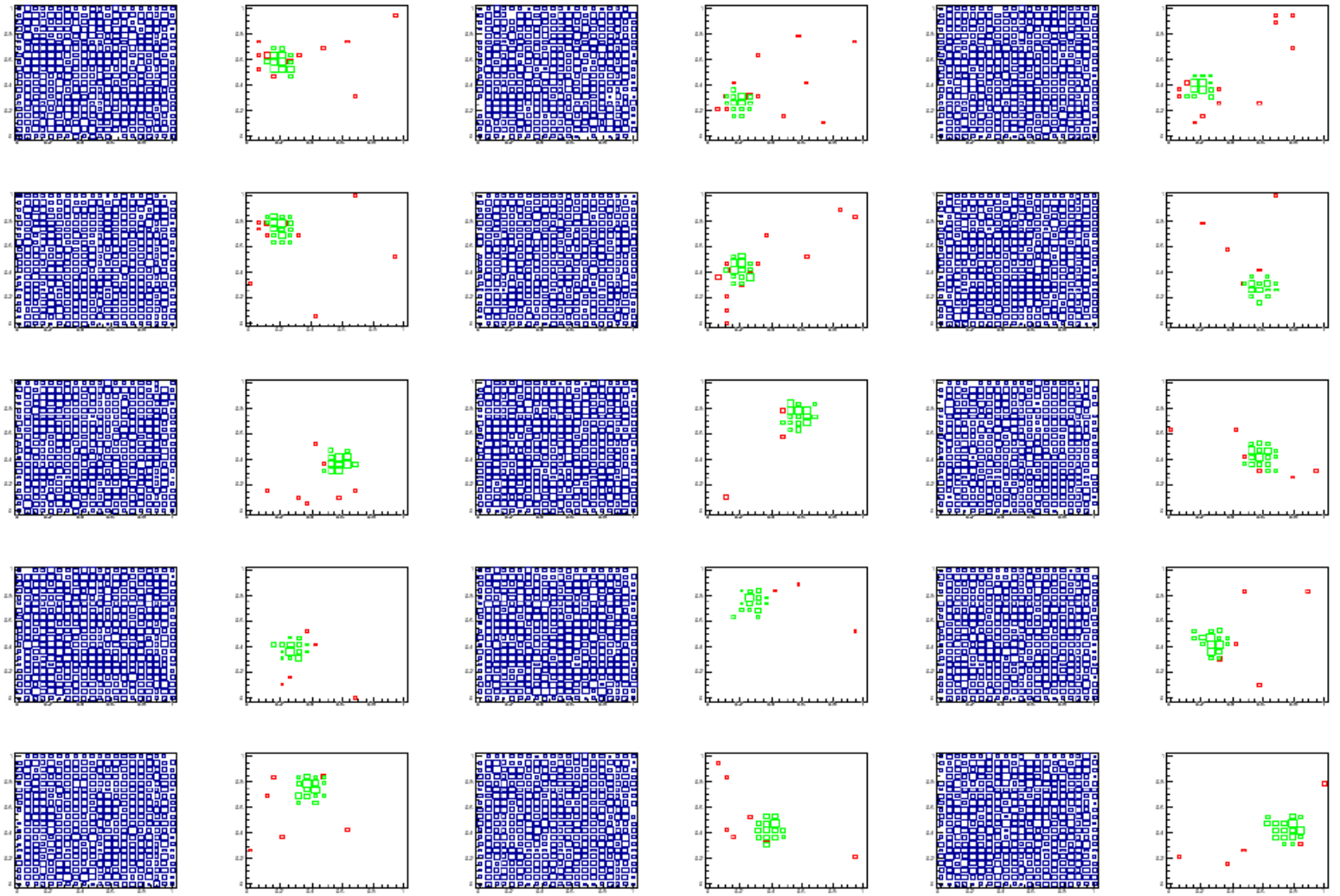
variables (1,2), (1,3), (1,4);

variables (1,5), (1,6), (2,3);

variables (2,4), (2,5), (2,6);

variables (3,4), (3,5), (3,6);

variables (4,5), (4,6), (5,6).



# Experiments – 1: HEPMASS dataset

We study a dataset in the UCI repository, with a X->tt signal ($M_X$=1TeV) and tt background. This data have 27 features.

RanBox results (best 10 boxes) on 250 signal/4750 background events:

RanBox easily finds a 5% signal, evidencing it in the five most significant boxes.

Similar results are obtained by RanBoxIter (see paper).

