SCHOOL OF DATA ANALYSIS

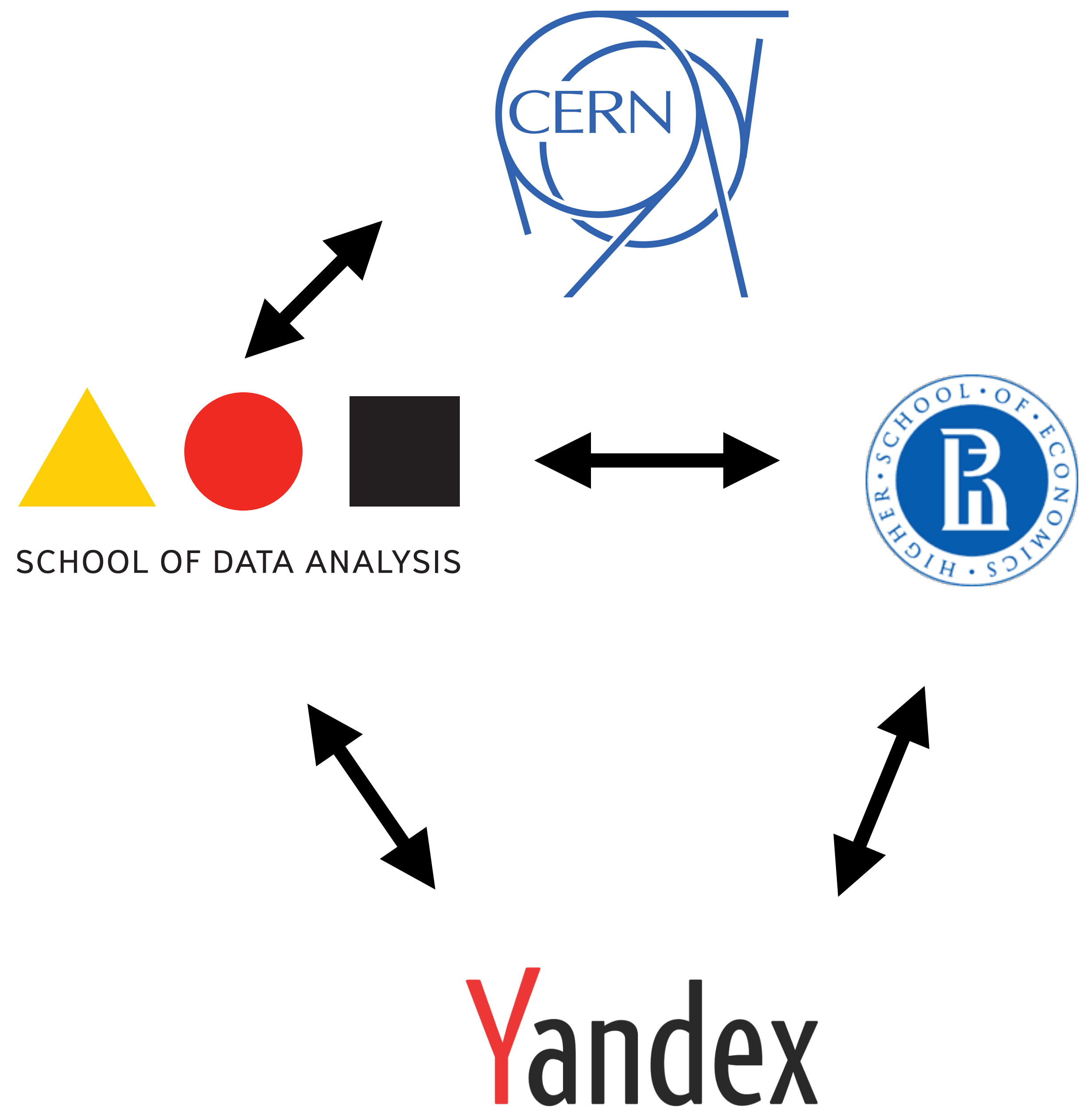
Recent advances in the Machine Learning Methods Applications in the High–Energy Physics

Denis Derkach

Yandex School of Data Analysis and Higher School of Economics, Moscow, Russia

INFN Bologna, 6 September 2016

Who we are?



leading search engine in Russia (and not only)

- Group working on data analyses in Natural sciences
  - 2 physicists and 7 mathematicians (out of them 5 students)
- Part of a nonprofit Yandex School of Data Analysis
- Members of the LHCb collaboration

Aim to apply machine learning in the real scientific world problems

Which tasks?

- Any, that can be formalised as a Machine Learning task
  - good dataset
  - clear rules to select winners
  - formalisable additional conditions
- Examples include:
  - Storage/Speed optimisation for triggers
  - Jet and flavour tagging algorithms
  - Brain cognitive studies
  - Ultra-high Cosmic Ray searches

Outline

- Recent examples of ML algorithm developed for an LHC experiment
  - High Level Trigger
  - Data Popularity
  - Anomaly Detection
- Generalised ML algorithms useful for analysis
  - BDT reweighting
  - flatness boosting

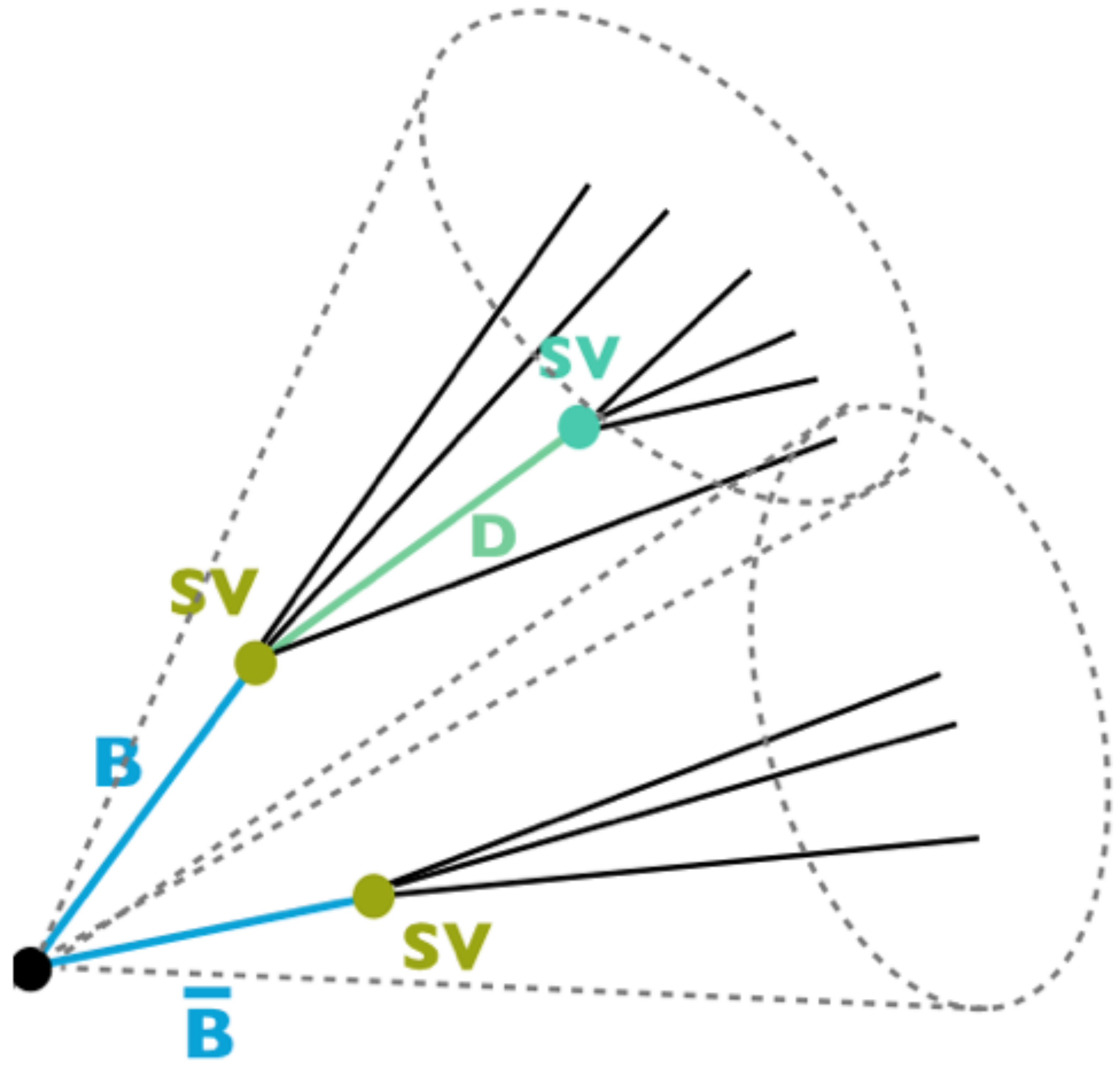
ML in Trigger

LHCb topological trigger

- Generic trigger for decays of beauty and charm hadrons
- Designed to be inclusive trigger for any B decay with at least 2 charged daughters including decays with missing particles
- Look for 2, 3, 4 track combinations in a wide mass range
- Use fast-track fit to improve signal efficiency and minbias rejection

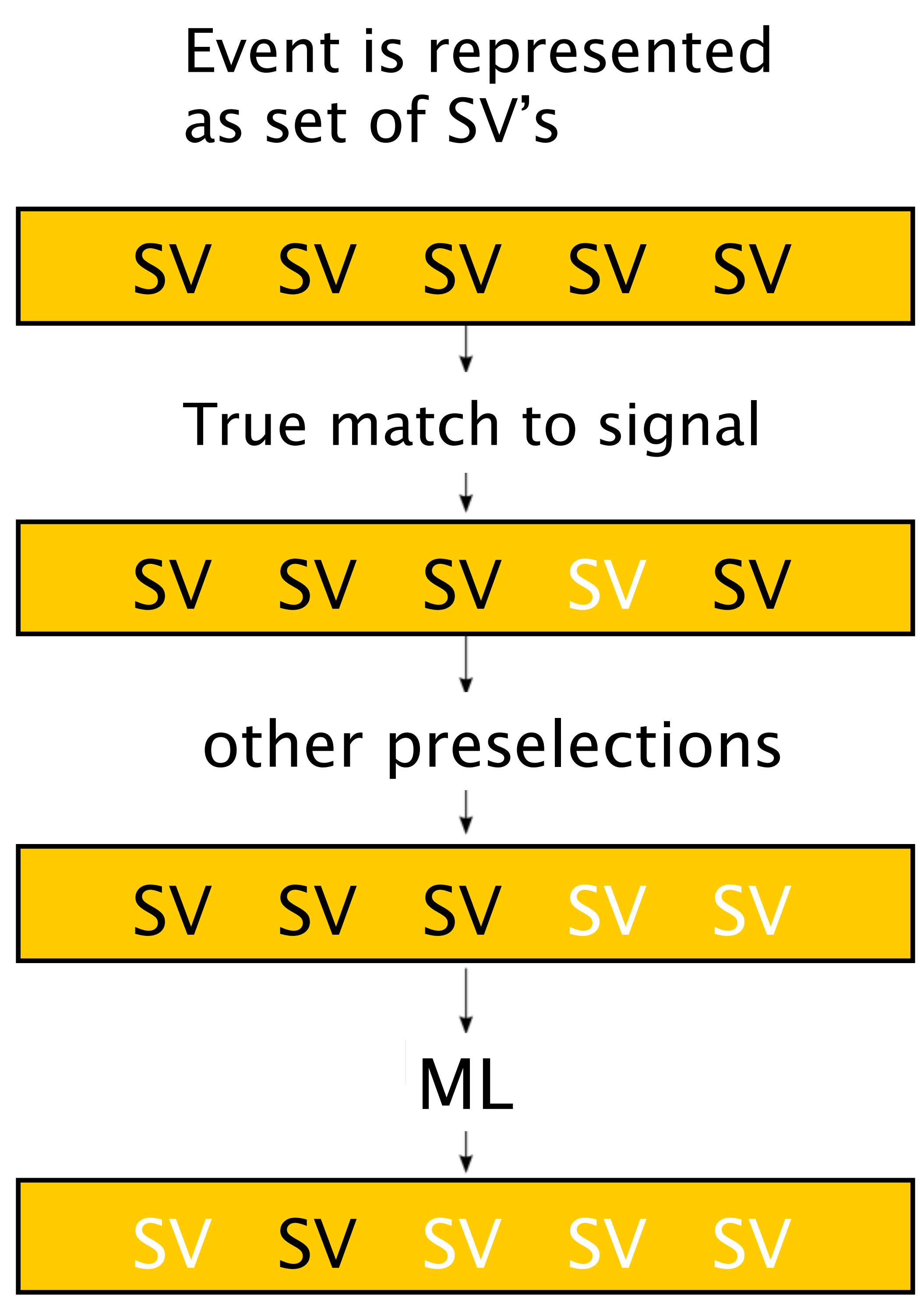
Event

- Sample: one proton-proton collision
- Event consists of:
  - tracks (track description)
  - secondary vertices (SV description)
  - unstructured data
- Questions:
  - How to describe event in ML terms?
  - How to train model on such events?



Data

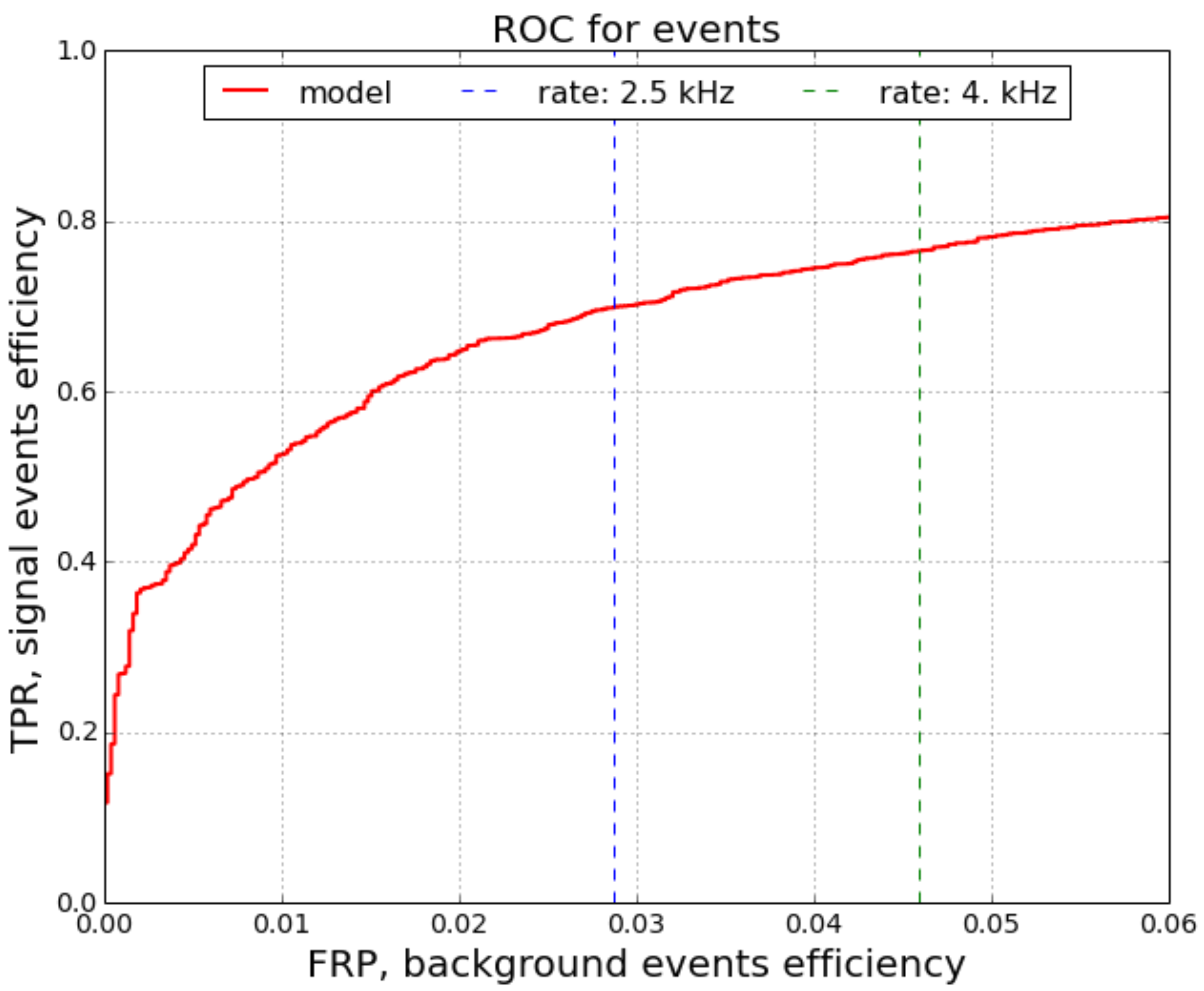
- Monte Carlo samples (used as signal-like) are simulated 13-TeV with B decays of various topologies
- Generic Pythia 13-TeV proton-proton collisions are used as background-like sample (also includes some signal)
- Training data are set of SVs for all events
- Most events have many secondary vertices (not all events have them)
- Goal is to improve efficiency for each type of signal events along fixed efficiency for background



If at least one SV in the event passed all stages, the whole event passes trigger

ROC curve, computed for events

- Output rate = false positive rate (FPR) for events (since background = generic event)
- Optimize true positive rate (TPR) for fixed FPR for events
- Weight signal events in such way that channels have the same amount of events.
- Optimize ROC curve in a small FPR region

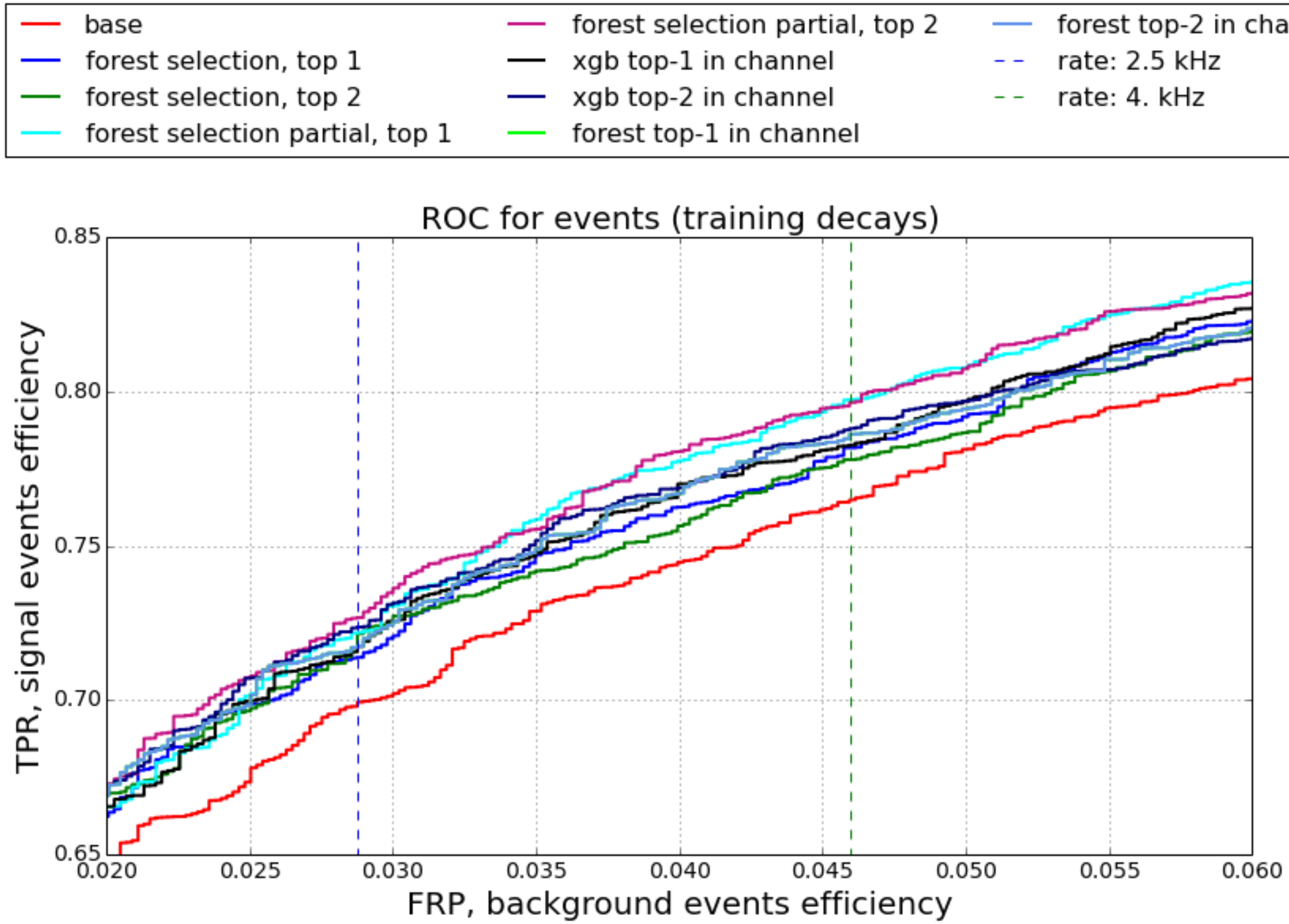


ROC curve interpretation

Random forest for SVs selection

- Train random forest (RF) on SVs (typically ~30 per event)
  - RF is stable to noise in data
  - RF doesn’t penalize in case of misclassification (can find noisy samples)
- Select top-1, top-2 SVs by RF predictions for each signal event
- Train classifier on selected SVs

