Use of advanced ML techniques in operations and data analysis at LHCb

Selected Topics

Michael D. Sokoloff

DSHEP-2017

The University of Cincinnati
on behalf of the LHCb Collaboration

May 9, 2017

Goals

Machine learning techniques are used for a variety of overlapping purposes in high energy physics. They complement and/or supplement other techniques, and each use case entails trade-offs. LHCb is using ML for:

- better characterizations of data;
  - characterize/identify events most likely to contain interesting events in the High Level trigger (a two-stage software process);
  - discriminate between well-reconstructed tracks and 'ghost' tracks in the reconstruction software;
  - discriminate between particle species (and ghosts) using information from a variety of detectors (PID);
- faster characterizations of data;
  - characterize/identify events most likely to contain interesting events in the High Level trigger (a two-stage software process);
- better use of computing resources;
  - optimize use of storage capacity;
- better use of human resources
  - simplfy HLT software;
  - reduce biases during analysis to minimize effort related to understanding potential systematic biases.

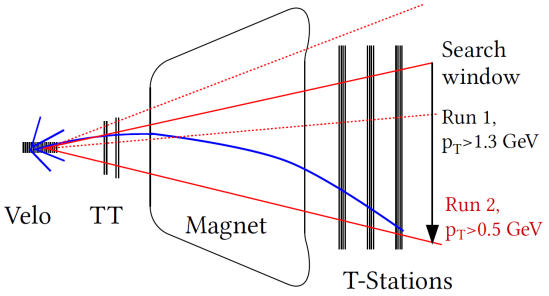
Overview of the LHCb Trigger

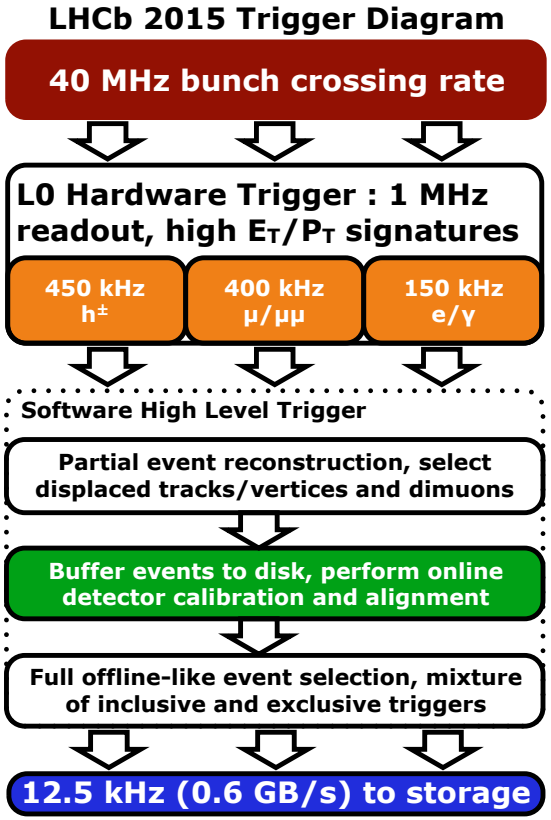
see LHCb-TALK-2016-362 and LHCb-PROC-2017-017

- Moved to *real time reconstruction, alignment and calibration* set-up in Run II

⇝ Need same reconstruction online and offline

- Track reconstruction in two stages
  - Fast stage (HLT1) for long tracks with $p_T > 500$ MeV and tighter track quality requirements
  - Full stage (HLT2) achieves offline efficiency and precision (details in backup)





Use of MVA in the Trigger

see Machine Learning in the LHCb trigger system, V Gligorov, Openlab meeting, CERN 27-04-2017

Where do we use ML/MVA today?

Within trigger, only in HLT. Some attempt made at using neural nets in hardware calorimeter trigger but no significant gain achieved, abandoned.

Reconstruction :

– Track reconstruction	**DNN & BDT (in places)**
– Fake track identification	**NEURAL NET**
– Particle identification	**NN/BDT (in places)**

Selection :

– First level inclusive charm/beauty triggers	**FISHER, BDT**
– Second level inclusive beauty triggers	**BDT**
– Second level inclusive D^* triggers	**BDT**
– Second level inclusive radiative decay triggers	**BDT**

Incidentally : these BDT triggers account for about 66% of the first-level trigger rate and around 30-40% of second-level trigger rate.