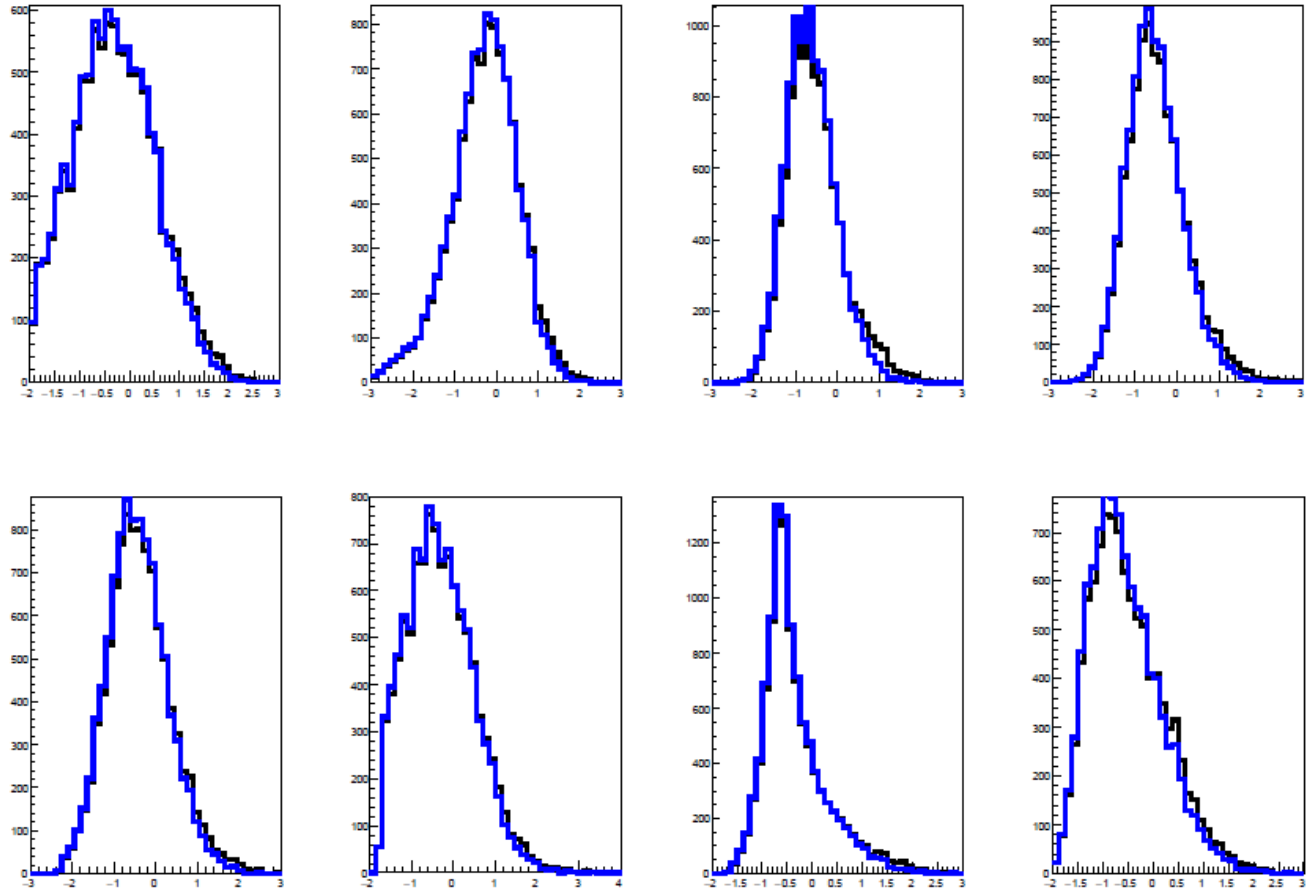
| $R_1$ | $N_{in}$ | $N_{exp}$ | $N_s$ | $\epsilon_s$ | Gain | Active features |
|---|---|---|---|---|---|---|
| 52.78 | 54 | 0.02 | 46 | 0.184 | 17.04 | 10101110011101000000110001 |
| 45.35 | 48 | 0.06 | 38 | 0.152 | 15.83 | 00011110001100101100010011 |
| 41.60 | 46 | 0.11 | 33 | 0.132 | 14.35 | 10001001011011011000100001 |
| 40.72 | 46 | 0.13 | 18 | 0.072 | 7.83 | 10100011011010100010010011 |
| 40.38 | 44 | 0.09 | 41 | 0.164 | 18.64 | 10010010010001000111011001 |
| 40.17 | 47 | 0.17 | 0 | 0.000 | 0.00 | 01100100010001011001100011 |
| 39.82 | 44 | 0.10 | 0 | 0.000 | 0.00 | 10000101010001100101010011 |
| 38.54 | 44 | 0.14 | 0 | 0.000 | 0.00 | 00100110110111000110110000 |
| 38.36 | 44 | 0.15 | 30 | 0.120 | 13.91 | 00011010111001000001101011 |
| 38.05 | 43 | 0.13 | 14 | 0.056 | 6.51 | 11000010011000111010011001 |

# 5% may look like a lot, but…

It would be easy to miss a 5% X->tt component in the data.