Random forest for SVs selection



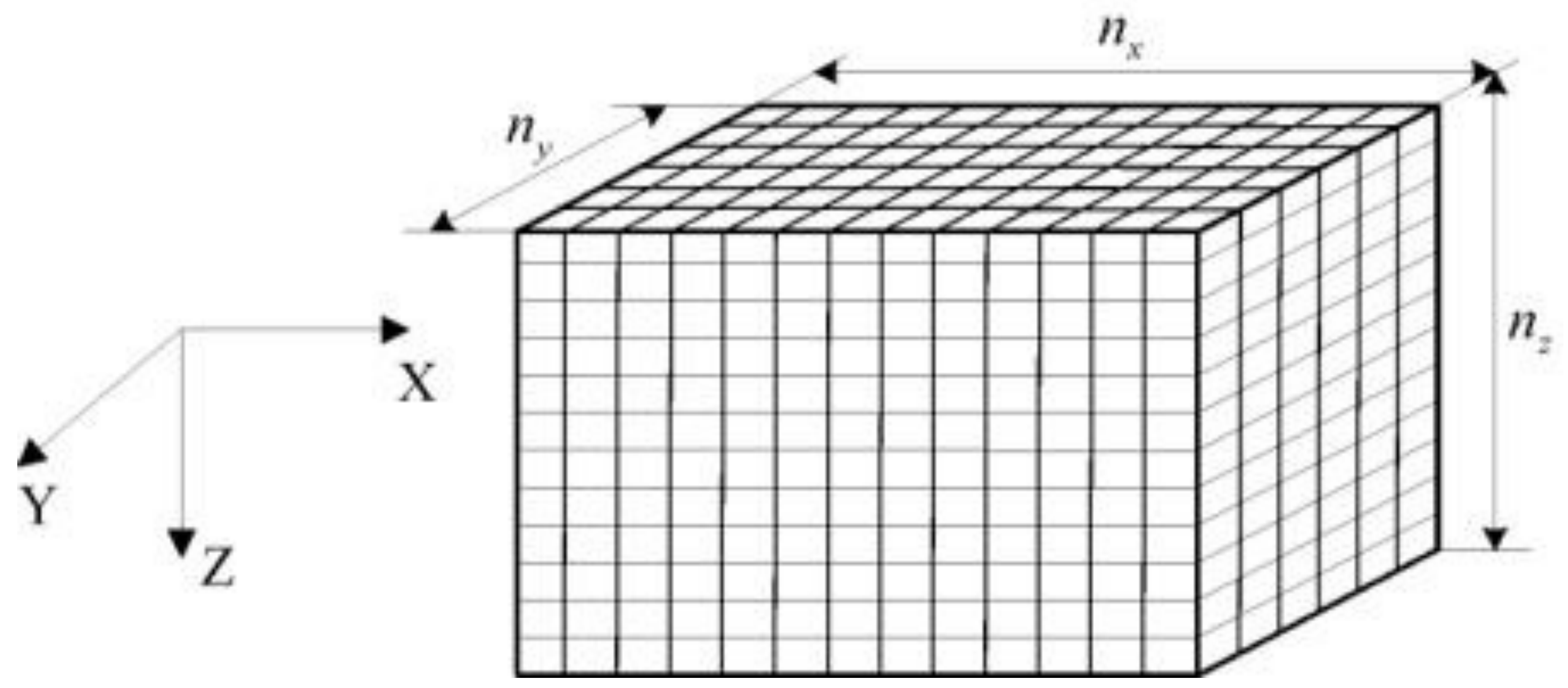
Online processing

There are two possibilities to speed up prediction operation:

- Bonsai boosted decision tree format (BBDT)
- Post-pruning

BBDT

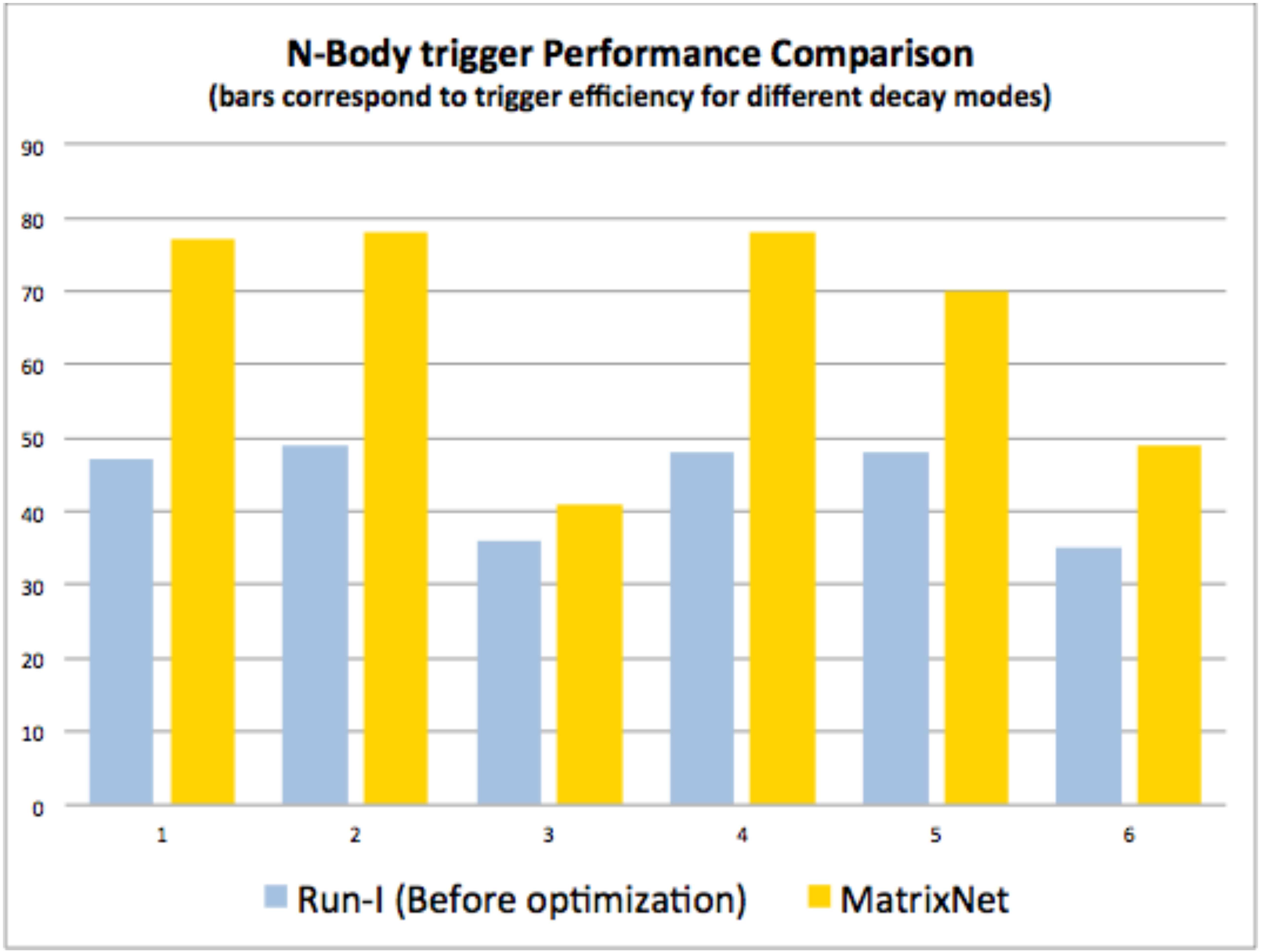
- Features hashing using bins before training
- Converting decision trees to n-dimensional table (lookup table)
- Table size is limited in RAM (1Gb), thus count of bins for each features should be small (5 bins for each of 12 features)
- Discretization reduces the quality
- Prediction operation takes one reading from the table



Post-pruning

- Train MatrixNet (MN) with several thousands trees
- Reduce this amount of trees to a hundred
- Quality stays close to the initial
- Greedily choose trees to minimise a special loss function

Topological trigger results (without RF trick)



1. $B^0 \to K^*[K^+\pi^-]\mu^+\mu^-$
2. $B^+ \to \pi^+K^-K^+$
3. $B_s^0 \to D_s^-[K^+K^-\pi^-]\mu^+\nu_\mu$
4. $B_s^0 \to \psi(1S)[\mu^+\mu^-]K^+K^-\pi^+\pi^-$
5. $B_s^0 \to D_s^-[K^+K^-\pi^-]\pi^+$
6. $B^0 \to D^+[K^-\pi^+\pi^+]D^-[K^+\pi^-\pi^-]$

https://github.com/yandexdataschool/LHCb-topo-trigger

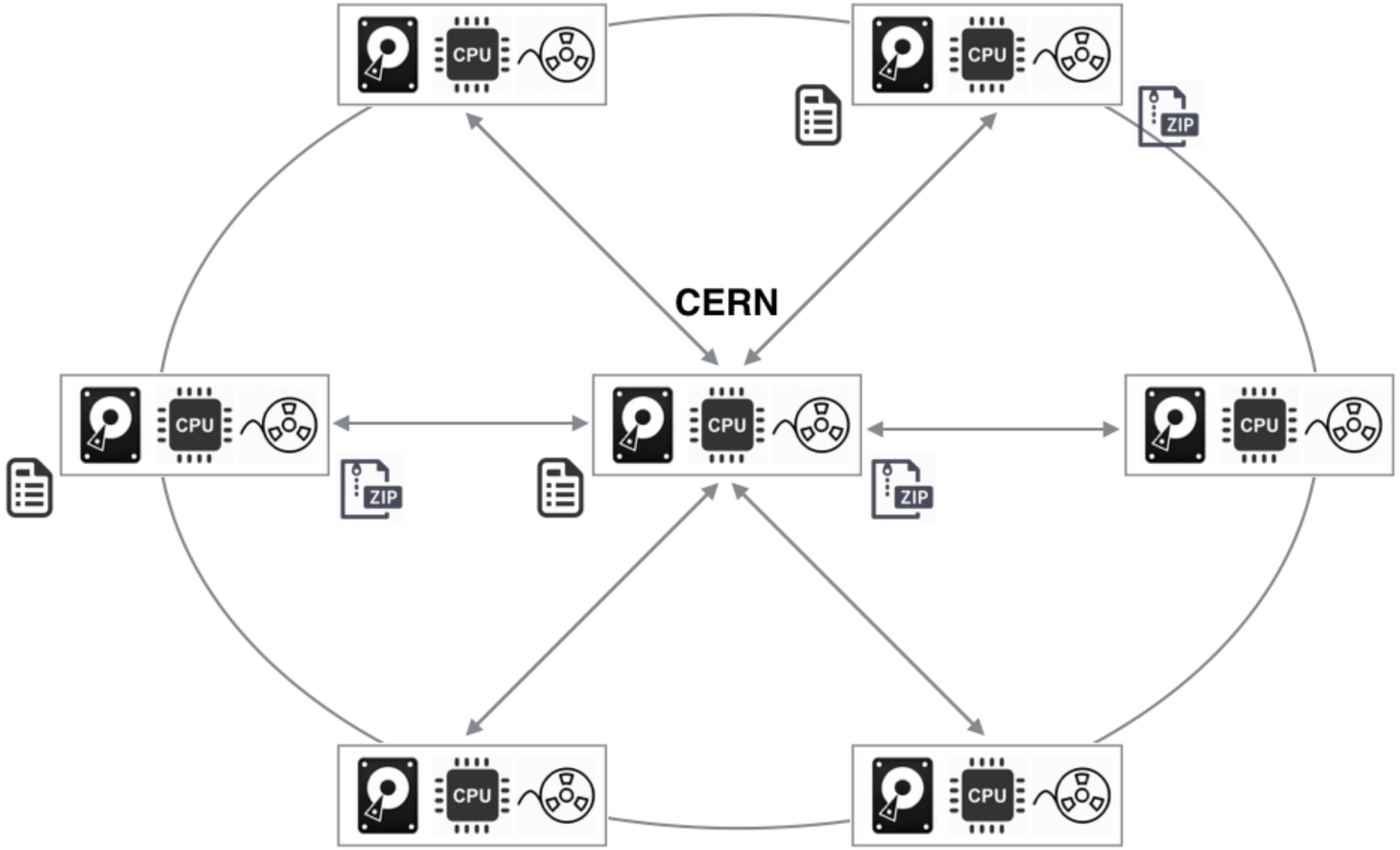
References

- https://github.com/yandexdataschool/LHCb-topo-trigger
- https://cdsweb.cern.ch/record/1384380/files/LHCb-PUB-2011-016.pdf
- http://arxiv.org/abs/1510.00572

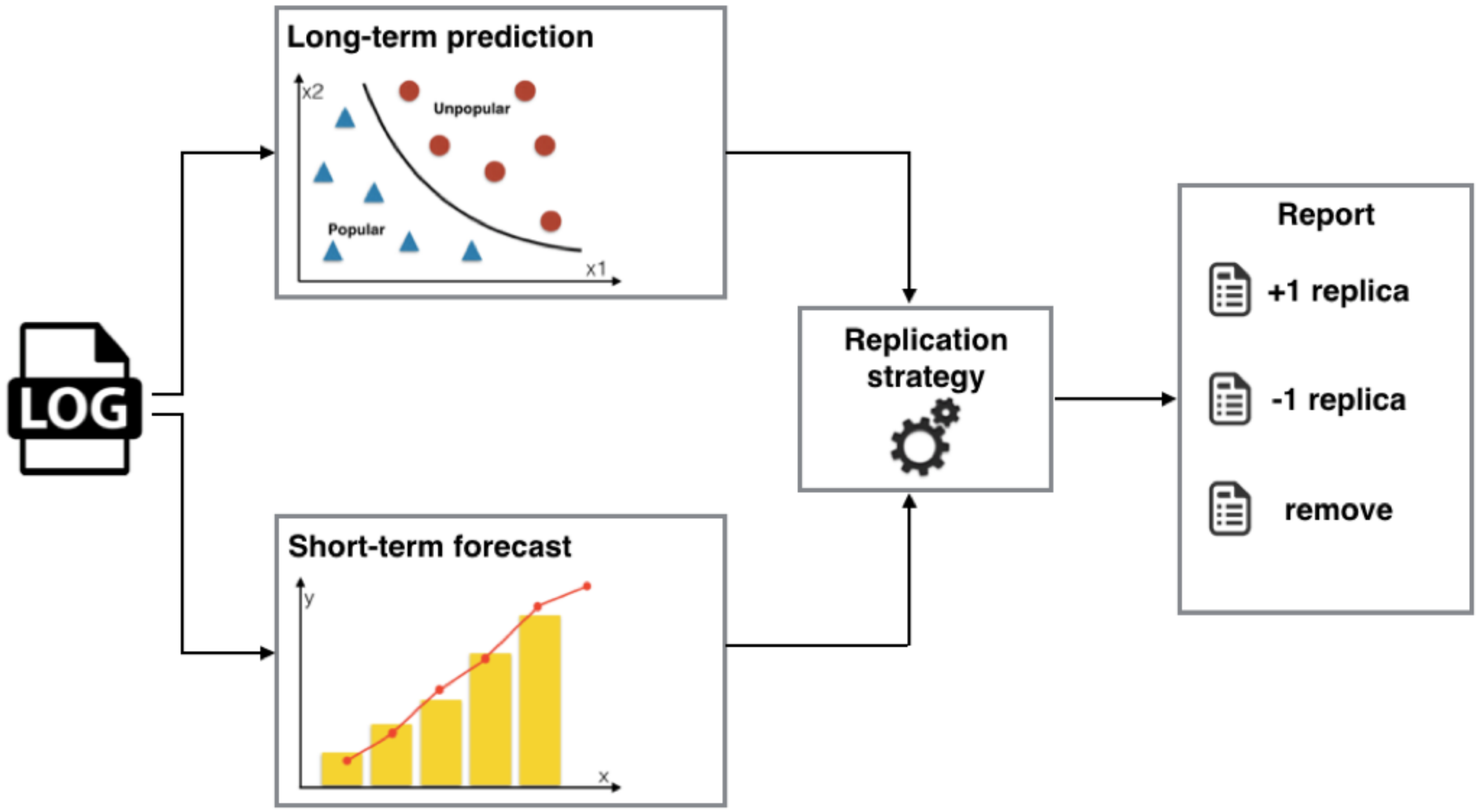
ML in Data Popularity

Problem

- PBs of real data and Monte Carlo are produced every year.
- The data is kept on disk and tape storage systems.
- Disks are faster but are way more expensive.
- Files are stored with several replicas.



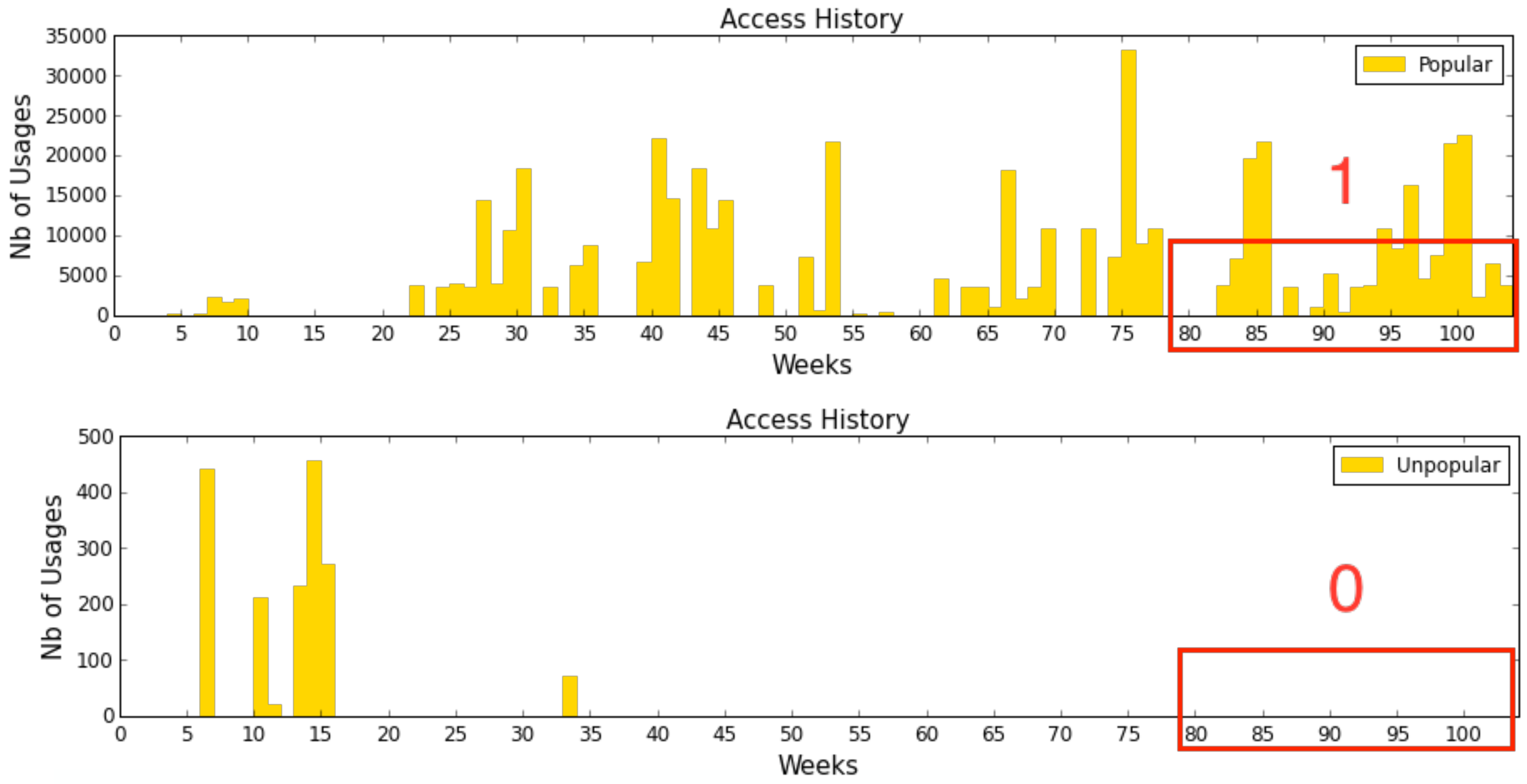
Formulation



- Need 3 algorithms:
  - The dataset popularity prediction (long term)
  - Number of accesses prediction (short term)
  - Optimisation of the data distribution
- We have:
  - access history of the LHCb data storage system for the last two and a half years