About 60% of Run I papers produced using BDT based trigger.

9

Use of Bonsai Boosted Decision Tree MVAs in the Trigger

see Machine Learning in the LHCb trigger system, V Gligorov, Openlab meeting, CERN 27-04-2017 & V. Gligorov and M. Williams, JINST 8 (2013) P02013

Implementation details

BDT weight files versioned separately to the code, which accesses the appropriate version at run-time.

Takes O(50 MB) loaded into memory as a 1D lookup table, but this goes in the shared memory so less of an impact when multithreading

Framework for developing/training/deploying new BDT based triggers minimizes effort for analysts

Machine Learning in Tracking

see LHCb-TALK-2016-362 and LHCb-PROC-2017-017

Machine learning in forward tracking

- NNs trained for background rejection at given (97 to 99 %) efficiency
  - Hidden Layer (HL) architecture most important hyperparameter
  - ⇝ NN in recovery loop (RL): 9 Input nodes, 16,10 HL nodes
  - ⇝ NN after stereo fit: 16 Input nodes, 17,9,5 HL nodes
- NN responses & other parameters tuned with MC and minimum bias data

MC performance 2016 w.r.t. 2015	ν =1.6 w/ RL	w/o RL
timing HLT1	±0 %	
timing HLT2	+4 %	−38 %
fake rate	−27 %	−35 %
fake rate HLT1	−15 %	
ε long	+0.5 %	+0.1 %
ε long from B	+0.2 %	−0.2 %
$\varepsilon_{\text{HLT1}}^{\text{long from B } p>3, p_T>0.5 \text{ GeV}}$	+0.1 %	

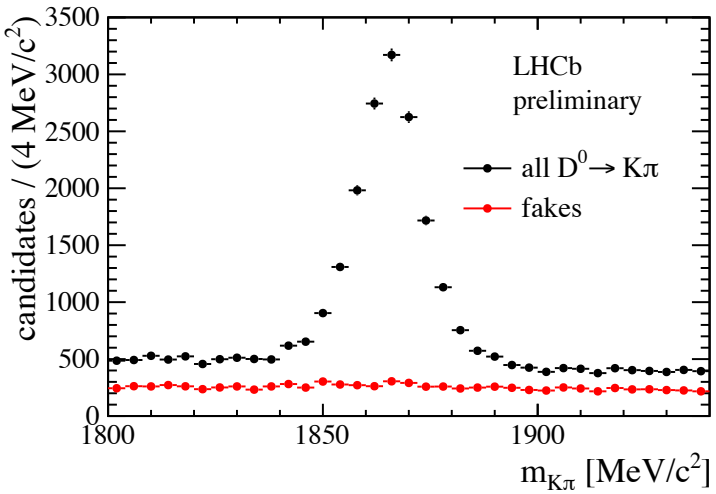
- Results:
  - Increased efficiency
  - Reduced fake rate considerably
  - Decreased speed compensated in later stages due to fake track rejection
  - NNs only contribute 2 % (HLT2), 0.5 % (HLT1) to timing of forward tracking algorithm

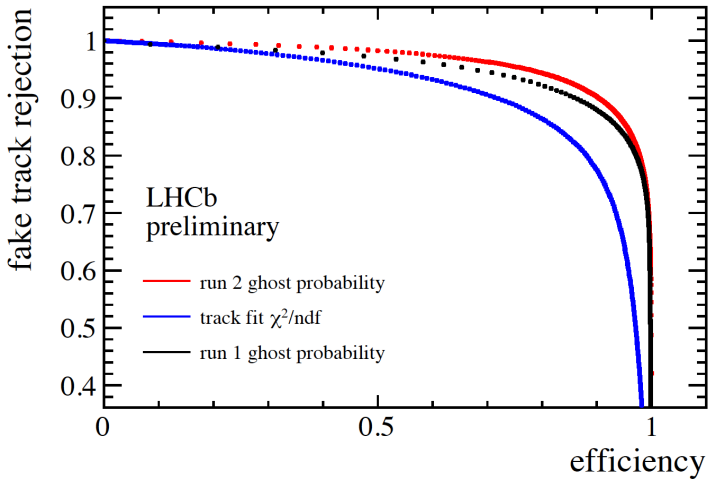
Use of MVA in the Trigger

see De Cian et al., LHCb-PUB-2017-011 (in preparation)