The standardized distributions of the most discriminating features are shown here for background (blue) and background+5% signal mixture (black)

→ you would need a precise background model to claim a signal in this sample



# Larger statistic sample

With larger statistic we can search for smaller signal contaminations:

| Test | $N_s/N_b$ | $R_1$ | $N_{in}/N_{exp}$ | $N_s$ | Gain | $SR_{1:10}$ | $\overline{\epsilon_s^{1:10}}$ |
|---|---|---|---|---|---|---|---|
| 1 | 1000/99,000 | 40.94 | 41/0.001 | 35 | 85.4 | 3 | 0.009 |
| 2 | 800/99,200 | 46.10 | 52/0.125 | 46 | 110.6 | 5 | 0.025 |
| 3 | 600/99,400 | 42.00 | 84/1.000 | 0 | 0.0 | 2 | 0.008 |
| 4 | 500/99,500 | 37.00 | 185/4.000 | 0 | 0.0 | 4 | 0.012 |
| 5 | 400/99,600 | 51.00 | 102/1.000 | 0 | 0.0 | 1 | 0.001 |
| 6 | 300/99,700 | 36.96 | 37/0.001 | 0 | 0.0 | 1 | 0.007 |

*Table 7: Sample results of* `RanBoxIter` *runs on 100,000 event samples from the HEPMASS dataset, with varying signal fraction. See the text for more details.*

We have signal-rich boxes returned as the best one for signal fraction above 0.8%; for smaller fractions, there are still signal-rich boxes (signal fraction >6* original fraction) in the list of ten highest-TS ones

# Comparison with Autoencoders

To better gauge whether the performance of RanBox is good or not we can see what an autoencoder does on the same data (250/4750 evts)

An architecture with 27 inputs, 6 intermediate nodes with swish activation, 27 output nodes identifies the following subsets as «most anomalous»

| n_sig | n_bkg | frac_sig | frac_bkg | s/b | cut |
|---|---|---|---|---|---|
| 13 | 49 | 0.052 | 0.01031 | 0.26530 | 1.14259 |
| 40 | 209 | 0.16 | 0.04400 | 0.19138 | 0.82392 |
| 50 | 277 | 0.2 | 0.05831 | 0.18050 | 0.76642 |
| 60 | 360 | 0.24 | 0.07578 | 0.16666 | 0.72336 |
| 70 | 424 | 0.28 | 0.08926 | 0.16509 | 0.69651 |
| 80 | 492 | 0.32 | 0.10357 | 0.16260 | 0.66743 |

| quantile | n_sig | n_bkg | frac_s | frac_b | s/b | cut |
|---|---|---|---|---|---|---|
| 0.99 | 10 | 40 | 0.04 | 0.00842 | 0.25 | 1.2135 |
| 0.98 | 18 | 82 | 0.072 | 0.01726 | 0.21951 | 1.0152 |
| 0.96 | 33 | 167 | 0.132 | 0.03515 | 0.19760 | 0.8679 |
| 0.95 | 40 | 210 | 0.16 | 0.04421 | 0.19047 | 0.8234 |
| 0.9 | 71 | 429 | 0.284 | 0.09031 | 0.16550 | 0.6947 |
| 0.8 | 111 | 889 | 0.444 | 0.18715 | 0.12485 | 0.5744 |

The anomalous regions found by the AE are way less signal-rich!

Reminder: best box found by RanBox

| $R_1$ | $N_{in}$ | $N_{exp}$ | $N_s$ | $\epsilon_s$ | Gain |
|---|---|---|---|---|---|
| 52.78 | 54 | 0.02 | 46 | 0.184 | 17.04 |

# Another try with an overparametrized AE

One may try with a different architecture: 27→54→108→27 nodes, using 20% dropout to prevent perfect reproduction of the inputs. The results of this setup are better than the original AE, but still not competitive with those of RanBox:

| quantile | n_sig | n_bkg | frac_sig | frac_bk | s/b | cut |
|---|---|---|---|---|---|---|
| 0.99 | 18 | 32 | 0.072 | 0.00673 | 0.56250 | 0.13119 |
| 0.98 | 32 | 68 | 0.128 | 0.01431 | 0.47058 | 0.11290 |
| 0.96 | 56 | 144 | 0.224 | 0.03031 | 0.38888 | 0.09699 |
| 0.95 | 63 | 187 | 0.252 | 0.03936 | 0.33689 | 0.09214 |
| 0.9 | 104 | 396 | 0.416 | 0.08336 | 0.26262 | 0.07693 |
| 0.8 | 155 | 845 | 0.620 | 0.17789 | 0.18343 | 0.06216 |

| Method | best S/N | $N_{signal}$ | $N_{total}$ |
|---|---|---|---|
| RanBox | 5.75 | 46 | 54 |
| RanBoxIter | 1.31 | 47 | 83 |
| AE 27→6 | 0.26 | 13 | 62 |
| AE 27→108 | 0.56 | 18 | 50 |