Data Popularity Prediction

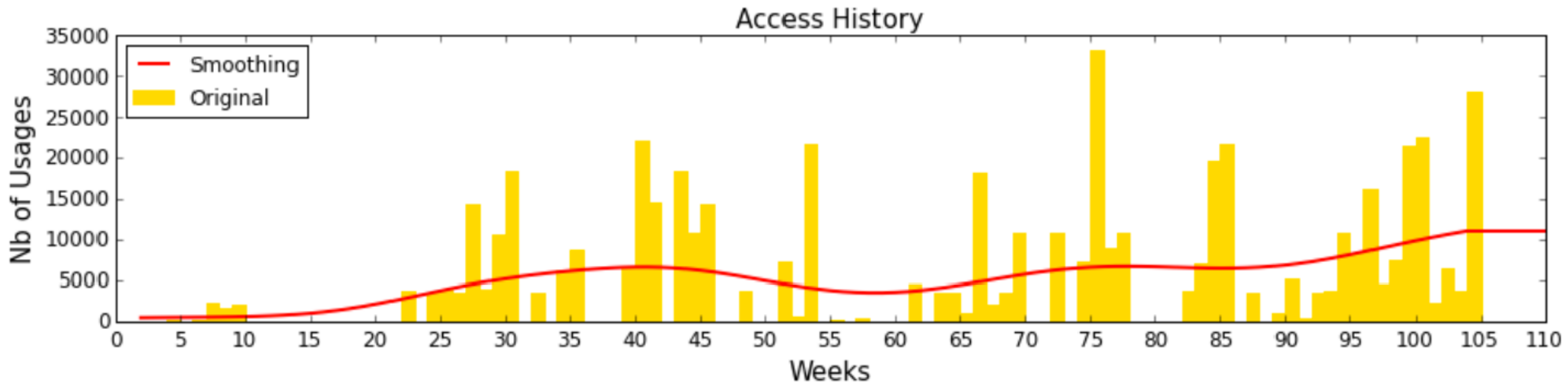
- Train Random forest to predict popular files



- Features:
  - recency, reuse distance, time of the first access, creation time, access frequency, type, extension, size

Probability	Our method		LRU	
	Saved space	Mistakes	Saved space	Mistakes
0.05	32 Tb	9,8 %	32 Tb	22,2 %
0.1	1387 Tb	4,6 %	1372 Tb	5,6 %
0.15	2002 Tb	6,1 %	2002 Tb	6,2 %
0.2	2224 Tb	6,7 %	2223 Tb	7,0 %

Data Distribution



For short-term forecast Brown exponential smoothing used

Based on the predicted_number_of_accesses/ number_of_replicas metric and long-term forecast, we take decision:

- Increase number of replicas.
- Decrease number of replicas.
- Remove from disks.

Min number of replicas	Space can be saved
1	5 152 Tb
2	2 717 Tb
3	616 Tb
4	3 Tb

Realisation

- Server can be cloned from git.
- After some installation procedures, used easily within python script:

```
data_path = 'data/inputs/popularity-910days.csv'
params="{'n_tb':100}"

r = post('https://popcon.cern.yandex.net/', files={'file':open(data_path)}, data={'params':params})
```

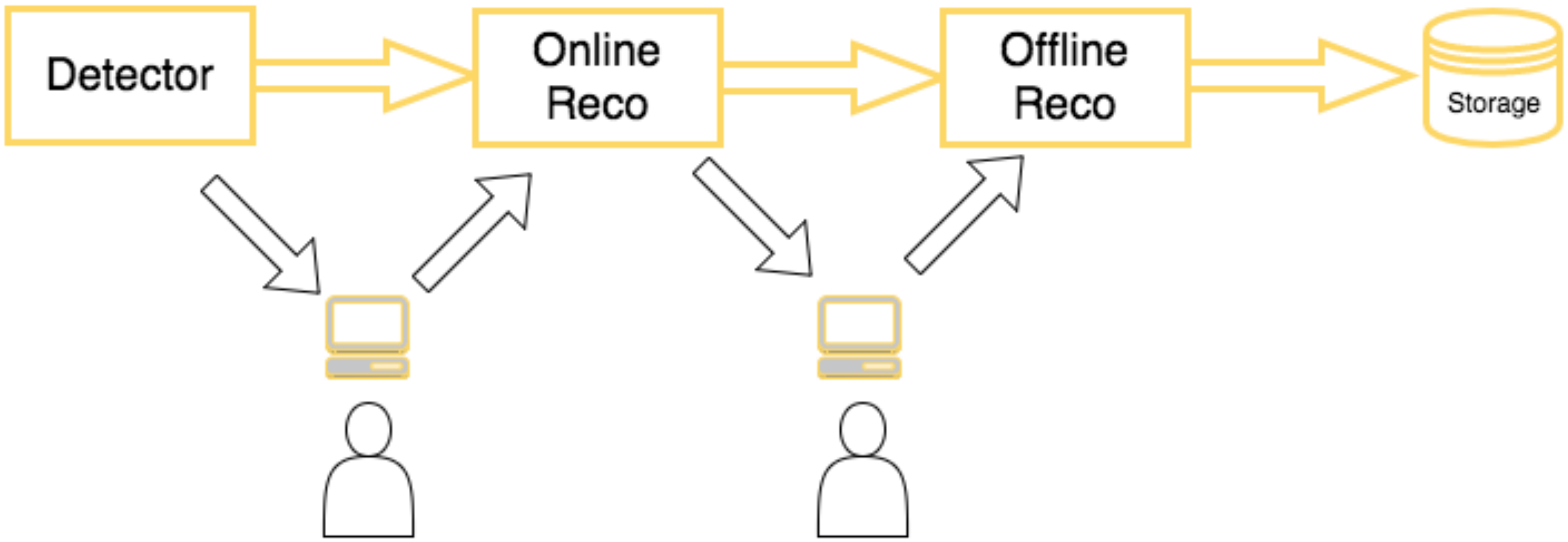
```
# save results
f = open('data/outputs/report.csv', 'w')
f.write(r.content)
```

References

- https://github.com/yandexdataschool/DataPopularity/tree/release_3.0.x
- To be shown at CHEP

ML for Anomaly Detection

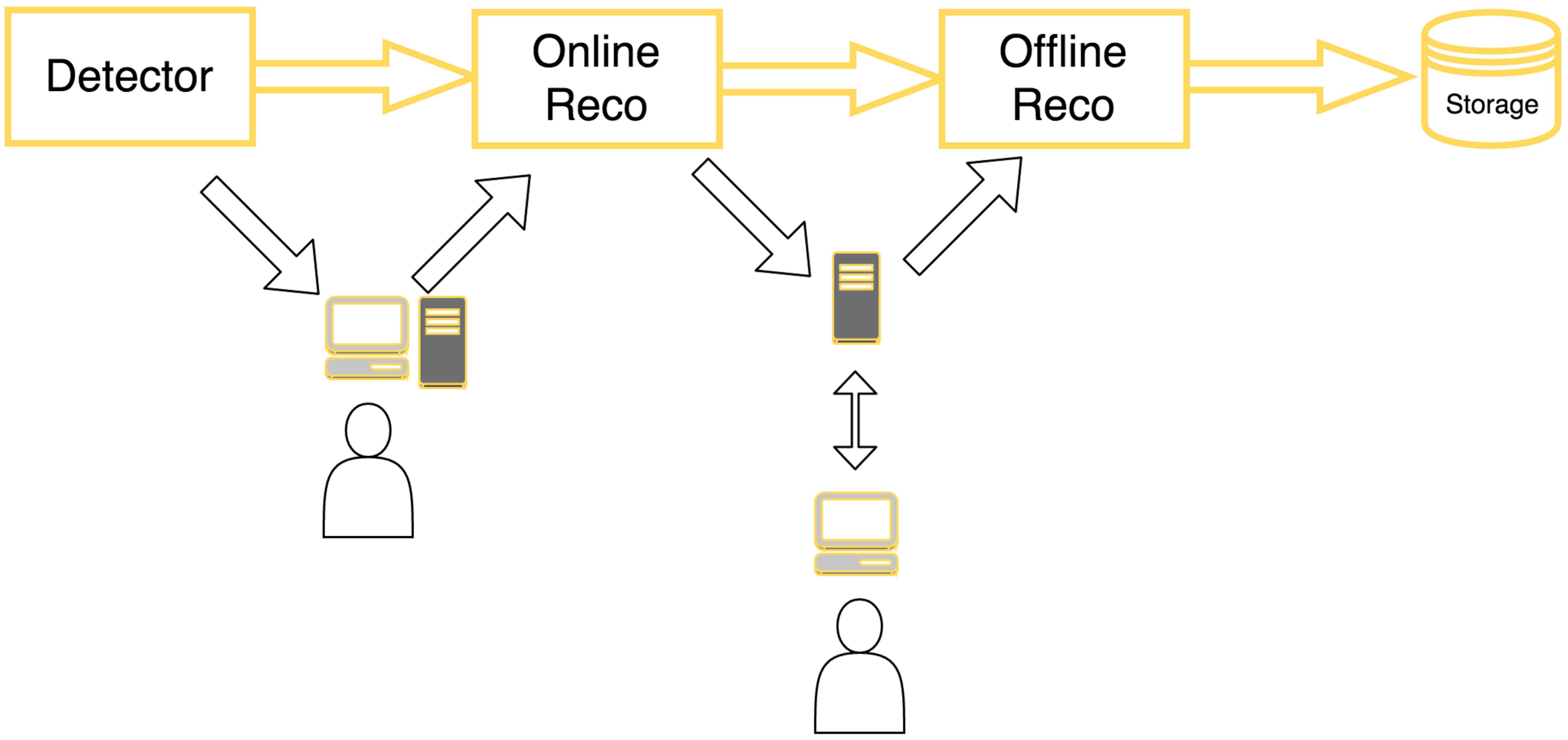
Typical Workflow



- Several people are typically on shifts controlling the flow of data from detector into the storage

Updated Workflow

- The monitoring systems can be updated with:
  - helper, a recommendation system for a shifter
  - solver, automated decision maker
  - both

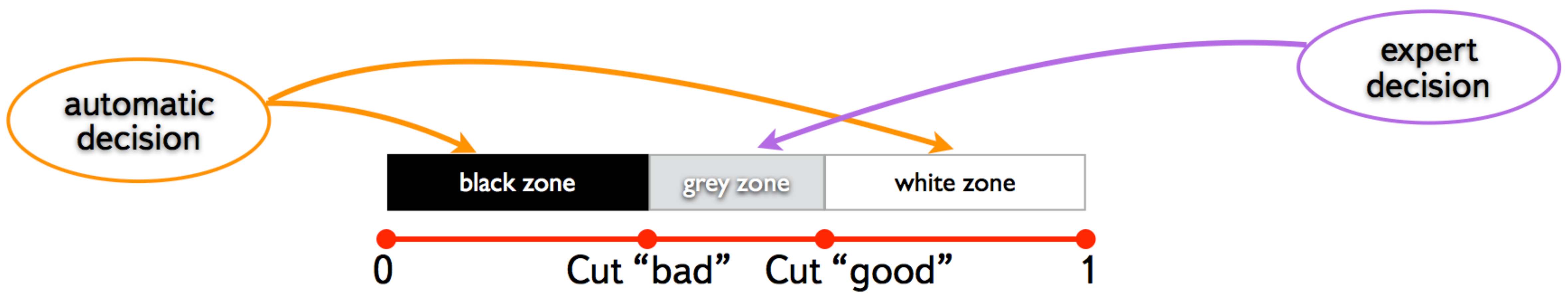


Approaches

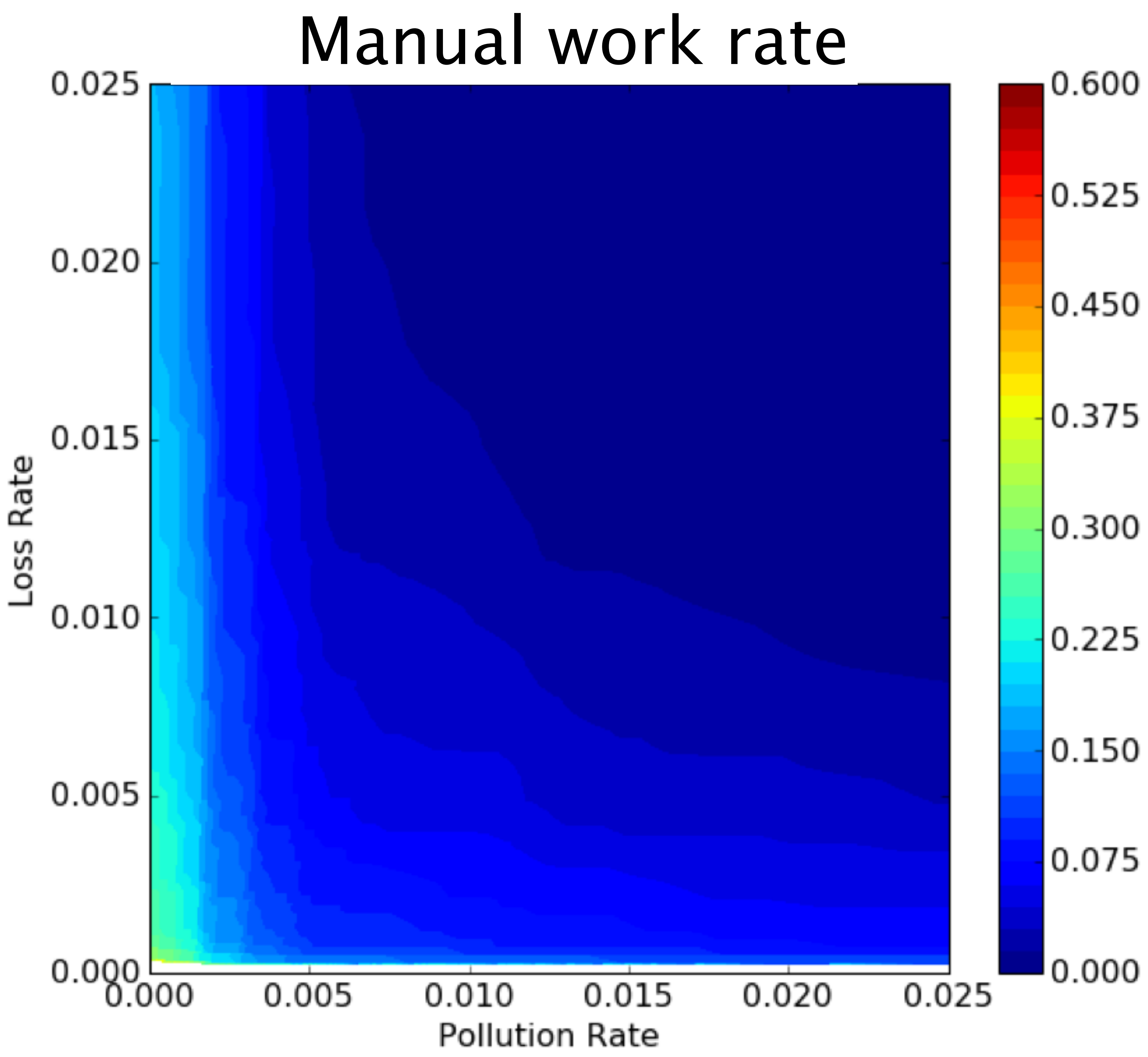
- Two ML approaches are possible in this case:
- Supervised approach
  - uses historical data processed by expert
  - ML algorithm learns the pattern that lead to the experts' decision
  - problem: hard to outperform the expert in quality
- Unsupervised approach
  - use time series to catch changes in data behaviour
  - problem: hard to validate

Supervised Learning

- Problem: CMS Data Certification
- Data: CMS 2010B run open data
- Aim: automated classification of LumiSections as “good” or “bad” using expert opinions on previous runs
- Features: particle flow jets, Calorimeter Jets, Photons, Muons



Supervised Learning

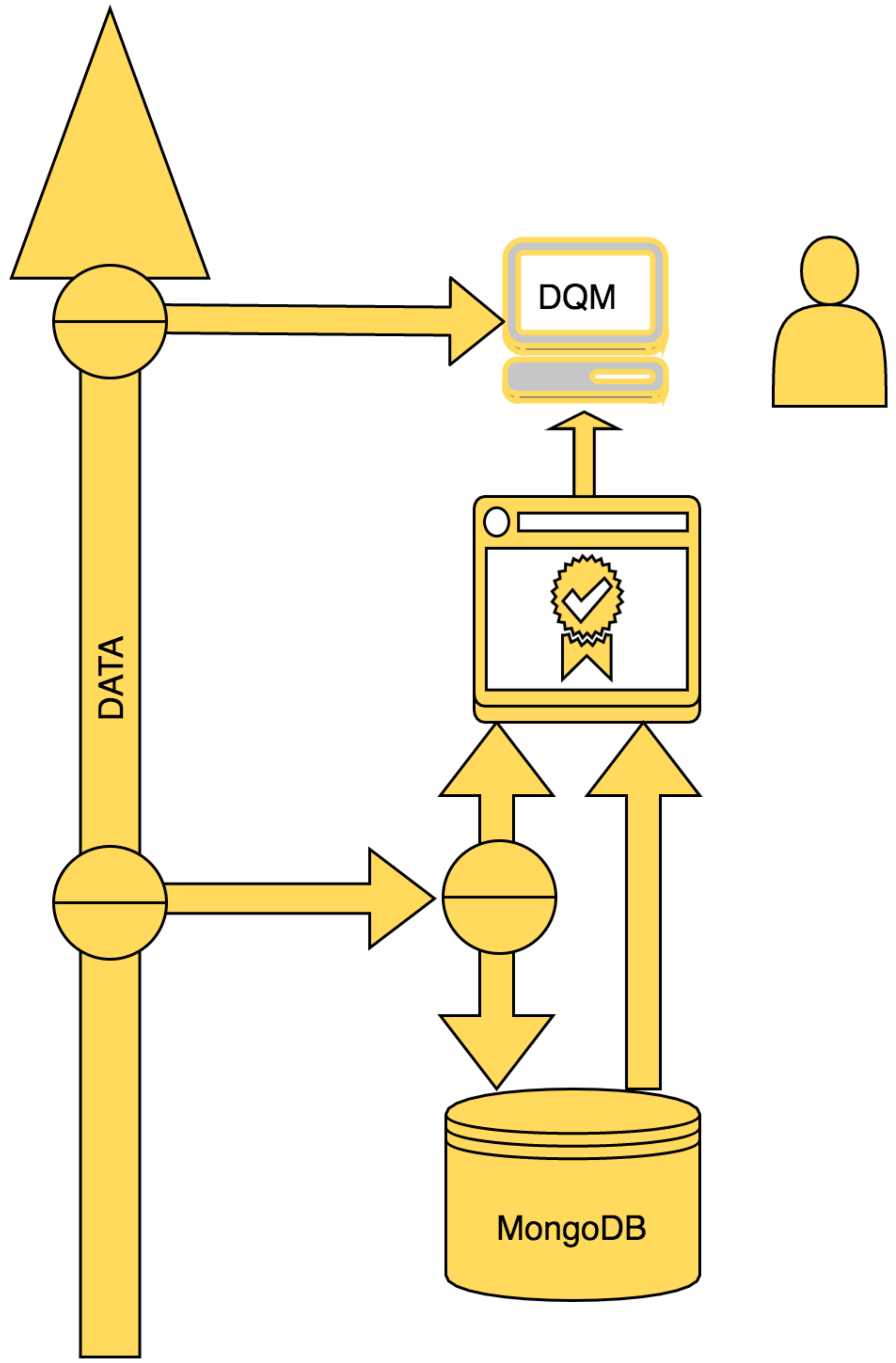


The aim is to minimise the Manual work with low Loss Rate ("good" classified as "bad") and Pollution Rate ("bad" classified as "good").