ML for fake track probability

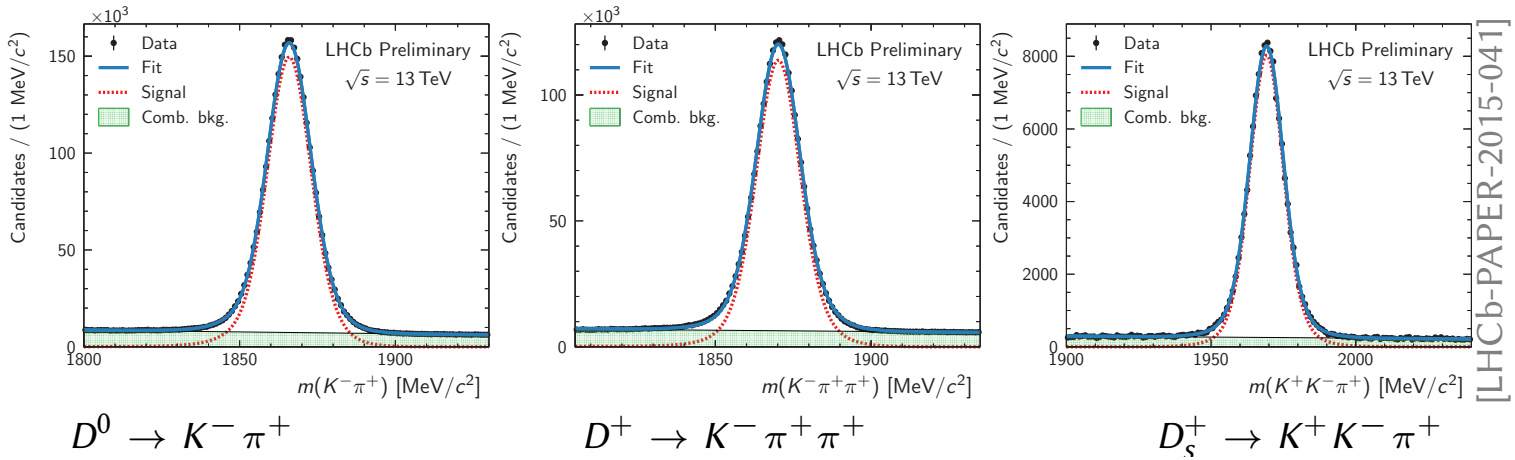
De Cian et al.
LHCb-PUB-2017-011





Fake track probability based on TMVA NN (CE estimator), most important features are hit multiplicities and partial chi2 information in different tracking subdetectors. Main timing cost network evaluation, custom activation function for speed. Extensive use of code profiling and autovectorization to optimize the .C output of TMVA for speed.

Real time signals in 2015



Trigger level signal purities and resolutions for charged particles identical to the best possible offline ones. Published first papers 2 weeks after data taken! 27

Machine Learning for Particle Identification (PID)

see see talk by Tatiana Likhomanenko at IML meeting, 18 January 2017

Challenge

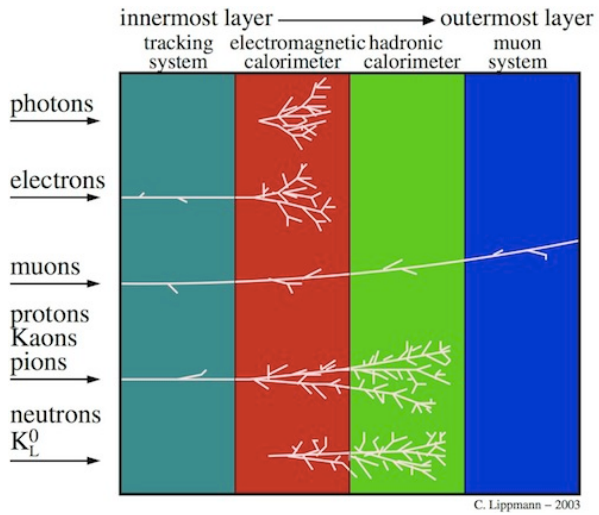
〉 Problem: identify charged particle associated with a track (multiclass classification problem)

  〉 particle types: Ghost, Electron, Muon, Pion, Kaon, Proton.