# * MiniBooNE dataset

Another test can be done with electron-neutrino interactions in MiniBooNE (famous dataset, used for BDTs testing in 2005). D=50 features are used to classify CC electron events from backgrounds.

250/4750 events are used for tests

Here we test dimensionality-reduction methods.
We see that CVR does better than PCA both with RanBox and with RanBoxIter

In fact we see the pathology of PCA: reducing the dimensionality with it removes any sensitivity to the signal

| | Test | $\mathscr{D}$ | $R_1$ | $N_{in}$ | $N_{exp}$ | $N_s$ | $SR_{1:10}$ | $\overline{\epsilon_s^{1:10}}$ |
|---|---|---|---|---|---|---|---|---|
| RanBox | 1 | 50 | 67.3 | 69 | 0.03 | 49 | 8 | 0.139 |
| | 2 | 40 (PCA) | 36.5 | 73 | 1.00 | 2 | 0 | 0.007 |
| | 3 | 30 (PCA) | 37.5 | 75 | 1.00 | 0 | 1 | 0.014 |
| | 4 | 40 (CVR) | 90.44 | 98 | 0.08 | 62 | 8 | 0.146 |
| | 5 | 30 (CVR) | 64.63 | 68 | 0.05 | 42 | 10 | 0.161 |
| RanBoxIter | 6 | 50 | 101.77 | 106 | 0.04 | 58 | 10 | 0.215 |
| | 7 | 40 (PCA) | 31.0 | 62 | 1.00 | 0 | 0 | 0.001 |
| | 8 | 30 (PCA) | 30.5 | 61 | 1.00 | 0 | 0 | 0.001 |
| | 9 | 40 (CVR) | 103.85 | 111 | 0.07 | 111 | 10 | 0.208 |
| | 10 | 30 (CVR) | 52.8 | 57 | 0.08 | 32 | 10 | 0.120 |

# * How low can we go?

Using RanBoxIter, and CWR with 10 removed features in the MiniBooNE data, we can see how sensitive the algorithm is to small signals, by progressively reducing the signal fraction in 5000 event datasets

→ signal usually found in best box if fraction >=1.4%

| Test | $N_s/N_b$ | $R_1$ | $N_{in}$ | $N_{exp}$ | $N_s$ | $SR_{1:10}$ | $\overline{\epsilon_s^{1:10}}$ |
|---|---|---|---|---|---|---|---|
| 9 | 250/4750 | 103.85 | 111 | 0.068 | 111 | 10 | 0.208 |
| 11 | 225/4775 | 95.90 | 106 | 0.105 | 62 | 10 | 0.183 |
| 12 | 200/4800 | 85.65 | 86 | 0.003 | 52 | 10 | 0.234 |
| 13 | 175/4825 | 77.33 | 81 | 0.047 | 4 | 4 | 0.053 |
| 14 | 150/4850 | 89.50 | 93 | 0.039 | 42 | 10 | 0.368 |
| 15 | 125/4875 | 61.50 | 67 | 0.088 | 31 | 9 | 0.168 |
| 16 | 100/4900 | 60.73 | 61 | 0.004 | 27 | 10 | 0.195 |
| 17 | 80/4920 | 70.40 | 71 | 0.008 | 23 | 10 | 0.205 |
| 18 | 70/4930 | 79.15 | 85 | 0.074 | 21 | 7 | 0.160 |
| 19 | 60/4940 | 70.60 | 74 | 0.047 | 0 | 1 | 0.030 |
| 20 | 50/4950 | 67.50 | 76 | 0.124 | 1 | 8 | 0.186 |

# Conclusions

- Deep learning achieves great results in a number of problems, and that is wonderful! - but we should not forget that there is nothing really special in it: overparametrization and gradient descent can be applied also to more mundane tools, with similar outcomes (but way more CPU-hungry)

- Looking in the copula space with some dimensionality reduction and gradient descent may be a powerful way to search for anomalies

**Not shown today:** a semi-supervised version of RanBox can be used to do data-driven background estimation:

1. define copula using real data
2. initialize box using signal MC
3. maximize test statistic defined as inverse of expected upper limit on signal xs

Currently under test in Bs→ττ CMS analysis!

# Thanks for your attention!