~90% saving on manual work is feasible for Pollution rate at 5‰

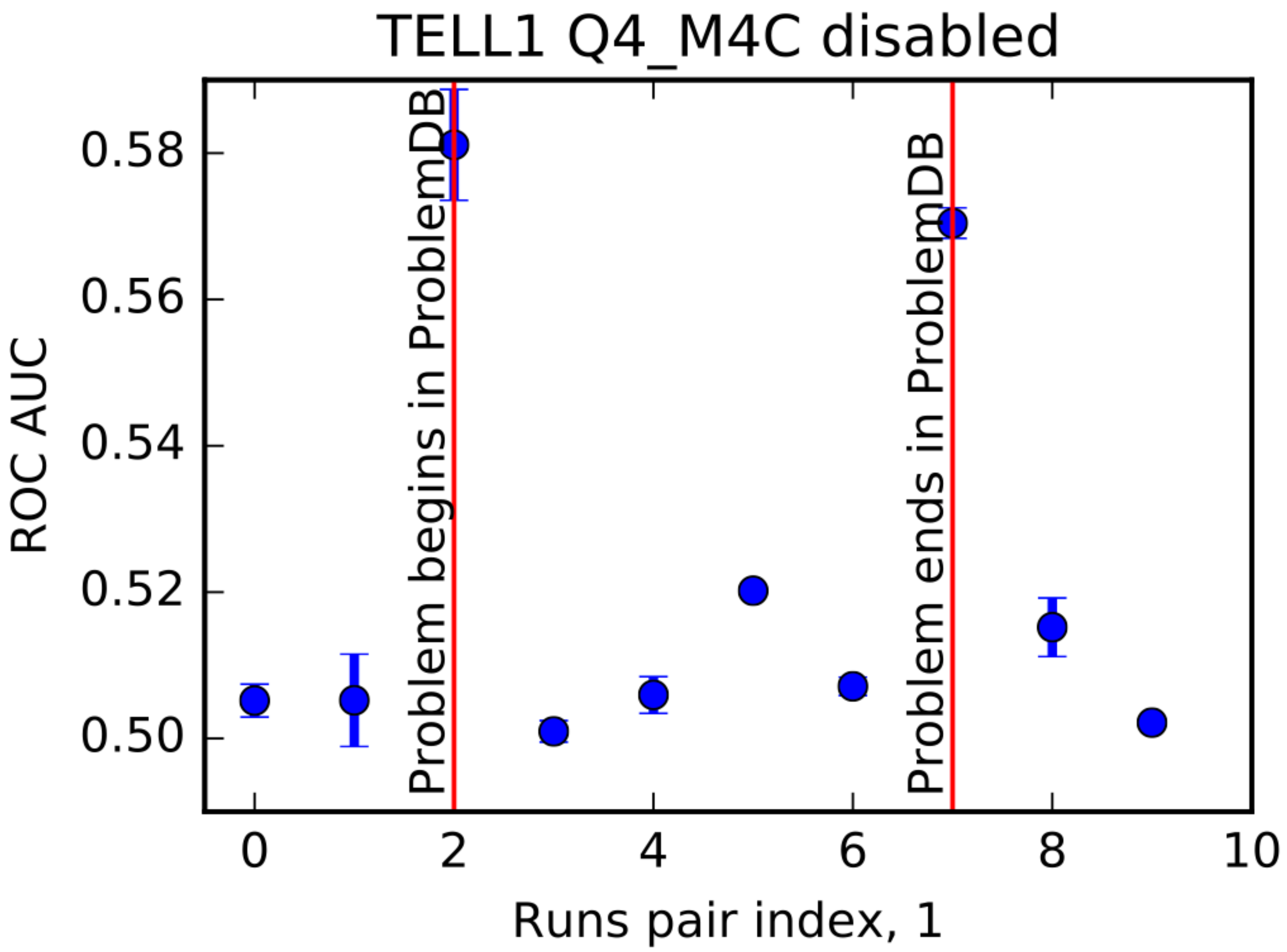
Unsupervised Learning

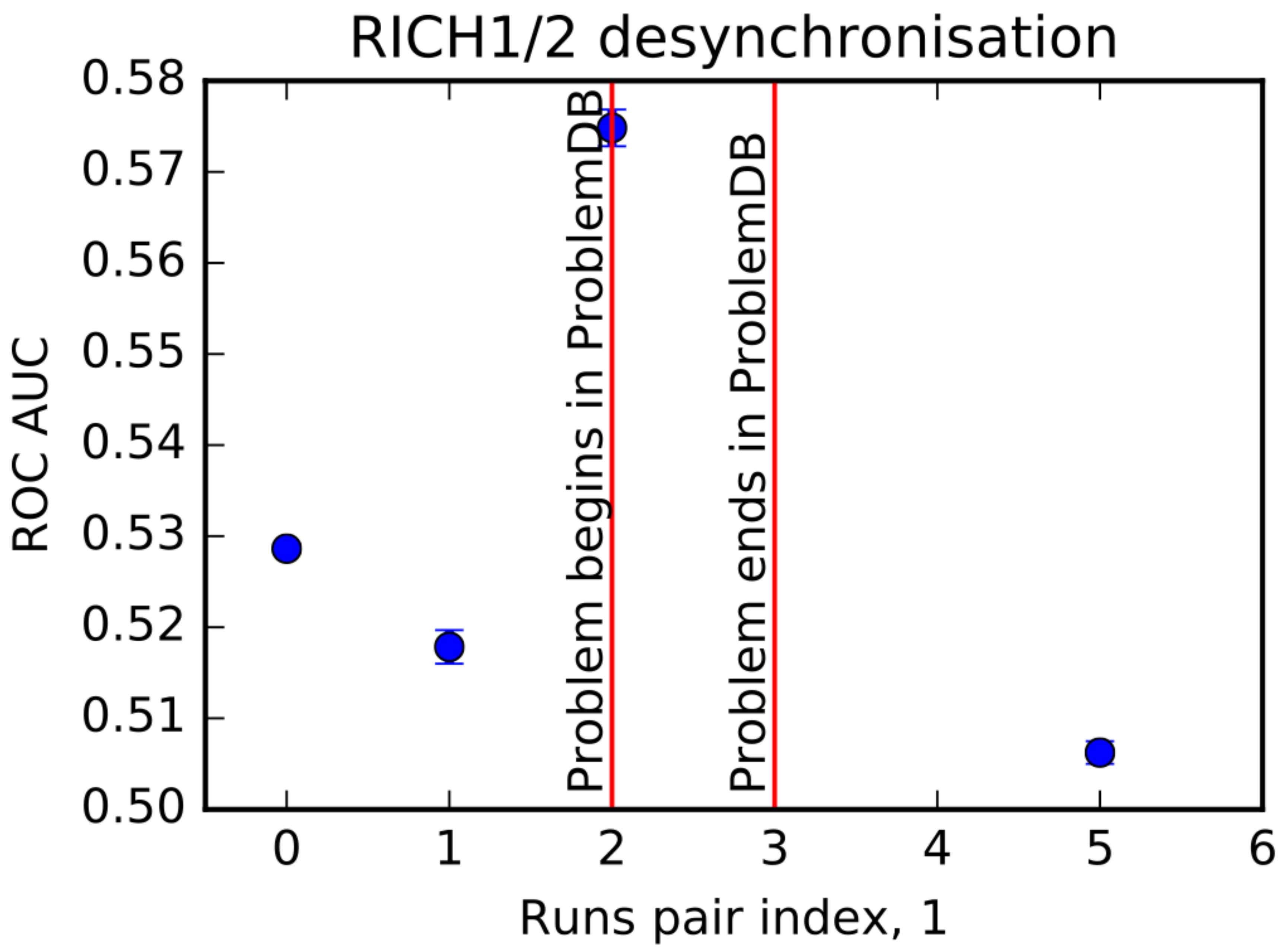
- Problem: LHCb Detector Monitoring
- Data: LHCb trigger streams
- Aim: Identification of problems using previous state of the system
- Features: trigger line decisions, other trigger objects.



Unsupervised Learning

- First attempts look promising





- work is ongoing

References

- http://arxiv.org/abs/1510.00132
- https://github.com/yandexdataschool/cms-dqm
- F. Ratnikov @ DSHEP https://indico.hep.caltech.edu/indico/conferenceOtherViews.py?confId=102&view=standard

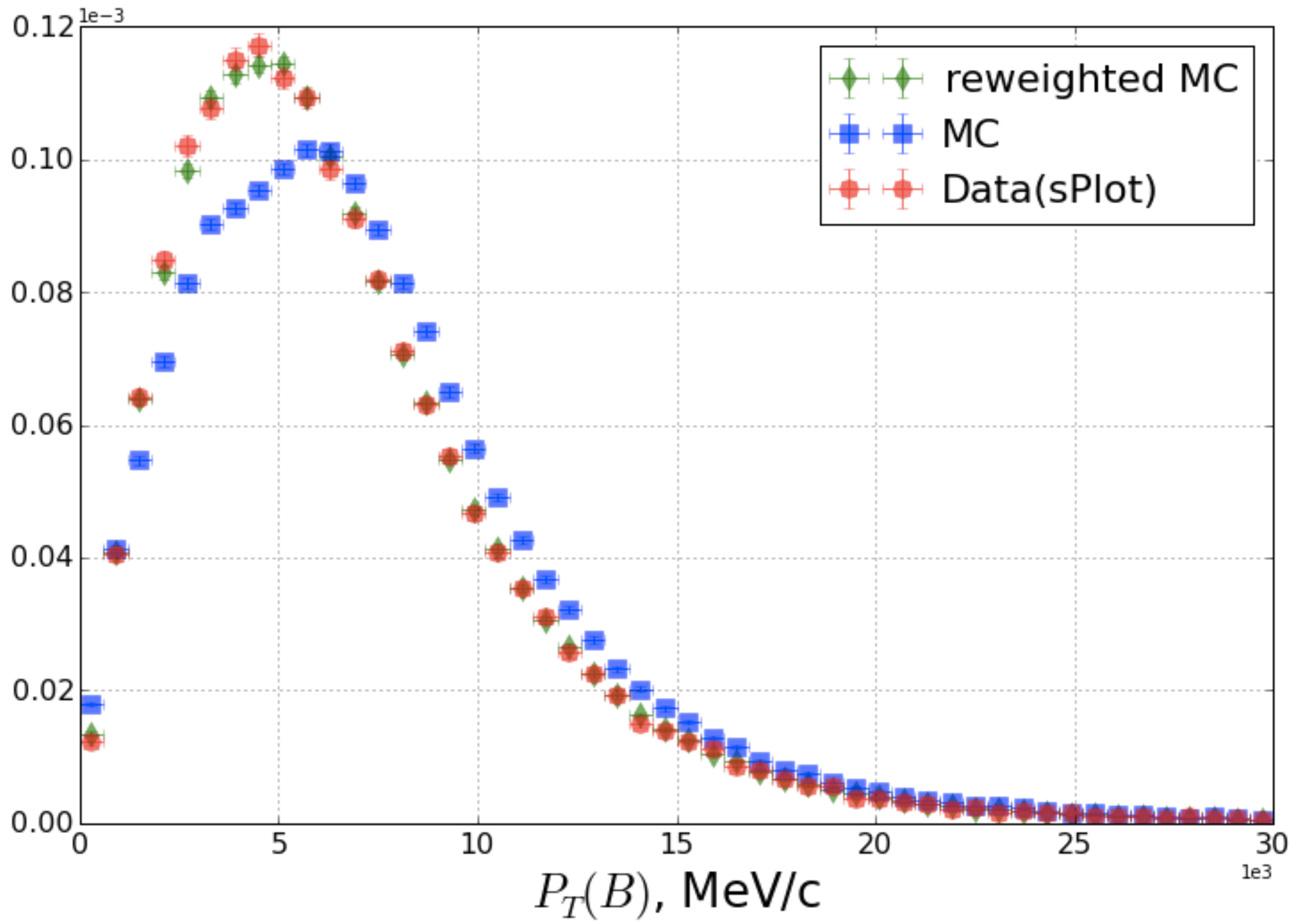
Reweighting problem in HEP

Data/MC disagreement

- Monte Carlo (MC) simulated samples are used for training and tuning a model
- After, trained model is applied to real data (RD)
- Real data and Monte Carlo have different distributions
- Thus, trained model is biased (and the quality is overestimated on MC samples)

Distributions reweighting

- Reweighting in HEP is used to minimize the difference between RD and MC samples
- The goal of reweighting: assign weights to MC s.t. MC and RD distributions coincide
- Known process is used, for which RD can be obtained (MC samples are also available)
- MC distribution is original, RD distribution is target



Typical approach: histogram reweighting

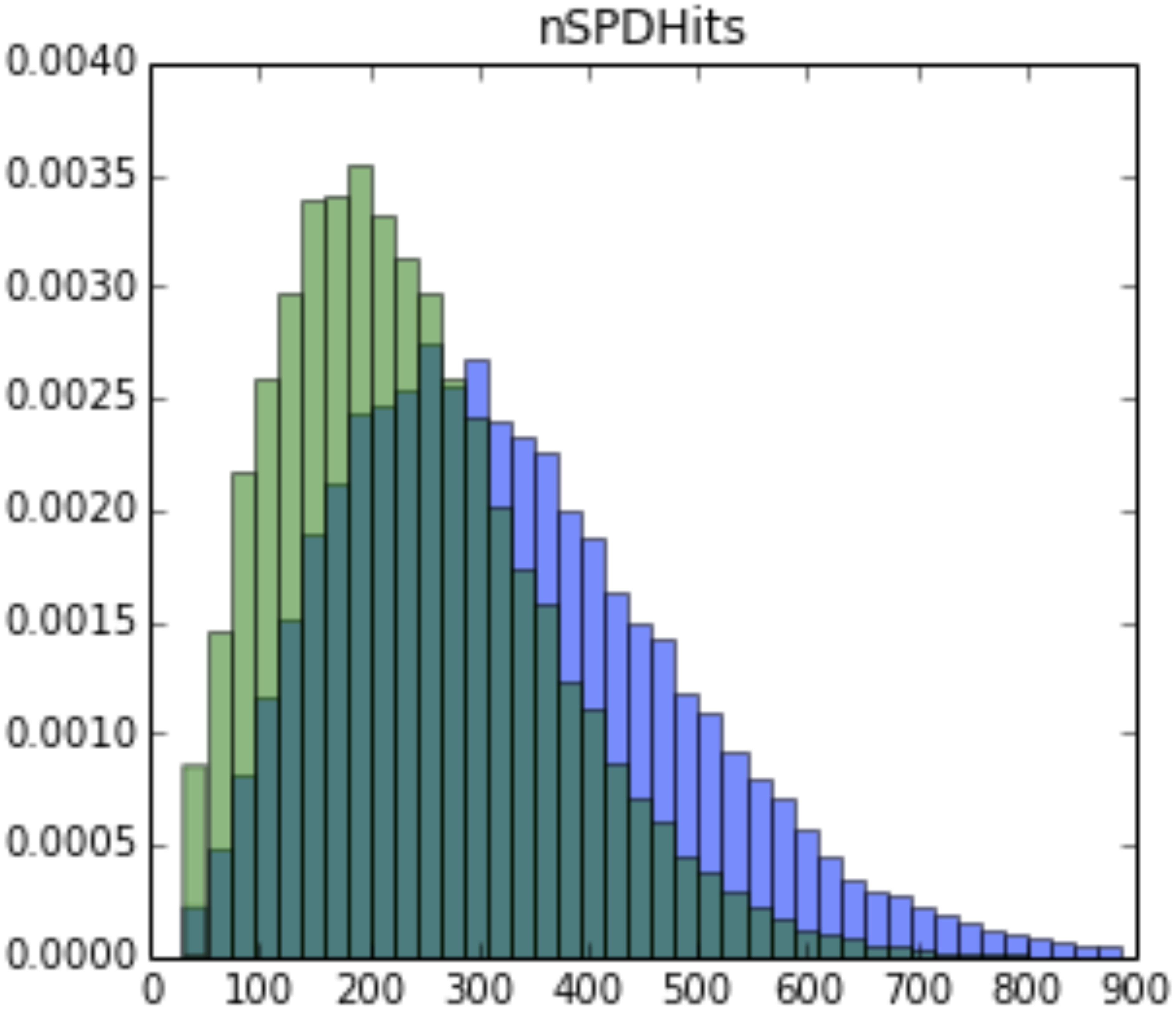
- variable(s) is split into bins
- in each bin the MC weight is multiplied by:

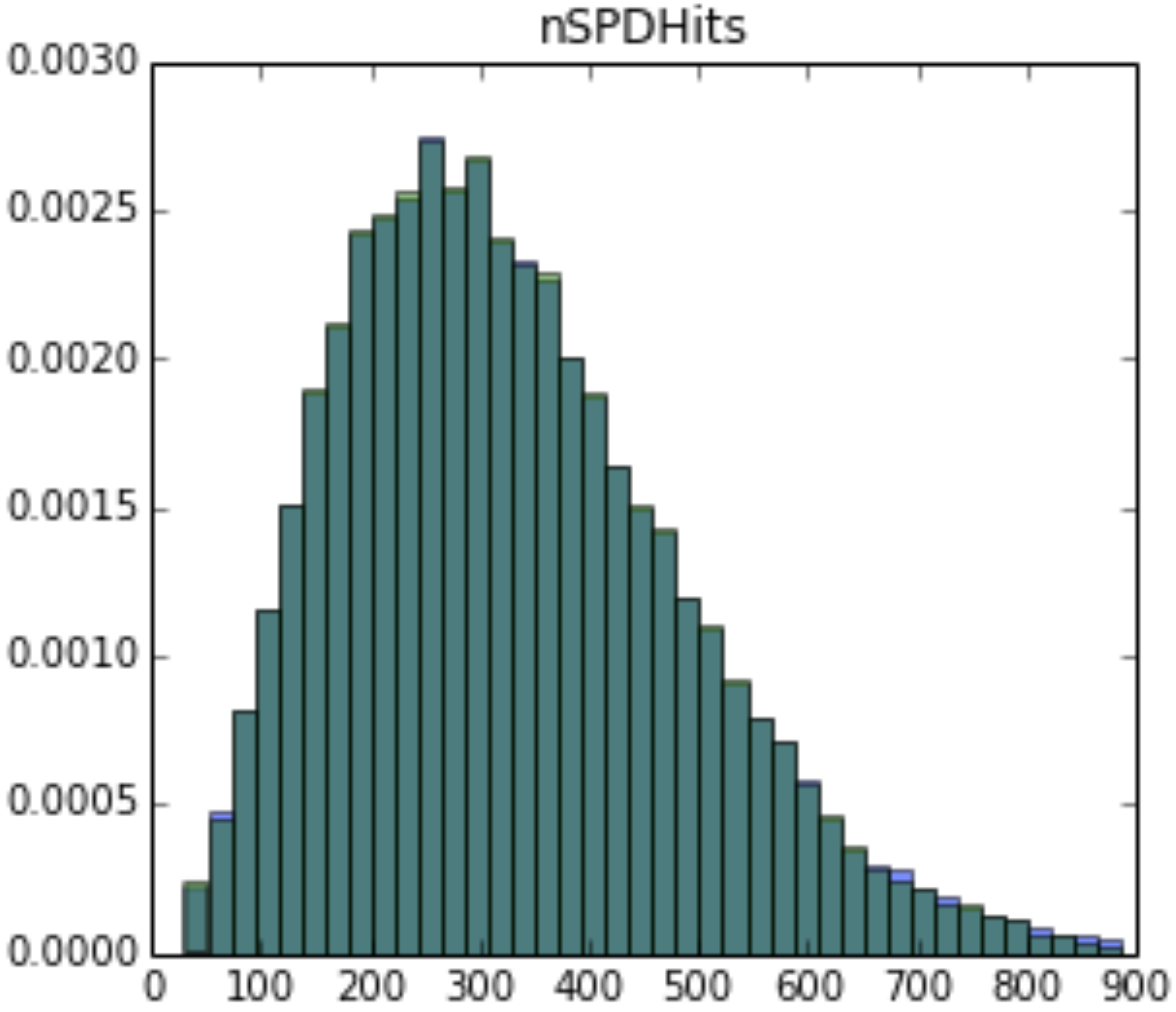
$$\text{multiplier}_{\text{bin}} = \frac{w_{\text{bin, target}}}{w_{\text{bin, original}}}$$

$w_{\text{bin, target}}$, $w_{\text{bin, original}}$ – total weights of events in a bin for target and original distributions

1. simple and fast
2. number of variables is very limited by statistics (typically only one, two)
3. reweighting in one variable may bring disagreement in others
4. which variable is preferable for reweighting?