〉 LHCb detector provides diverse plentiful information, collected by subdetectors: CALO, RICH, Muon and Track observables

  〉 this information can be efficiently combined using ML

〉 Monte Carlo simulated samples for various decays are available (6 millions tracks in training and 6 millions in test)



3

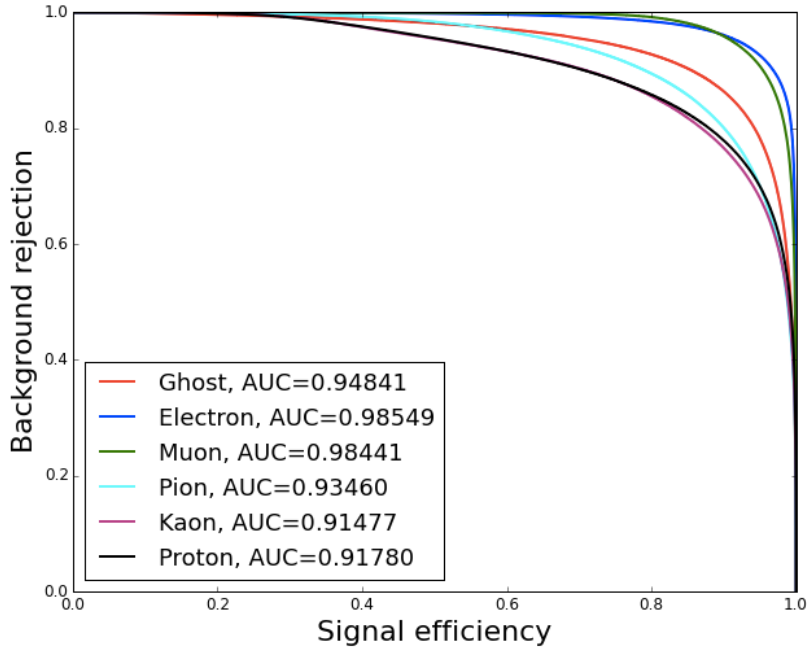
Area Under the Curve (AUC) as a Metric

see see talk by Tatiana Likhomanenko at IML meeting, 18 January 2017

Total Model Quality

Signal efficiencies and background rejection rates derived from real data.

- There are 6 classes, that is why OvR approaches is fine.
- In analyses, we are mostly interested in selecting one type (e.g., muon).
  - We use one-vs-rest ROC curves and area under the curves (AUC) to measure quality of the classification (also for multiclass classification algorithms).



6

Comparing performances of various NNs and BDTs

see see talk by Tatiana Likhomanenko at IML meeting, 18 January 2017

method	ghost	electron	muon	pion	kaon	proton
Baseline NN (one v rest)	0.9484	0.9854	0.9844	0.9345	0.9147	0.9178
Keras DL (multiclass)	0.9632	0.9914	0.9925	0.9587	0.9319	0.9320
Stacked NN	0.9624	0.9911	0.9924	0.9580	0.9316	0.9314
Special NN	0.9622	0.9910	0.9923	0.9573	0.9309	0.9307
XGBoost (grad boosted)	0.9609	0.9908	0.9922	0.9568	0.9303	0.9302
Special BDT (add ln combs)	0.9636	0.9913	0.9926	0.9576	0.9309	0.9310
Flat in p + pT + η + nTracks	0.9600	0.9874	0.9884	0.9503	0.9130	0.9129

- **Multiclass classification works effectively for PID**
- **Modern NN technologies, especially DL, improve PID performance**
- **BDTs and NNs have similar reach**
- **Uniform boosting (flatness) provides slightly lower discriminating power, but also provides better control of biases and systematic uncertainties.**

Boosting to Uniformity

For concepts, see arXiv:1305.7248 [nucl-ex] and arXiv:1410.4140 [hep-ex] and for code see the hep_ml and Yandex repositories.