Typical approach: example

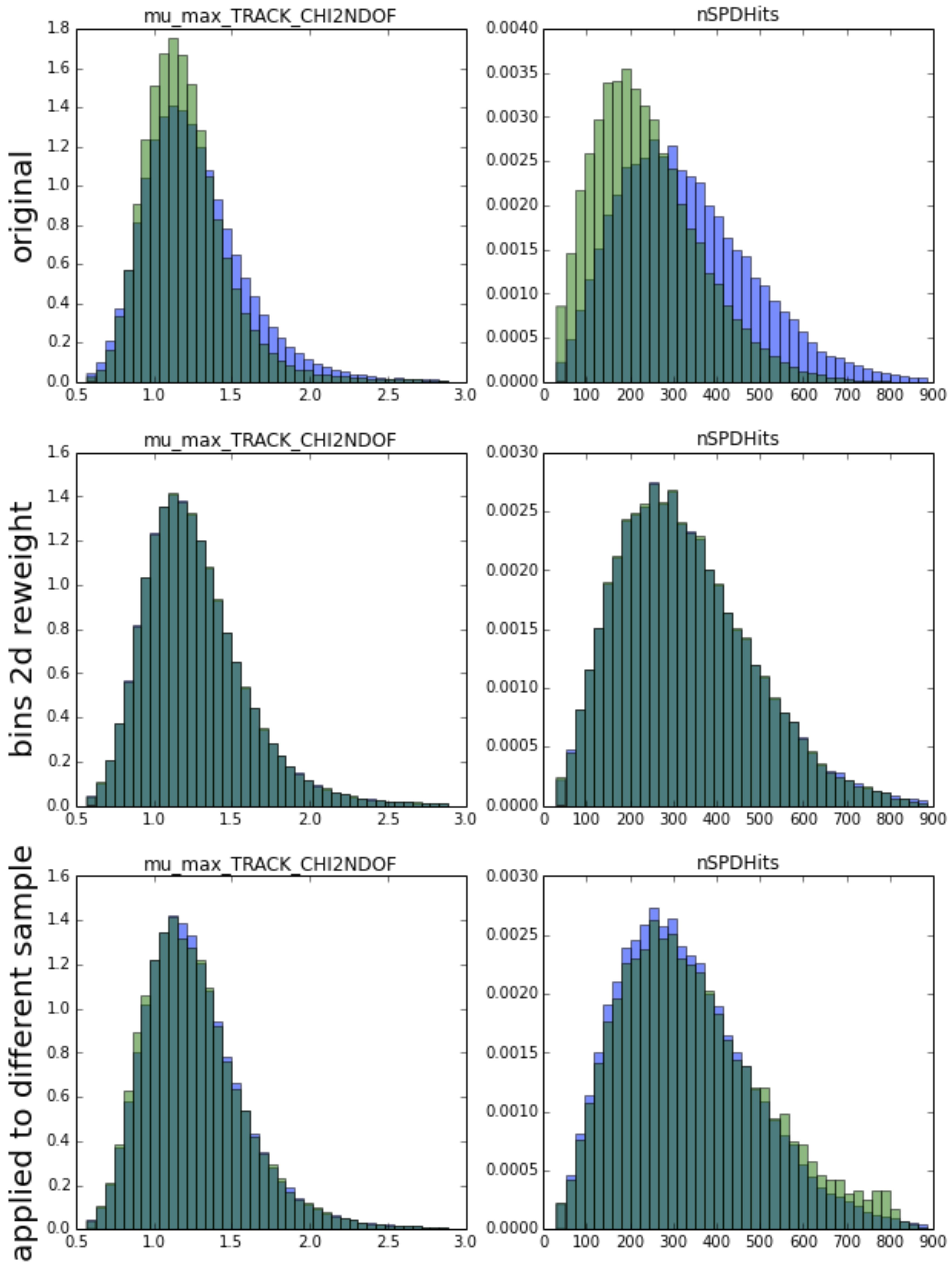




Typical approach: example

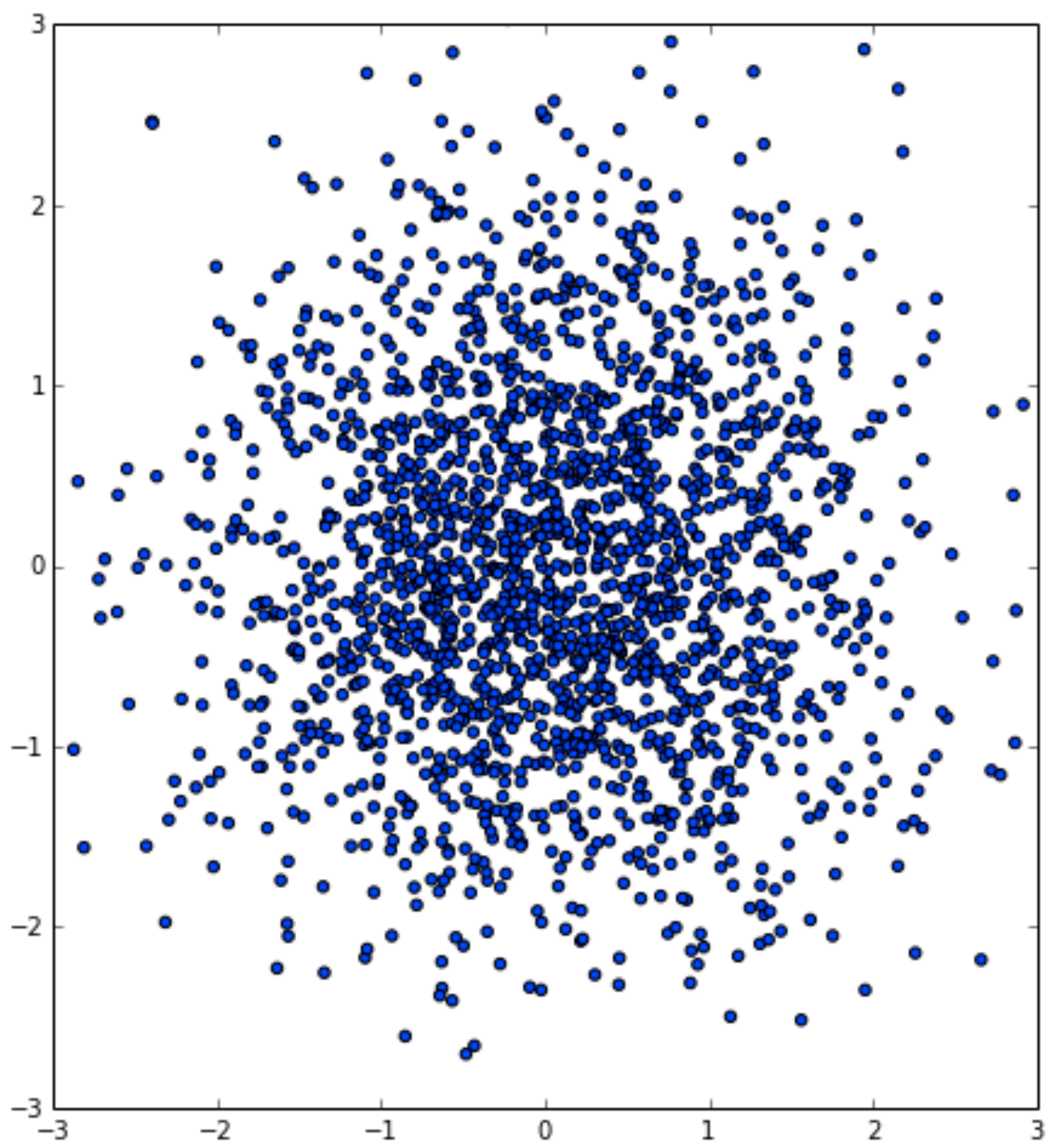
- Problems arise when there are too few events in a bin
- This can be detected on a **holdout** (see the latest row)
- Issues:
    1. few bins – rule is rough
    2. many bins – rule is not reliable

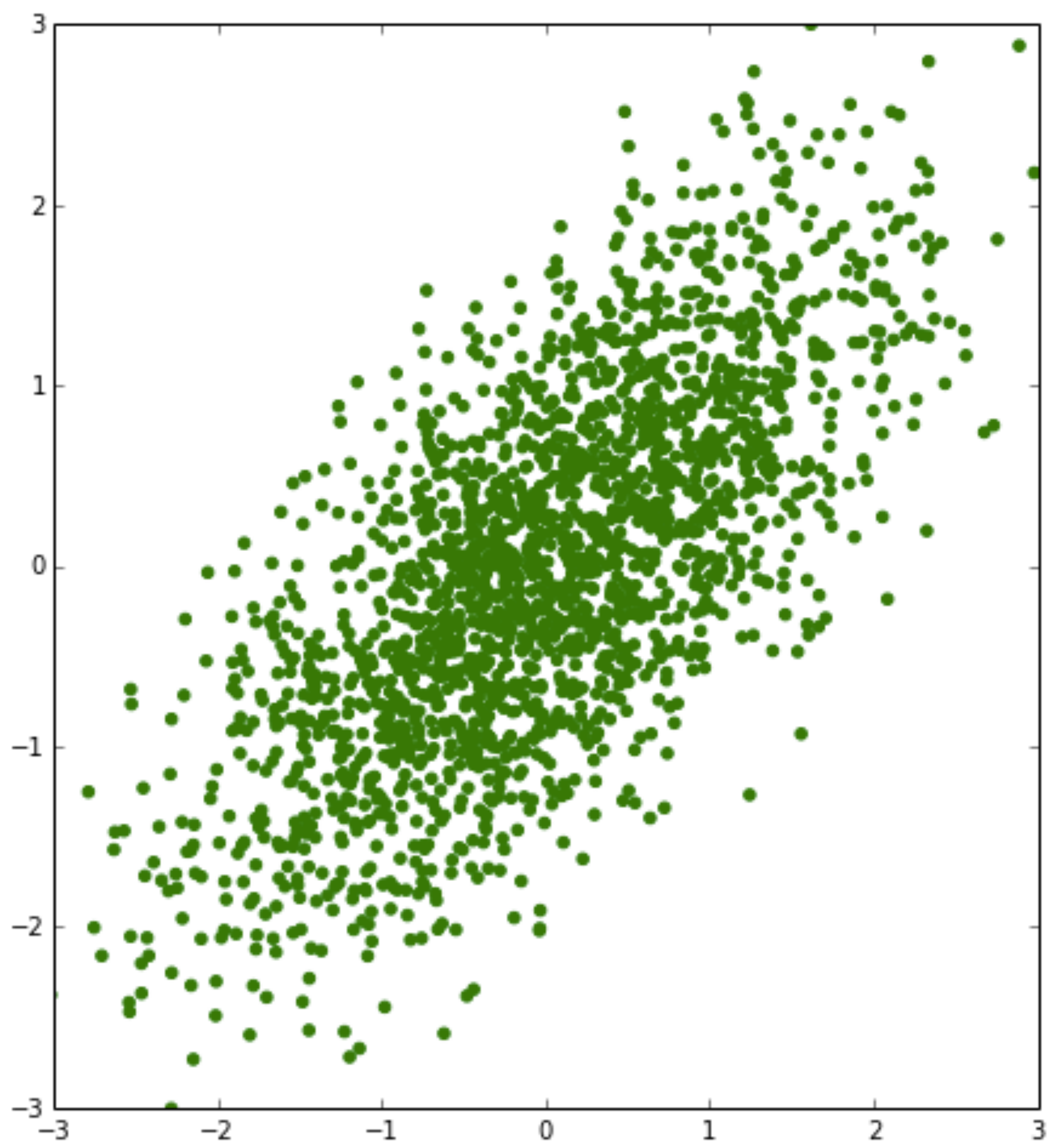
Reweighting rule must be checked on a holdout!



Reweighting quality

- How to check the quality of reweighting?
- One dimensional case: two samples tests (Kolmogorov-Smirnov test, Mann-Whitney test, ...)
- Two or more dimensions?
- Comparing 1d projections is not a way





Comparing nDim distributions using ML

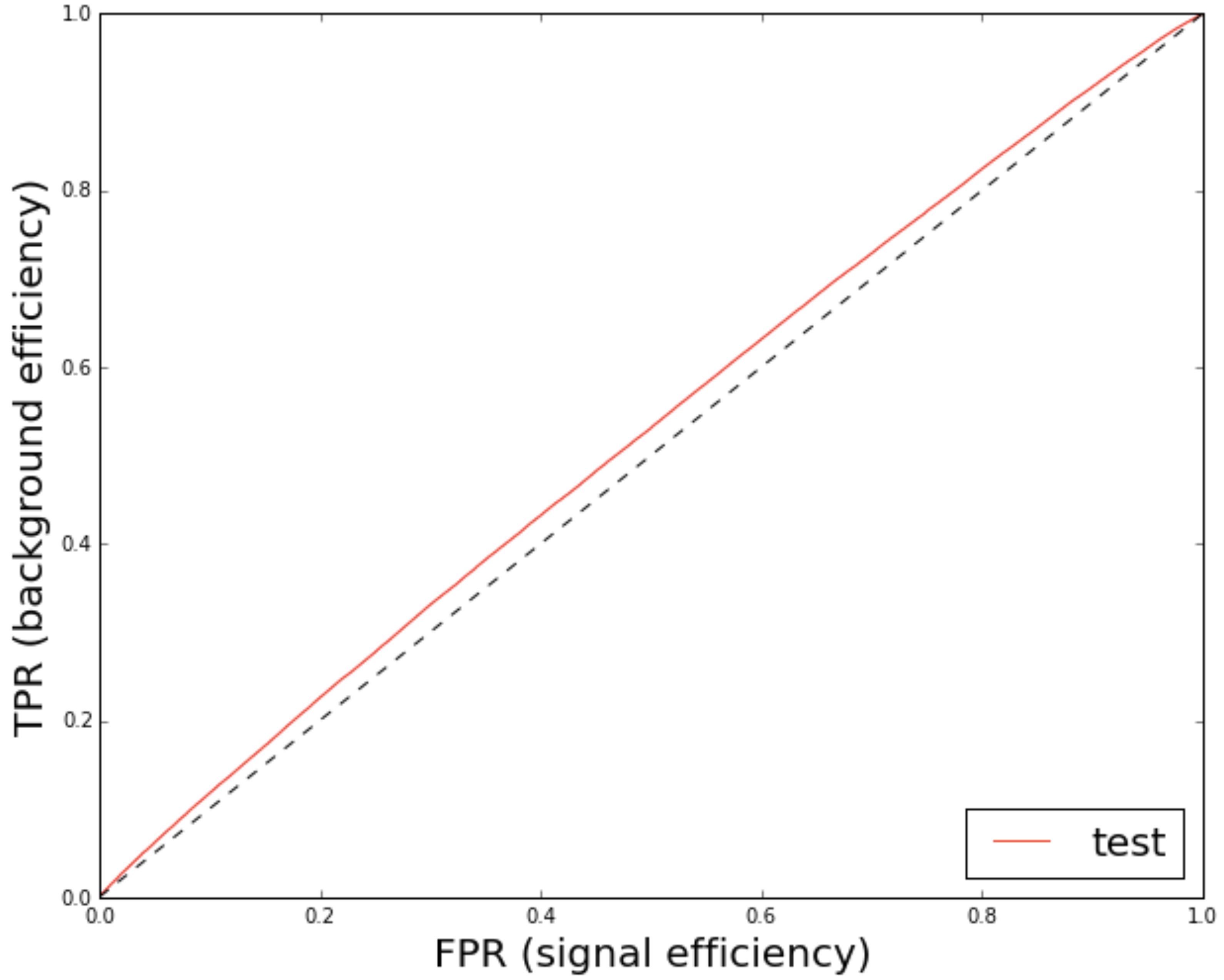
- Final goal: classifier doesn't use data/MC disagreement information = classifier cannot discriminate data and MC

- Comparison of distributions shall be done using ML:

  - train a classifier to discriminate data and MC

  - output of the classifier is one-dimensional variable

  - looking at the ROC curve (alternative of two sample test) on a holdout
    (should be 0.5 if the classifier cannot discriminate data and MC)



Density ratio estimation approach

- We need to estimate density ratio: $\frac{f_{RD}(x)}{f_{MC}(x)}$
- Classifier trained to discriminate MC and RD should reconstruct probabilities $p_{MC}(x)$ and $p_{RD}(x)$
- For reweighting we can use $\frac{f_{RD}(x)}{f_{MC}(x)} \sim \frac{p_{RD}(x)}{p_{MC}(x)}$

1. Approach is able to reweight in many variables
2. It is successfully tried in HEP, see D. Martschei et al, "Advanced event reweighting using multivariate analysis", 2012
3. There is poor reconstruction when ratio is too small / high
4. It is slower than histogram approach

Way to Succeed

- Write ML algorithm to solve directly reweighting problem
- Remind that in histogram approach few bins is bad, many bins is bad too.
- What can we do?
- Better idea…
  - Split space of variables in several large regions
  - Find this regions ‘intellectually’

Decision tree for reweighting

Write ML algorithm to solve directly reweighting problem:

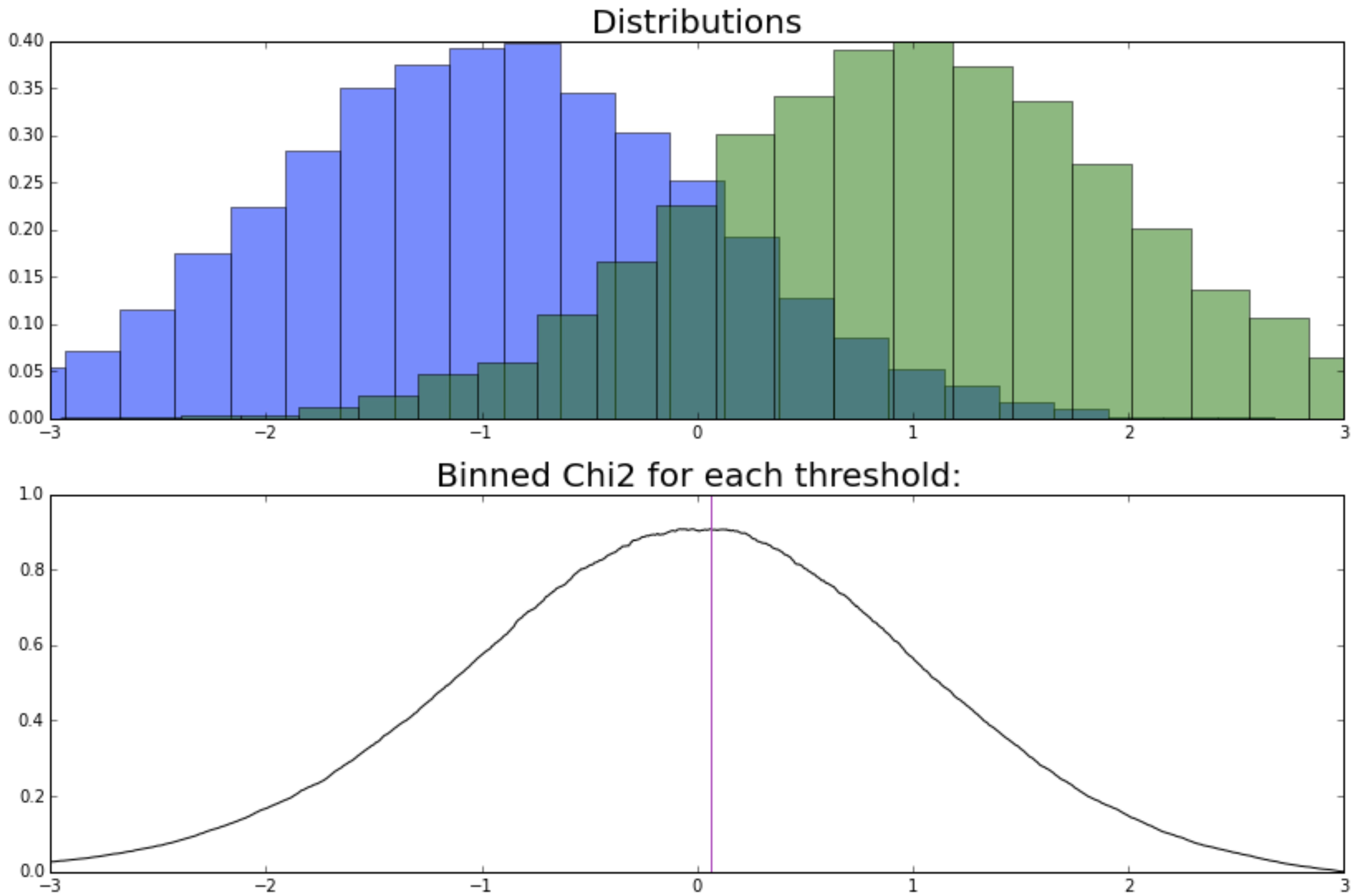
- Tree splits the space of variables with orthogonal cuts (each tree leaf is a region, or bin)
- There are different criteria to construct a tree (MSE, Gini index, entropy, …)
- Find regions with the highest difference between original and target distribution

Spitting criteria

Finding regions with high difference between original and target distribution by maximizing symmetrized:

$$\chi^2 = \sum_{leaf} \frac{(w_{\text{leaf, original}} - w_{\text{leaf, target}})^2}{w_{\text{leaf, original}} + w_{\text{leaf, target}}}$$

A tree leaf may be considered as 'a bin'; $w_{\text{leaf, original}}, w_{\text{leaf, target}}$ – total weights of events in a leaf for target and original distributions.



BDT reweighter

Many times repeat the following steps:

- build a shallow tree to maximize symmetrized χ^2
- compute predictions in leaves:

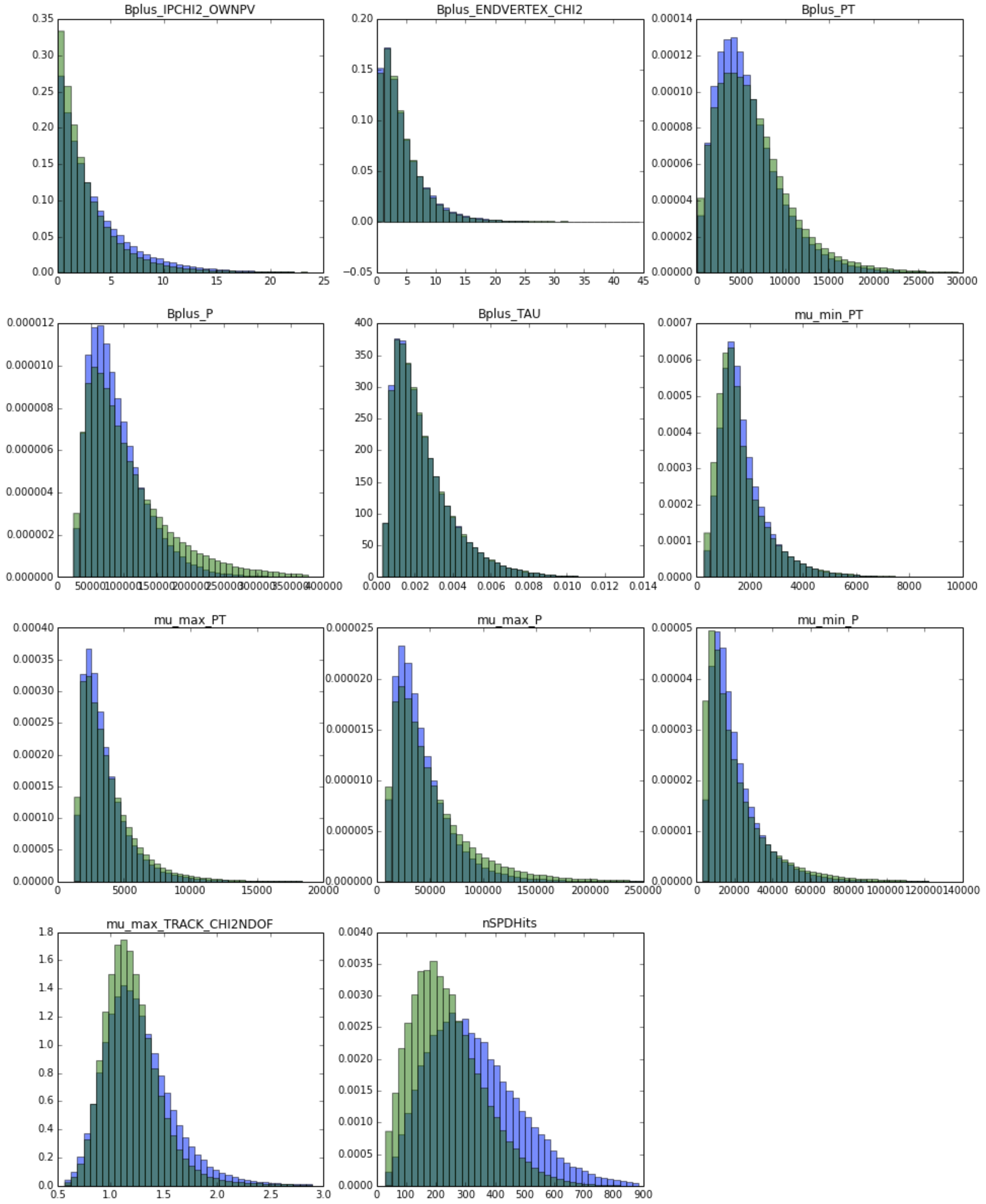
$$\text{leaf\_pred} = \log \frac{w_{\text{leaf, target}}}{w_{\text{leaf, original}}}$$

- reweight distributions:

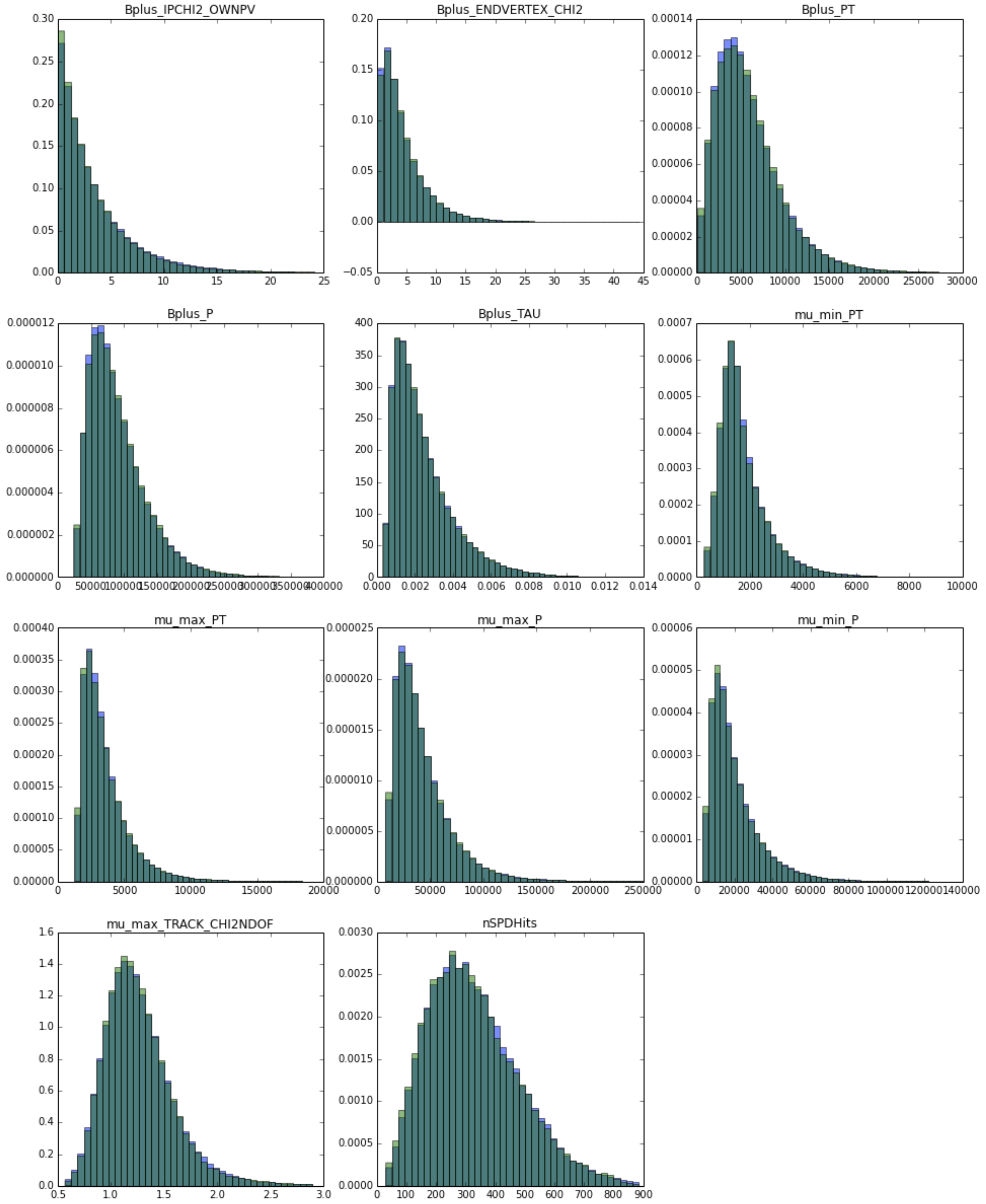
$$w = \begin{cases} w, & \text{if event from target (RD) distribution} \\ w \cdot e^{\text{pred}}, & \text{if event from original (MC) distribution} \end{cases}$$

BDT reweighter DEMO

before BDT reweighting



after BDT reweighting

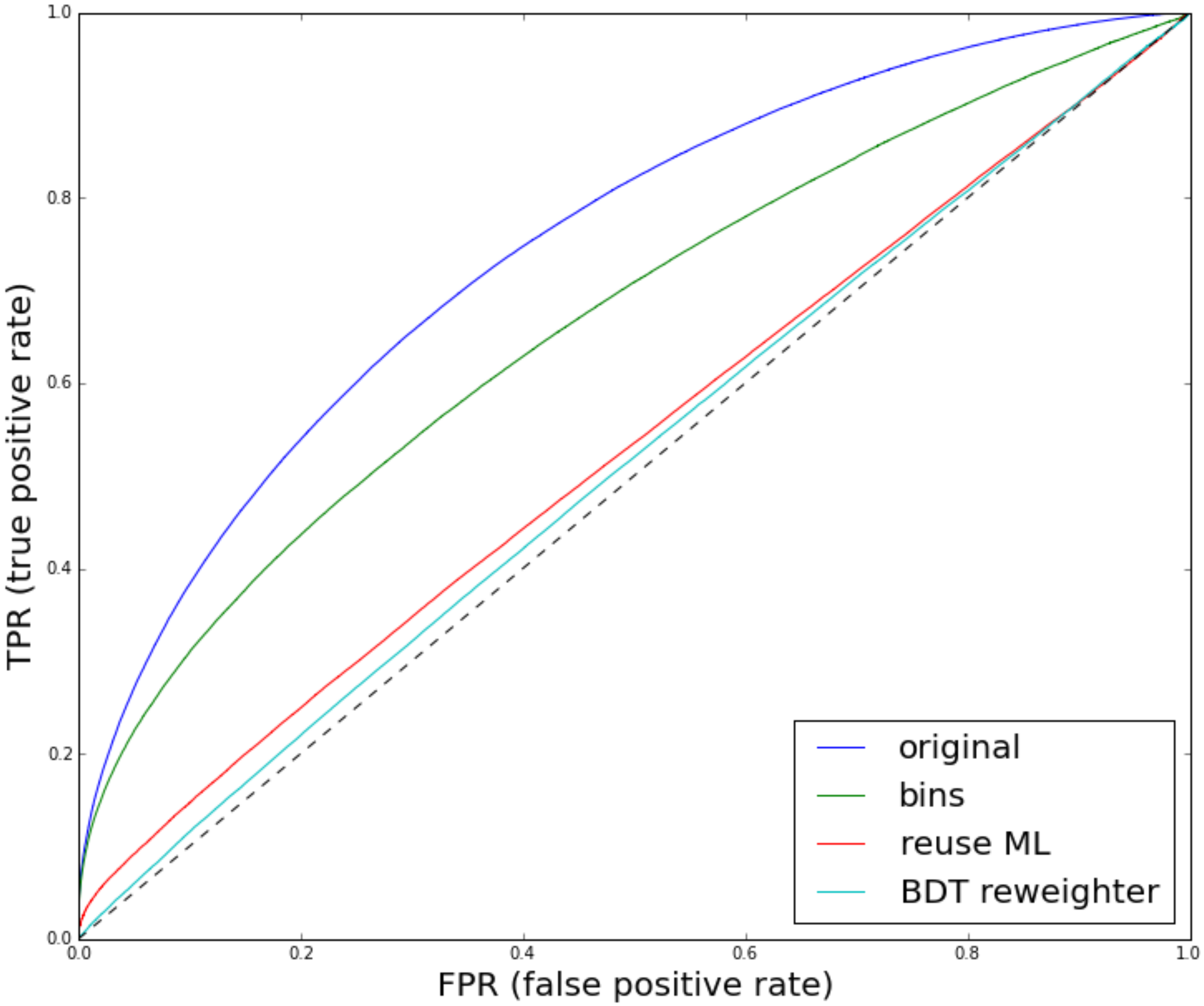


Kolmogorov-Smirnov distance for 1d projections

Bins reweighter uses only 2 last variables (60×60 bins); BDT reweighter uses all variables

	KS original	KS bins reweight	KS GB reweight
Feature			
Bplus_IPCHI2_OWNPV	0.080	0.064	0.003
Bplus_ENDVERTEX_CHI2	0.010	0.019	0.002
Bplus_PT	0.060	0.069	0.004
Bplus_P	0.111	0.115	0.005
Bplus_TAU	0.005	0.005	0.003
mu_min_PT	0.062	0.061	0.004
mu_max_PT	0.048	0.056	0.003
mu_max_P	0.093	0.098	0.004
mu_min_P	0.084	0.085	0.004
mu_max_TRACK_CHI2NDOF	0.097	0.006	0.005
nSPDHits	0.249	0.009	0.005

Comparing reweighting with ML



hep_ml library

```
from hep_ml.reweight import GBReweighter
gb = GBReweighter()
gb.fit(mc_data, real_data, target_weight=real_data_sweights)
gb.predict_weights(mc_other_channel)
```

Being a variation of GBDT, BDT reweighter is able to calculate feature importances. Two features used in reweighting with bins are indeed the most important.

	importance
feature	
mu_max_TRACK_CHI2NDOF	0.240272
nSPDHits	0.209090
Bplus_P	0.122314
mu_min_P	0.115245
Bplus_PT	0.080641
Bplus_IPCHI2_OWNPV	0.068209
mu_max_P	0.060518
mu_max_PT	0.037863
mu_min_PT	0.037761
Bplus_ENDVERTEX_CHI2	0.026598
Bplus_TAU	0.001489

Summary

1. Comparison of multidimensional distributions is ML problem
2. Reweighting of distributions is ML problem
3. Check reweighting rule on the holdout

BDT reweighter

- uses each time few large bins (construction is done intellectually)
- is able to handle many variables
- requires less data (for the same performance)
- ... but slow (being ML algorithm)

References

- https://arxiv.org/abs/1608.05806
- http://arogozhnikov.github.io/2015/10/09/gradient-boosted-reweighter.html
- https://arogozhnikov.github.io/hep_ml/

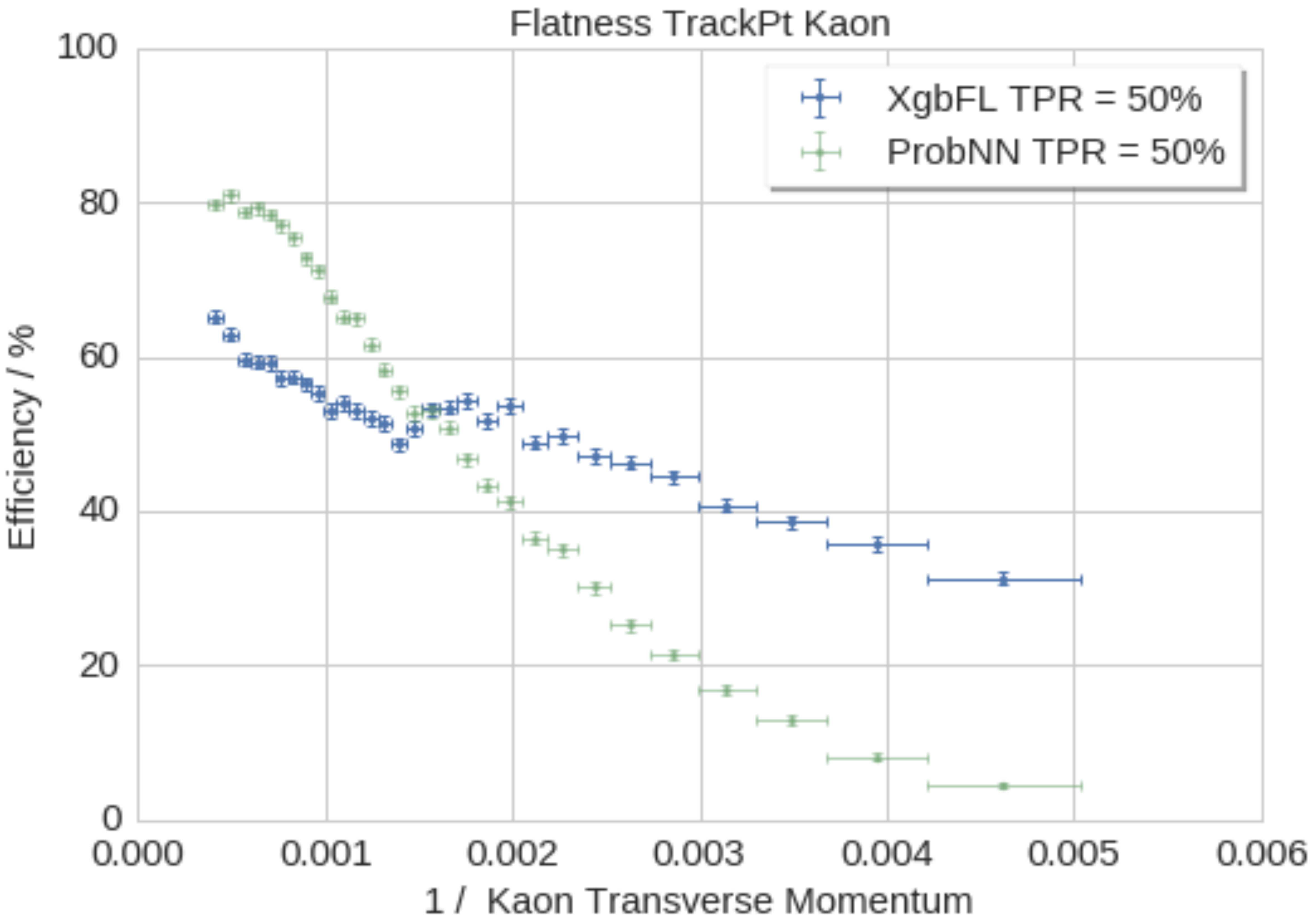
Boosting to uniformity

Uniformity

Uniformity means that we have constant efficiency (FPR/TPR) against some variable.

Applications:

- trigger system (flight time)
  flat signal efficiency
- particle identification (momentum)
  flat signal efficiency
- rare decays (mass)
  flat background efficiency
- Dalitz analysis (Dalitz variables)
  flat signal efficiency

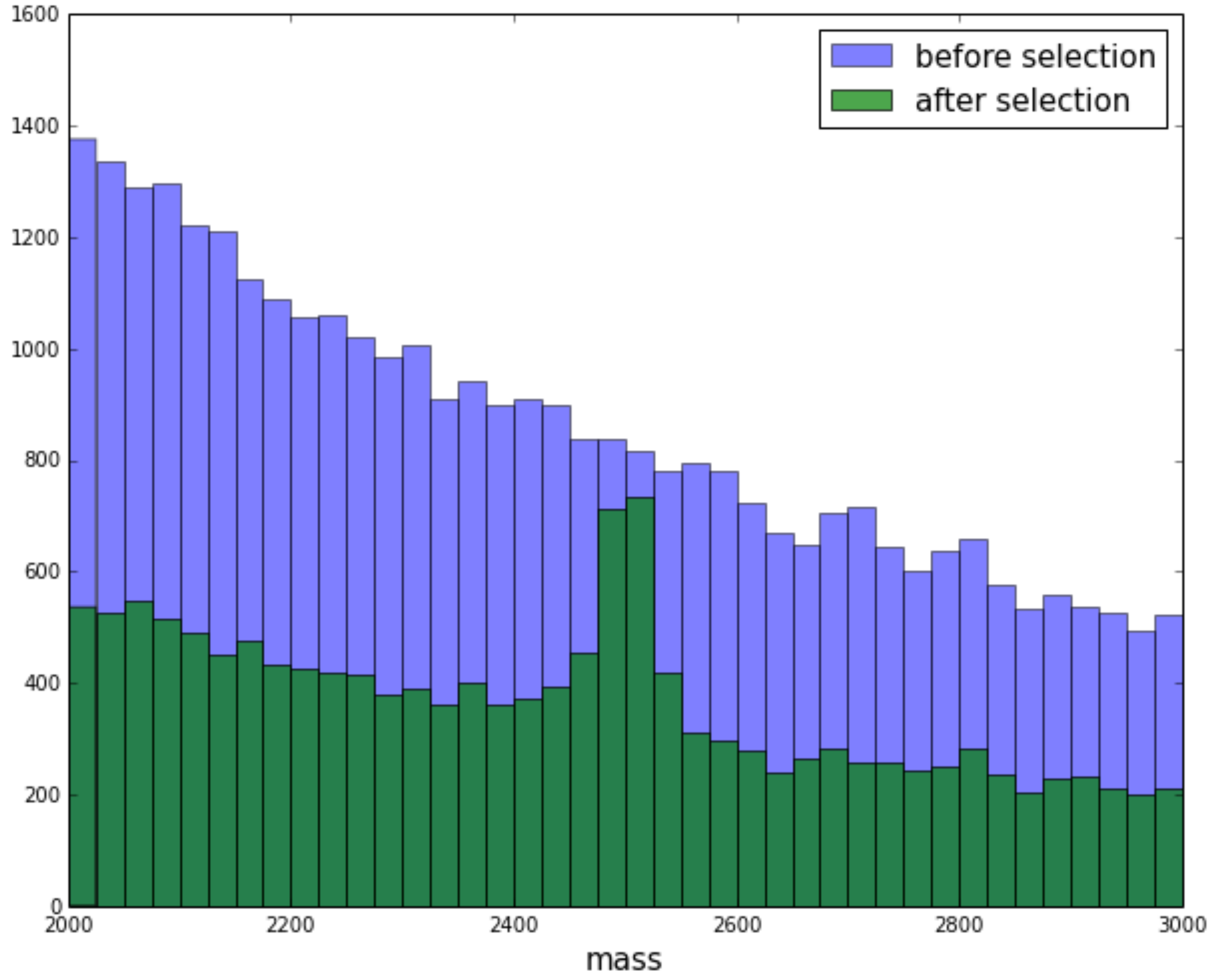


Non-flatness along the mass

High correlation with the mass can create from pure background false peaking signal (specially if we use mass sidebands for training)

Goal: **FPR** $= const$ for different regions in mass

FPR = background efficiency



Basic approach

- reduce the number of features used in training
- leave only the set of features, which do not give enough information to reconstruct the mass of particle
  - simple and works
- sometimes we have to lose information

Can we modify ML to use all features, but provide uniform background efficiency (FPR)/signal efficiency (TPR) along the mass?