Uniform gradient boosting trades off some optimization performance for relatively uniform response. This can simplify analyses, control potential biases, and reduce systematic uncertainties.

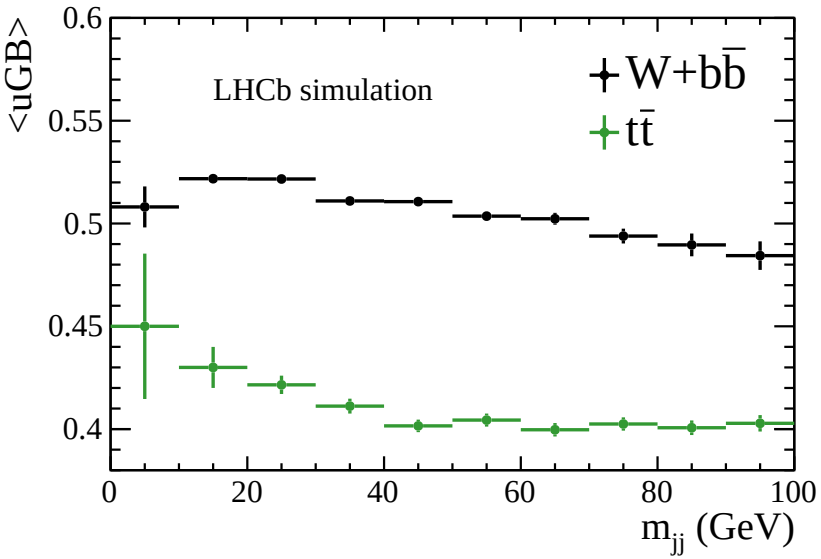


Figure: Average of uGB response in different intervals of m_{jj} for $\mathrm{W}+\mathrm{b\bar{b}}$ (black) and $\mathrm{t\bar{t}}$ (green). The vertical error bars represent the standard error of the uGB mean in each interval. From Phys. Lett. B 767 (2017) 110.

Changing gears: Kernel density estimation (KDE) basics

Let x_i be the data points from which we have to estimate the PDF. Kernel density estimator is

$$P_{\mathrm{KDE}}(x) = \sum_i K(x - x_i)$$

Here $K(x)$ is a kernel. Can use various forms. Here, consider a parabola:

$$K(x) = 1 - (x/h)^2$$

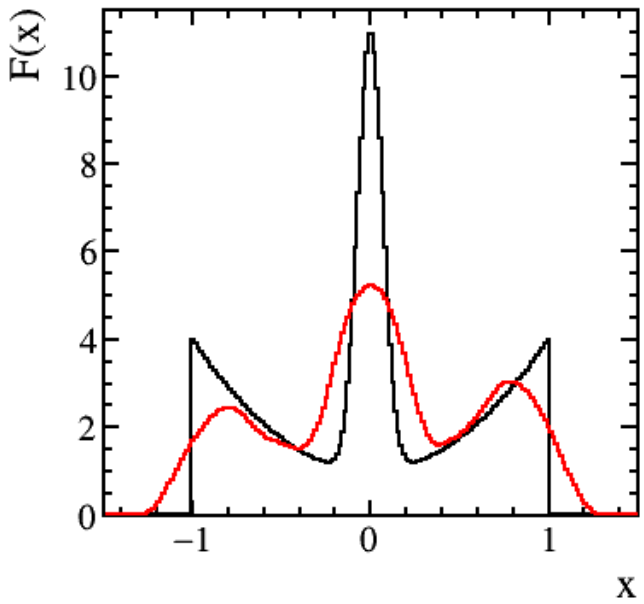
Optimal in some sense (although the others, such as Gaussian, are almost as good).
Note the resulting $P_{\mathrm{KDE}}(x)$ in $i \to \infty$ limit is rather a convolution of the true PDF with the kernel $K(x)$. Thus, structures with the width $\leq$ kernel width are smeared.
Kernel width h (bandwidth) needs to be optimised to reach balance between bias (wide kernels) and stat. fluctuations (narrow kernels).
For HEP-related discussion, see