Gradient boosting recall

Gradient boosting greedily builds an ensemble of estimators

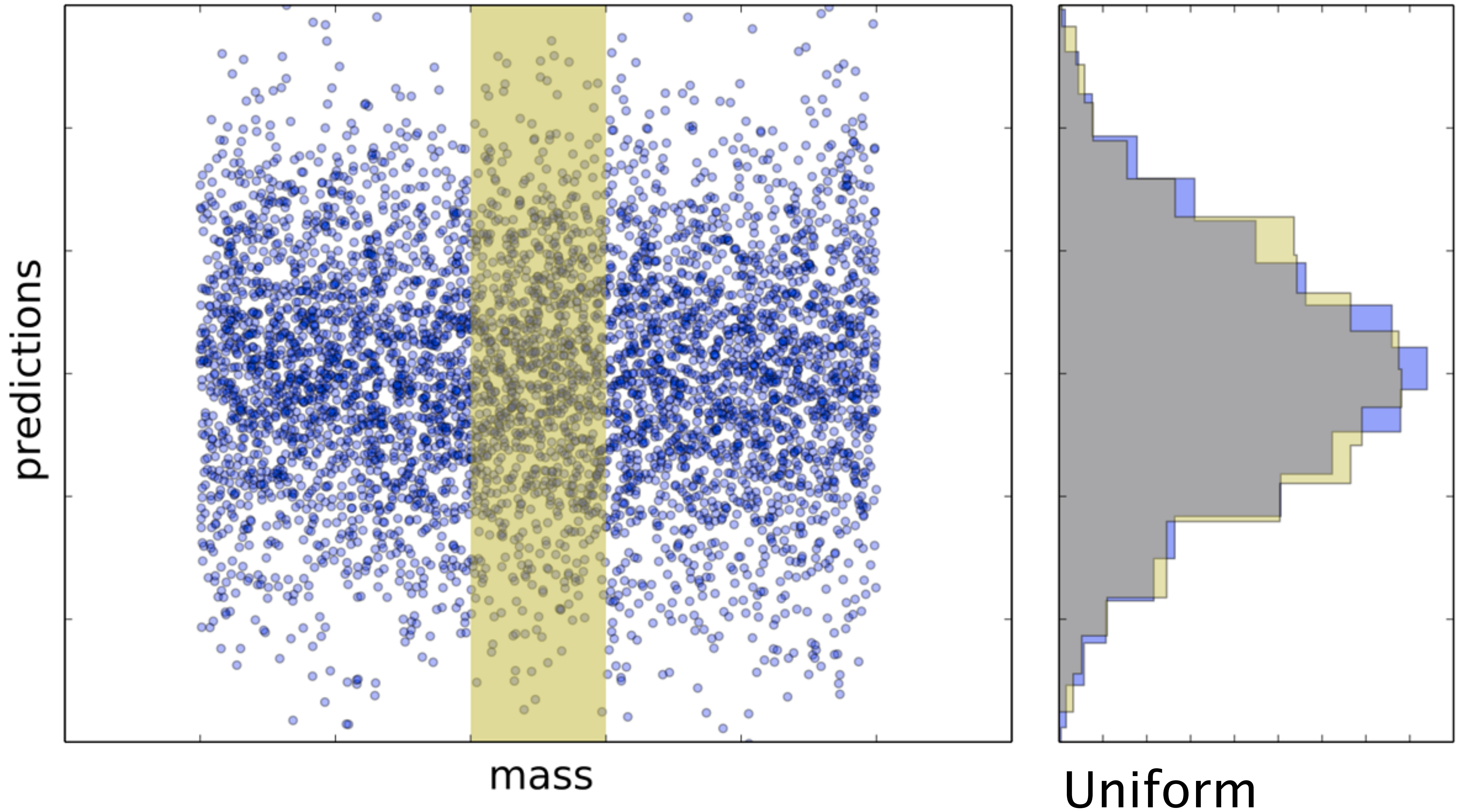
$$D(x) = \sum_j \alpha_j d_j(x)$$

by optimizing some loss function. Those could be:

- MSE: $\mathcal{L} = \sum_i (y_i - D(x_i))^2$
- AdaLoss: $\mathcal{L} = \sum_i e^{-y_i D(x_i)}, \qquad y_i = \pm 1$
- LogLoss: $\mathcal{L} = \sum_i \log(1 + e^{-y_i D(x_i)}), \qquad y_i = \pm 1$

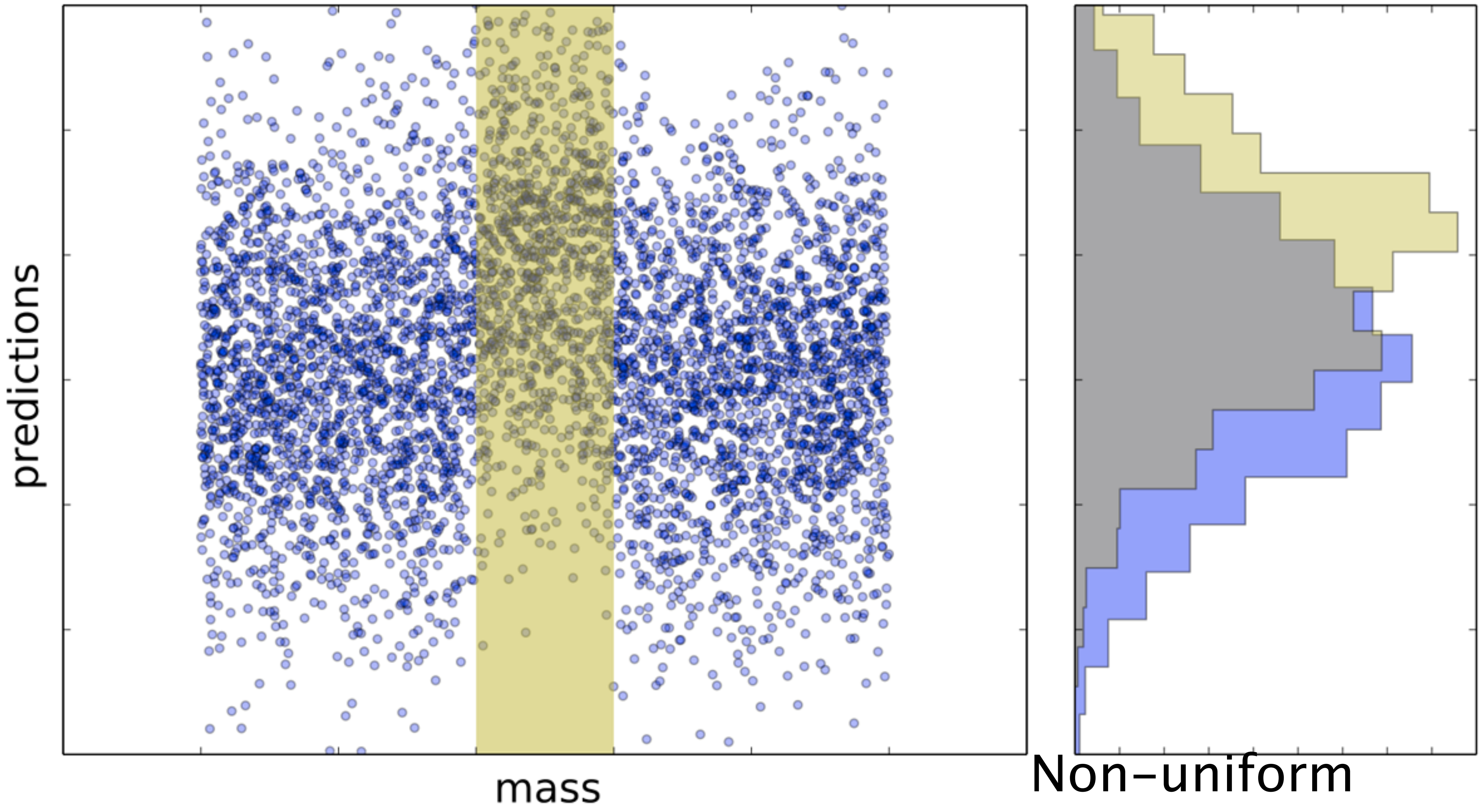
Next estimator in series approximates gradient of loss in the space of functions

Non-uniformity measure



Uniform

predictions



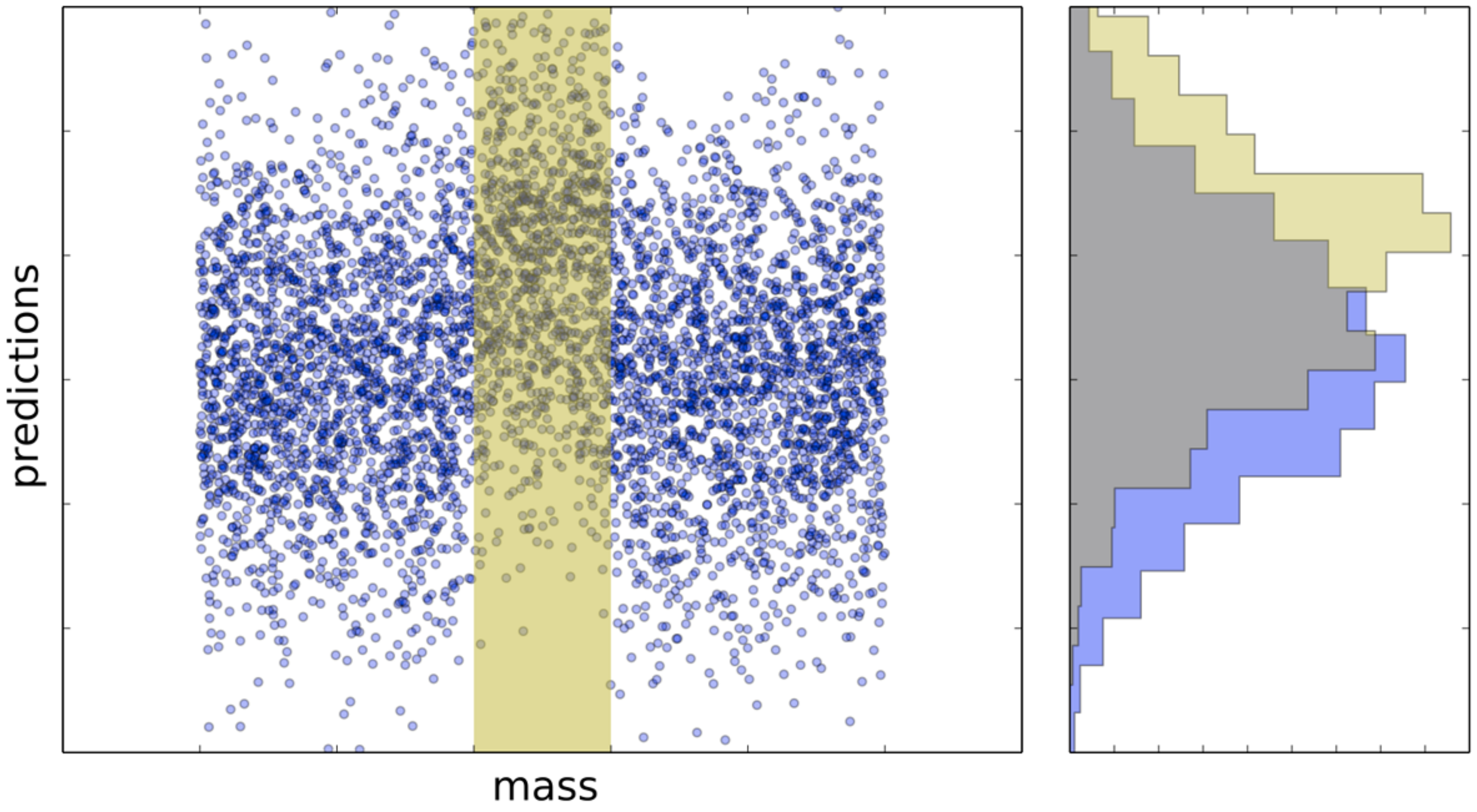
Non-uniform
predictions
(peak in highlighted
region)

- difference in the efficiency can be detected by analyzing distributions
- uniformity = no dependence between the mass and predictions

Non-uniformity measure

Average contributions (difference between global and local distributions) from different regions
in the mass: use for this Cramer-von Mises measure (integral characteristic)

$$\text{CvM} = \sum_{\text{region}} \int |F_{\text{region}}(s) - F_{\text{global}}(s)|^2 \, dF_{\text{global}}(s)$$



Minimizing non-uniformity

- why not minimizing CvM as a loss function with GB?
- … because we can’t compute the gradient, but ROC AUC, classification accuracy are not differentiable too
- also, minimizing CvM doesn't encounter classification problem:
  the minimum of CvM is achieved i.e. on a classifier with random predictions

Flatness loss (FL)

- Put an additional term in the loss function which will penalize for non-uniformity predictions:

$$\mathcal{L} = \mathcal{L}_{\text{adaloss}} + \alpha \mathcal{L}_{\text{FL}}$$

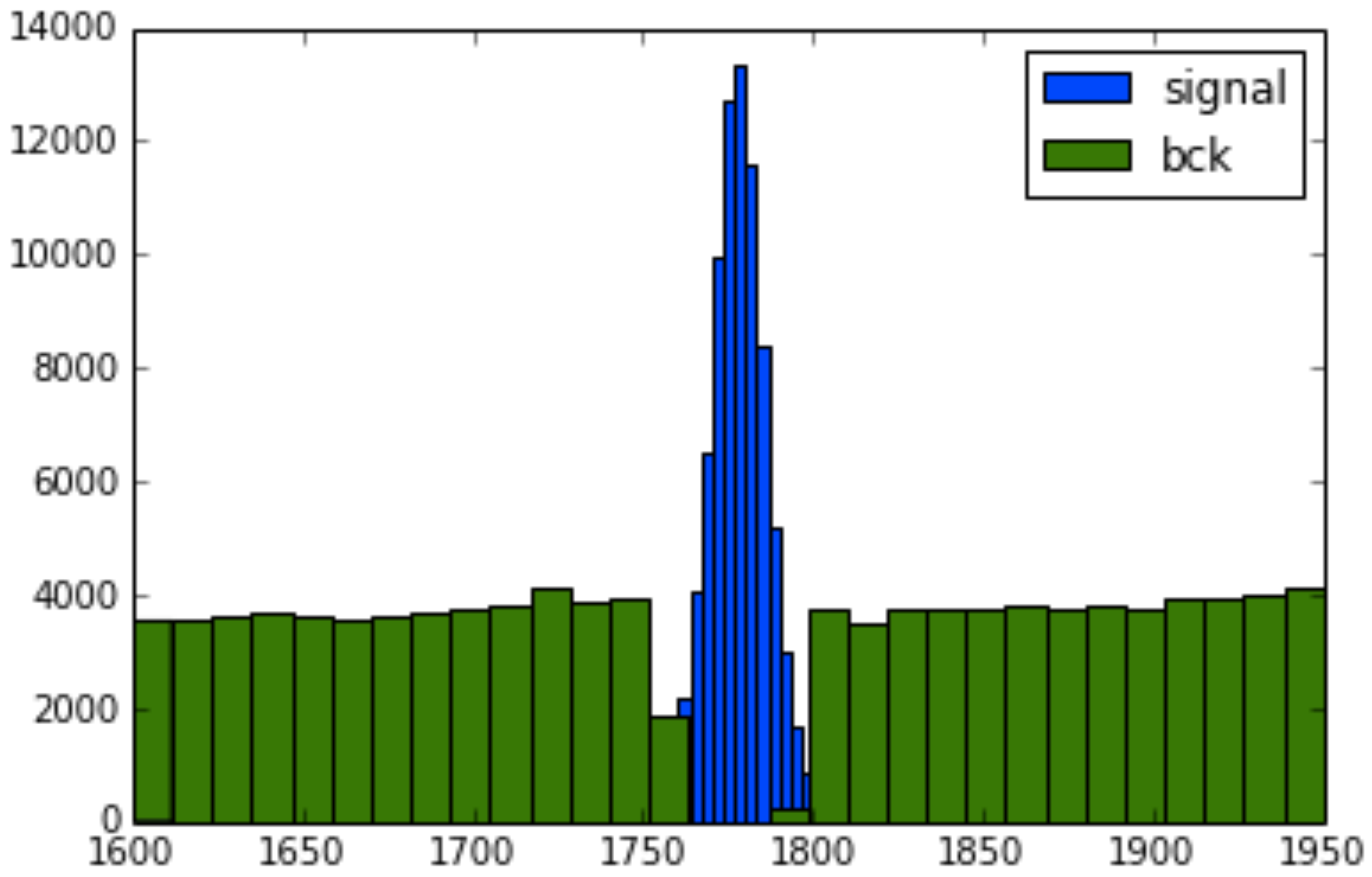
- Flatness loss approximates non-differentiable CvM measure:

$$\mathcal{L}_{\text{FL}} = \sum_{\text{region}} \int \left| F_{\text{region}}(s) - F_{\text{global}}(s) \right|^2 ds$$

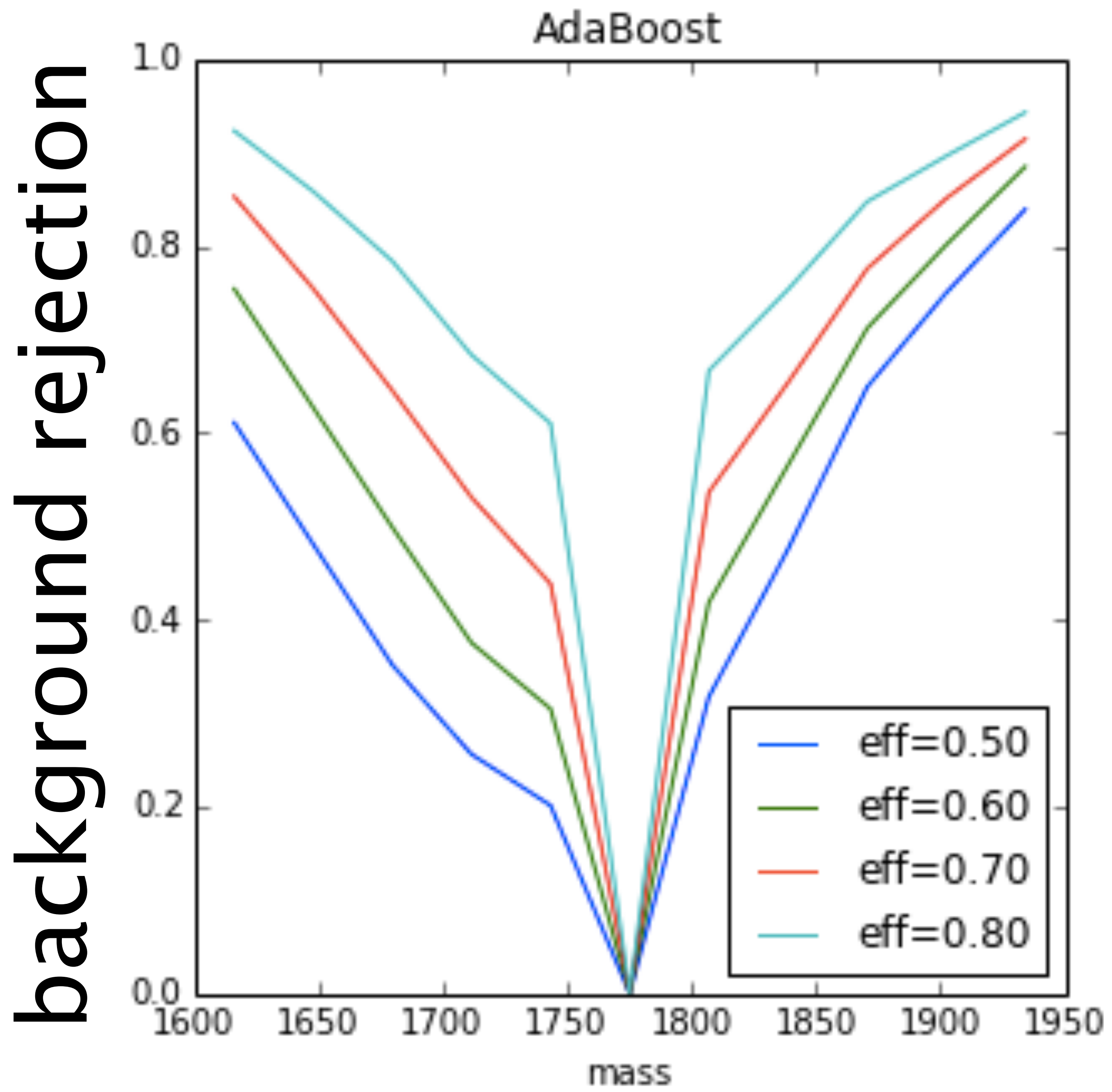
$$\frac{\partial}{\partial D(x_i)} \mathcal{L}_{\text{FL}} \sim 2(F_{\text{region}}(s) - F_{\text{global}}(s)) \Big|_{s=D(x_i)}$$

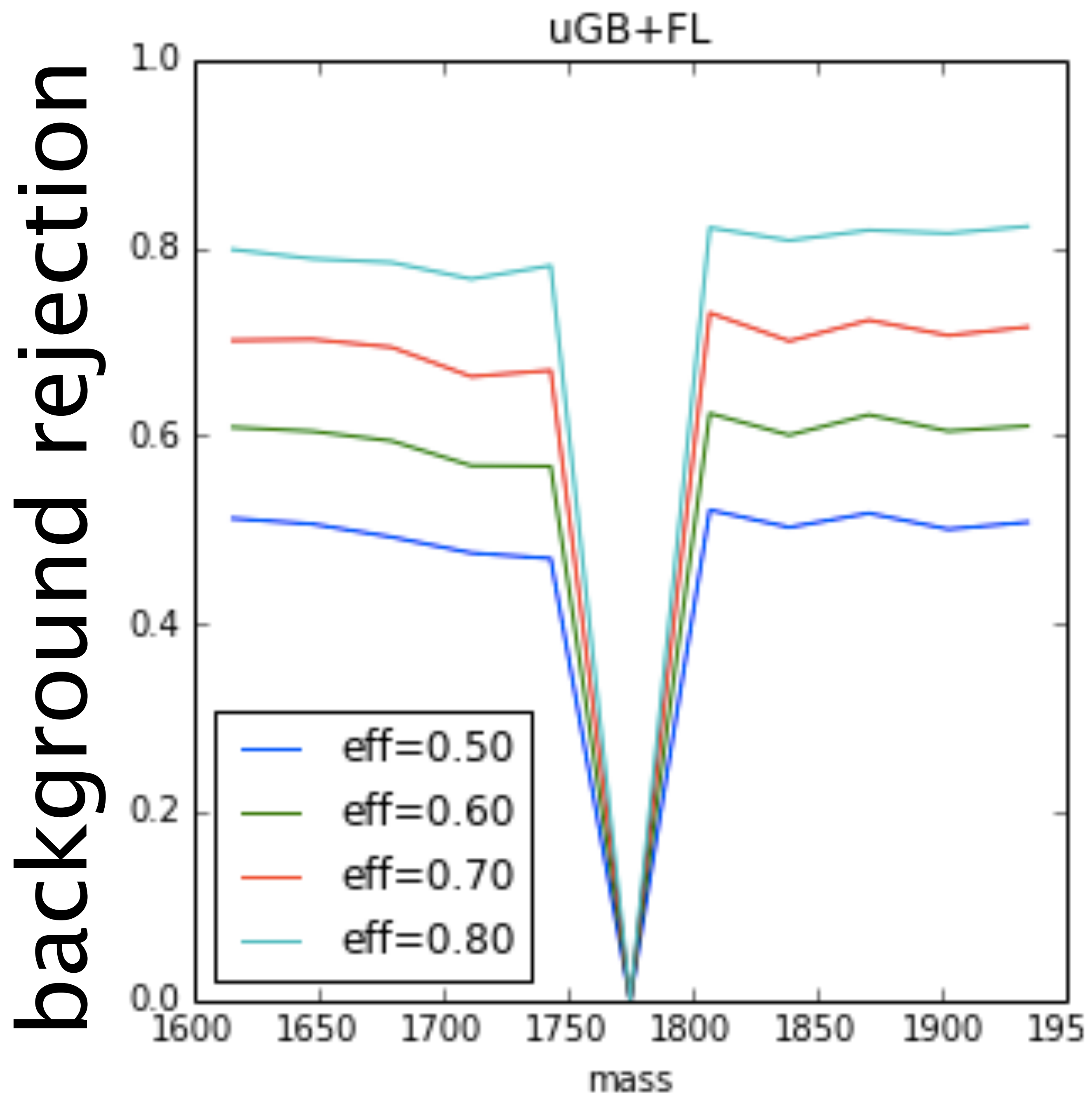
Rare decay analysis DEMO

- when we train on a sideband vs MC using many features, we easily can run into problems (there exist several features which depend on the mass)



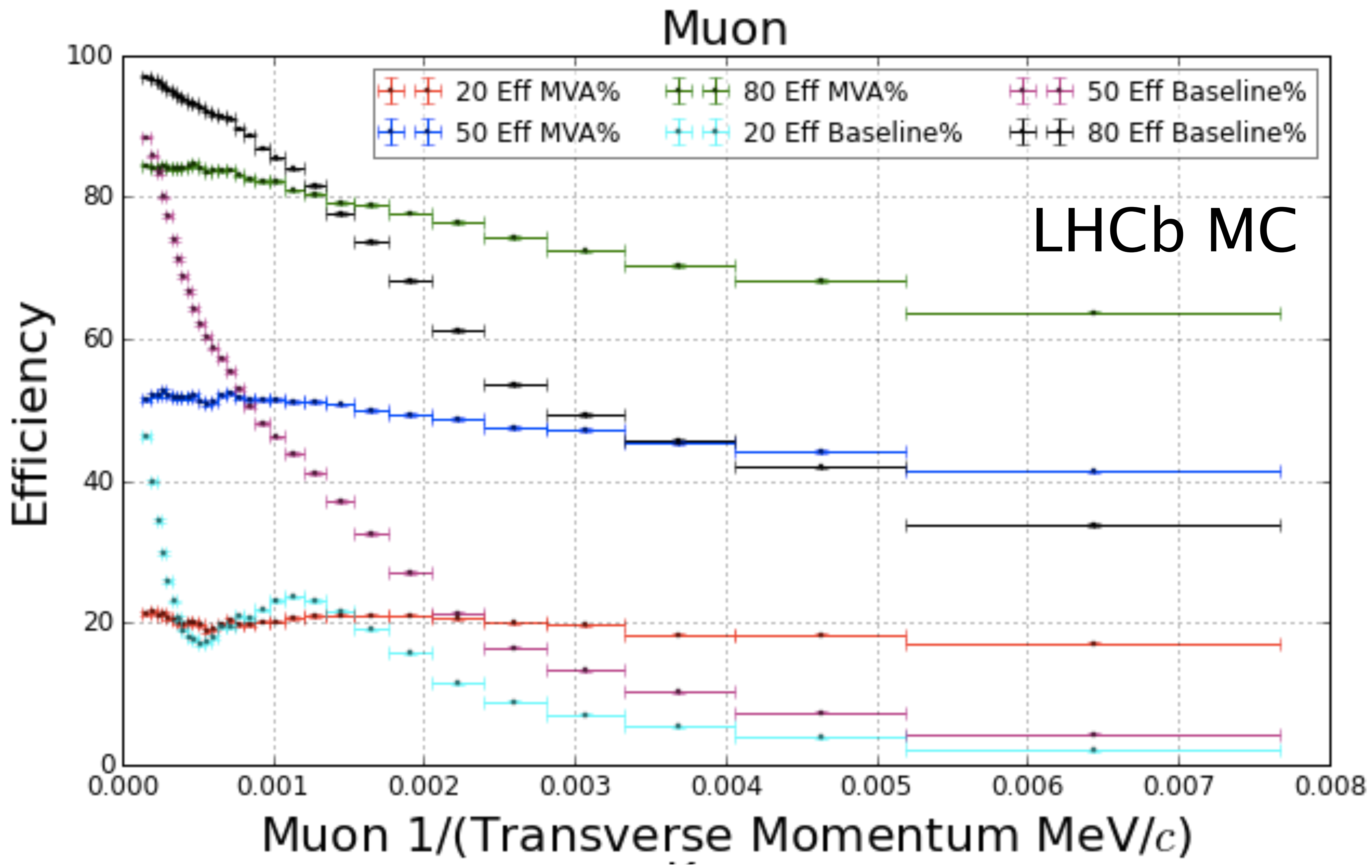
Rare decay analysis DEMO

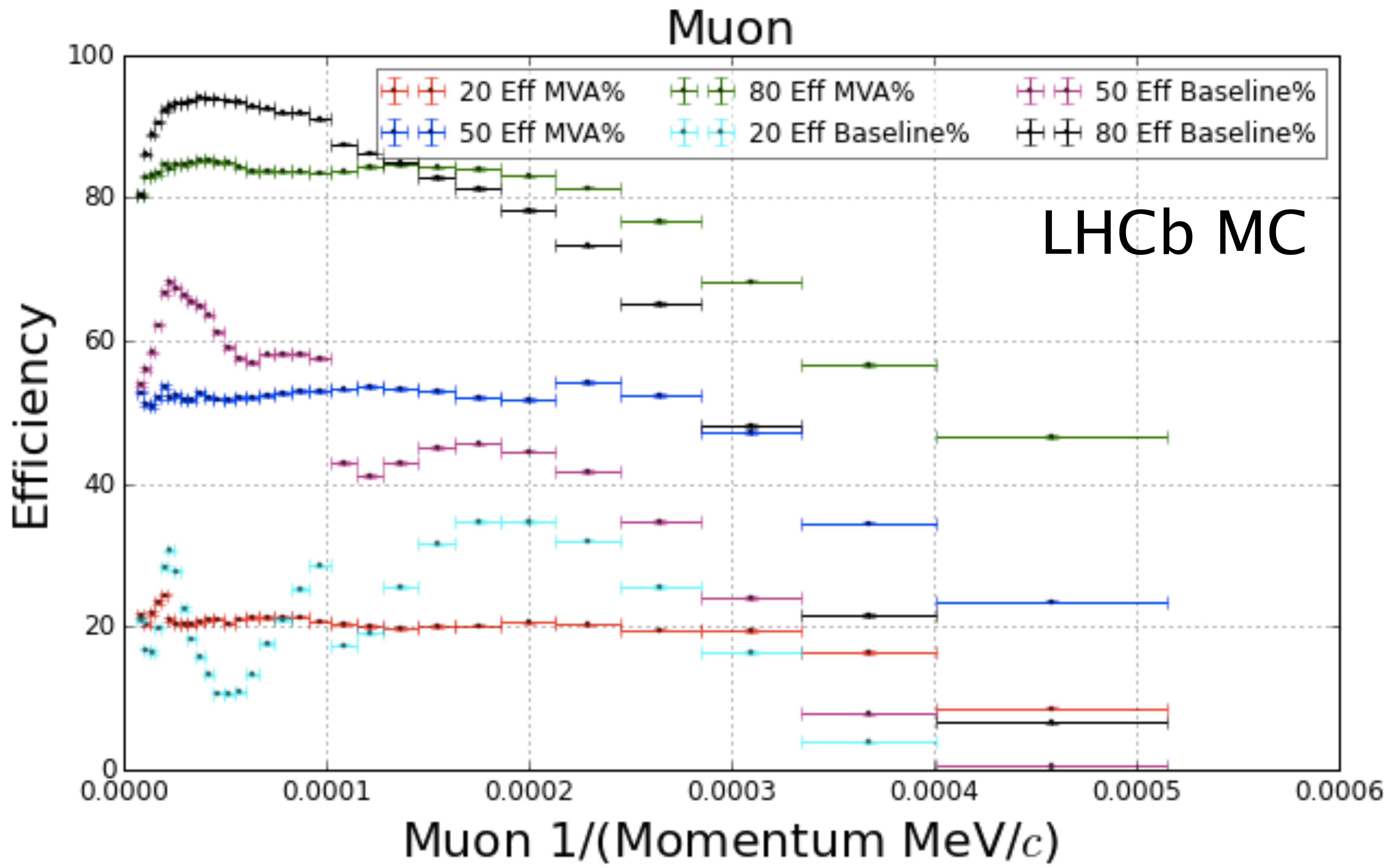




all models use the same set of features for discrimination, but AdaBoost got serious dependence on the mass

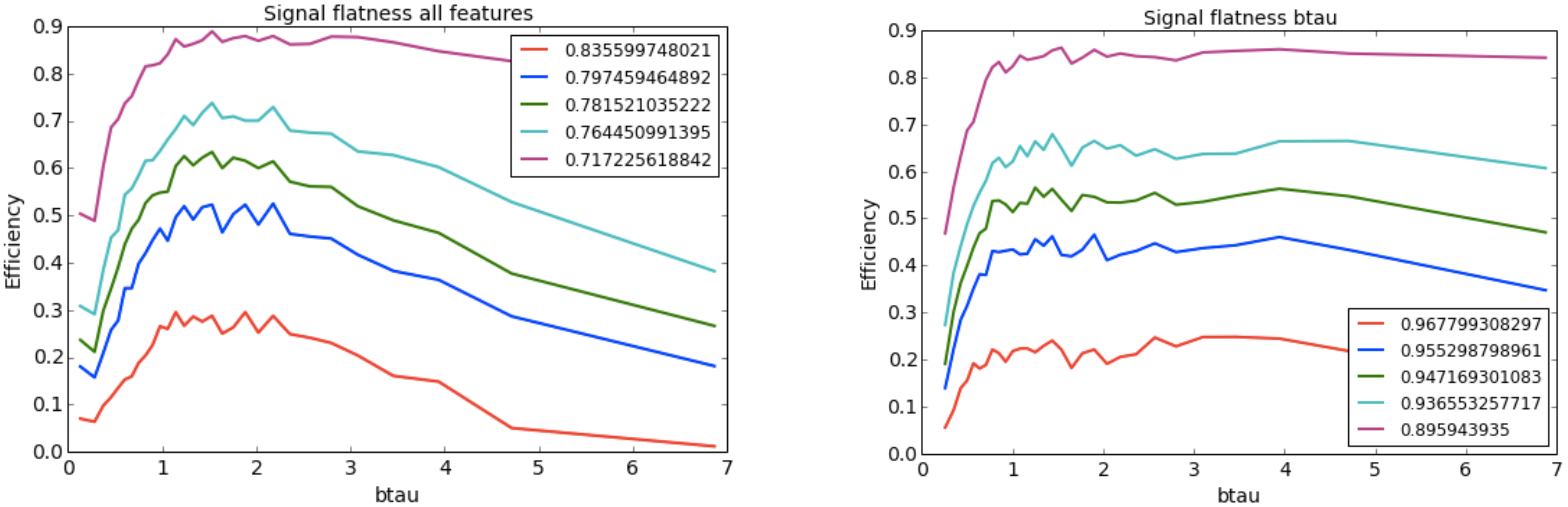
PID Demo (based on LHCb MC)





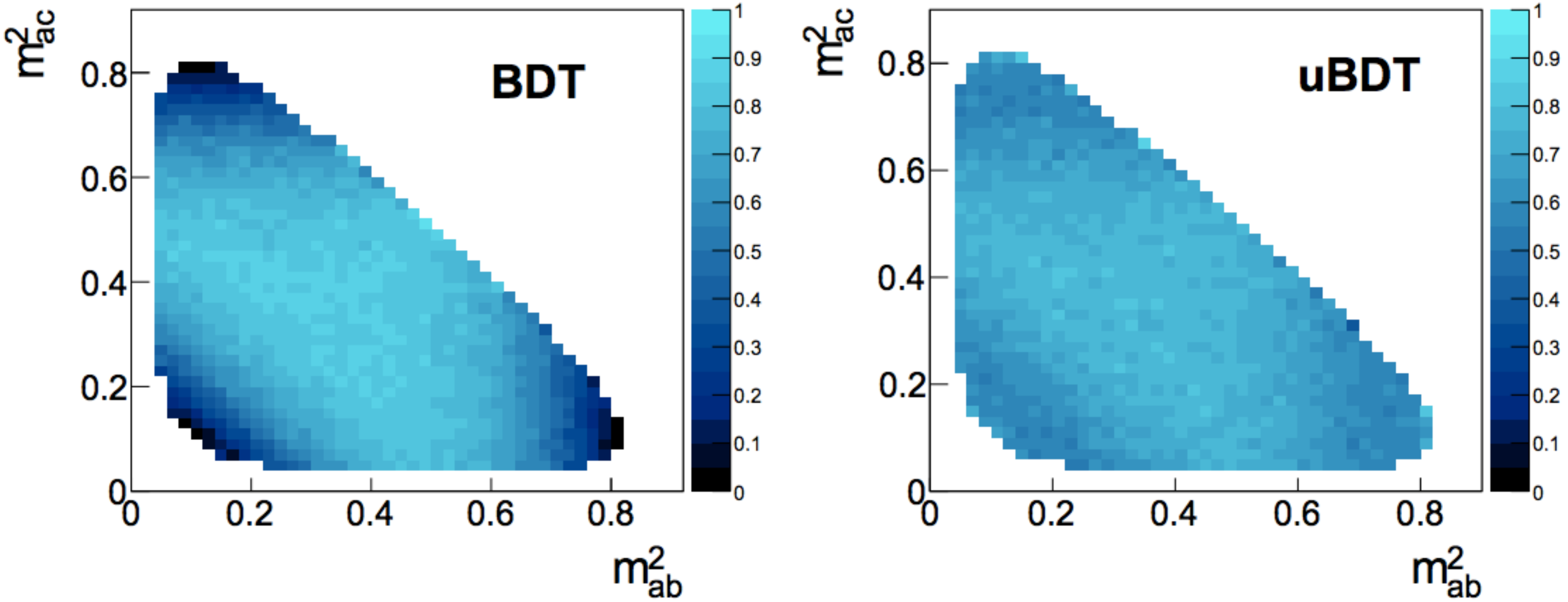
Features strongly depend on the momentum and transverse momentum. Both algorithms use the same set of features.
Used MVA is a specific BDT implementation with flatness loss.

Trigger DEMO



Both algorithms use the same set of features.
The right one is uGB+FL.

Dalitz analysis DEMO



The right one is uBoost algorithm. Global efficiency is set 70%

hep_ml library

```python
from hep_ml import gradientboosting as ugb

# define flatness loss along momentum and transverse momentum
loss = ugb.KnnAdaLossFunction(uniform_features=['trackP', 'trackPt'],
                              knn=10, uniform_label=1)
# define uGB+FL
ugb = ugb.UGradientBoostingClassifier(loss=loss, max_depth=4,
                                      n_estimators=100,
                                      learning_rate=0.4)
ugb.fit(data, target)
ugb.predict_proba(data_test)
```

Summary

1. uBoost approach
2. Non-uniformity measure
3. uGB+FL approach: gradient boosting with flatness loss (FL)

uBoost, uGB+FL:

- produce flat predictions along the set of features
- there is a trade off between classification quality and uniformity

References

- https://arxiv.org/abs/1305.7248
- https://arxiv.org/abs/1410.4140
- https://arogozhnikov.github.io/hep_ml/

Thanks for attention

Special thanks

- To people from Yandex group who were involved in preparing the slides:
  - Tatiana Likhomanenko
  - Fedor Ratnikov
  - Alex Rogozhnikov
  - Mikhail Hushchyn