[K. Cranmer, Comp. Phys. Comm. 136 (2001) 198-207]

KDE implementations

- `RooFit`: `RooKeysPDF` (1-dim), `RooNDKeysPDF` (N-dim).
  - Gaussian kernel
  - Both fixed and adaptive kernels
  - Boundary correction using data reflection
- `scikit-learn`: `sklearn.neighbors.KernelDensity`.
  - Choice of various kernels
  - Only fixed kernel
  - Different metrics
  - Optimisation using KD-tree $\Rightarrow$ faster lookup
- `Meerkat` implementation (by Anton Poluektov): see below
  - Attempt to solve problems related to boundary effects and curse of dimensionality.

KDE: boundary effects



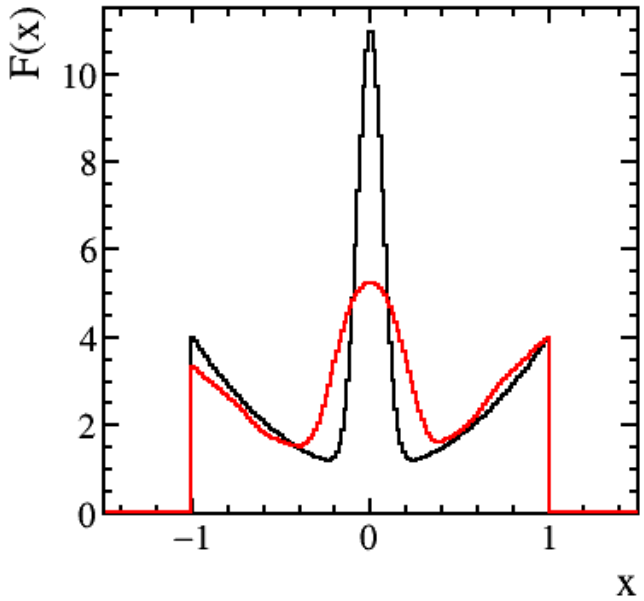
The usual problem with KDE is boundary effects.
Methods to correct for this:

- Data reflection.
- Kernel modification near boundary.

Normally work with simple boundaries (1D, linear). Not easy to apply to *e.g.* conventional Dalitz plots.

$$P_{\text{true}}(x) = 1+3x^2+10e^{-x^2/0.12}$$

KDE: correcting for boundary effect



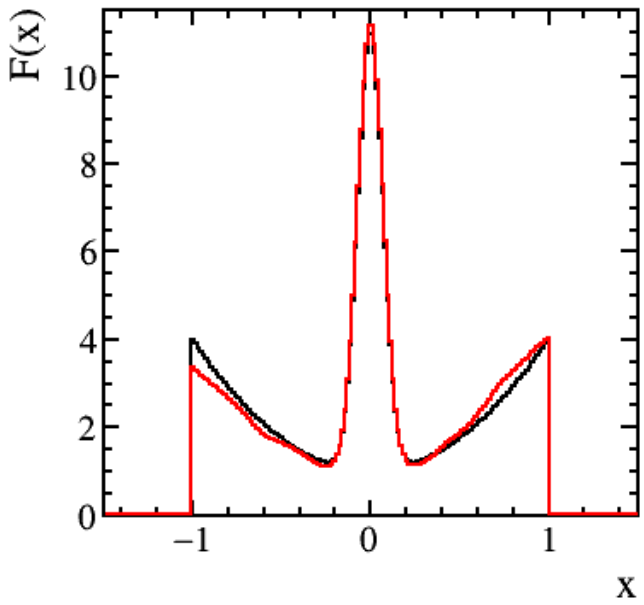
Simple correction: divide result of KDE by the convolution of kernel with flat density:

$$P_{\text{corr}}(x) = \begin{cases} \dfrac{\sum_{i=1}^{N} K(x-x_i)}{(U \otimes K)(x)} & \text{for } x \in X, \\ 0 & \text{otherwise.} \end{cases}$$

$$P_{\text{true}}(x) = 1+3x^2+10e^{-x^2/0.1^2}$$

$$U(x) = \begin{cases} 1 & \text{for } x \in X, \\ 0 & \text{otherwise.} \end{cases}$$

KDE: correcting for boundary effect



$$P_{\text{true}}(x) = 1+3x^2+10e^{-x^2/0.1^2}$$

$$P_{\text{appr}} = 1 + 10e^{-x^2/0.1^2}$$

Suppose we approximately know how the PDF behaves at the boundaries. A more sophisticated correction:

$$P_{\text{corr}}(x) = \frac{\sum_{i=1}^{N} K(x - x_i)}{(P_{\text{appr}} \otimes K)(x)} \times P_{\text{appr}}(x).$$

replaces KDE by an approximation PDF at boundaries and in regions with narrow structures.

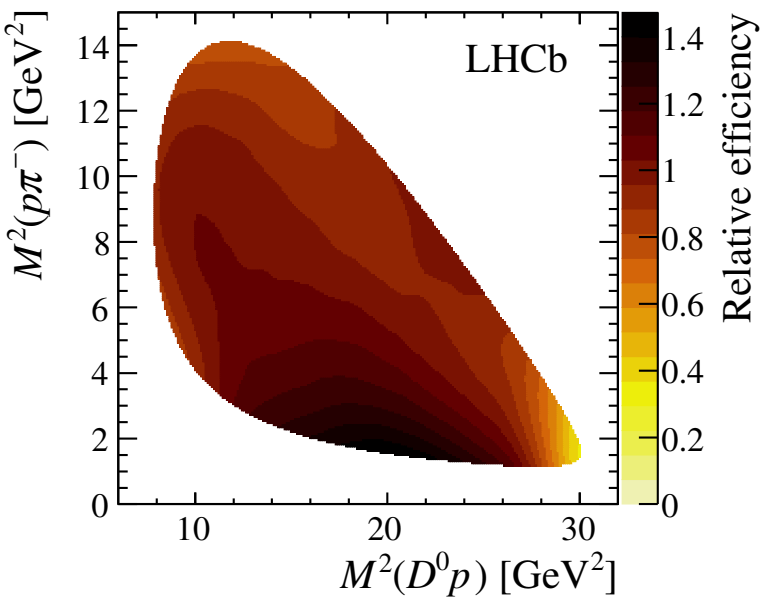
For details and a discussion of how to extend to multiple dimensions, see [Anton Poluektov, JINST 10 P02011 (2015)]

Relative KDE: multidimensional case

Multiple dimensions typically need wide kernels (or very large samples). As an example of how approximation PDFs can help:

Efficiency shape in multiple dimensions:

- Approximation PDF from high-statistics fast MC sample (e.g. generator-level MC with simple kinematic cuts) and narrow kernel.
- Relative KDE based on full Geant simulation and wider kernel.



This is the relative selection efficiency over the $\Lambda_b^0 \to D^0 p\pi^-$ phase space described in arXiv:1701.07873

Meerkat library

[http://meerkat.hepforge.org]

The above procedure is implemented in the Meerkat library (Multidimensional Efficiency Estimation using Relative Kernel Approximation Technique). Obviously not limited to efficiency estimation. Direct usage of relative KDE formulas is *slow* because convolution should be done in every point x. For practical applications, use binned approach with multilinear interpolation:

$$P_{\text{interp}}(x) = \frac{Bin\left[\sum_{i=1}^{N} K(x - x_i)\right]}{Bin\left[(P_{\text{appr}} \otimes K)(x)\right]} \times P_{\text{appr}}(x).$$

Time to estimate the PDF is linear with the size of the sample, and memory is constant (no need to store the whole data sample in memory). Very large data samples can be used practically (A.P. has used a 10^8 event sample for a 5D distribution).

Summary

some selected uses of ML in LHCb

Machine Learning has been used extensively in LHCb since Run 1. It provides many benefits.

- our reconstruction/trigger software executes much more quickly (tracking and BBDT triggers are examples).
- our reconstruction/trigger software characterizes data more effectively, allowing us to do more physics with limited bandwidth out of the pit (BBDT and PID are examples).
- Using BDTs, NNs, DL, and KDEs allows us to do better physics analyses

No one approach addresses all problems most effectively.

- we continue to develop new alogrithms and tools;
- we compare performances with real data.

Thank You