SCHOOL OF DATA ANALYSIS

Non-trivial applications of boosting

Tatiana Likhomanenko

Machine leaning course at Imperial College London, Jan-Feb 2017

*many slides are taken from Alex Rogozhnikov's presentations

Boosting recapitulation

- Boosting combines weak learners to obtain a strong one
- It is usually built over decision trees
- State-of-the-art results in many areas
- General-purpose implementations are used for classification and regression

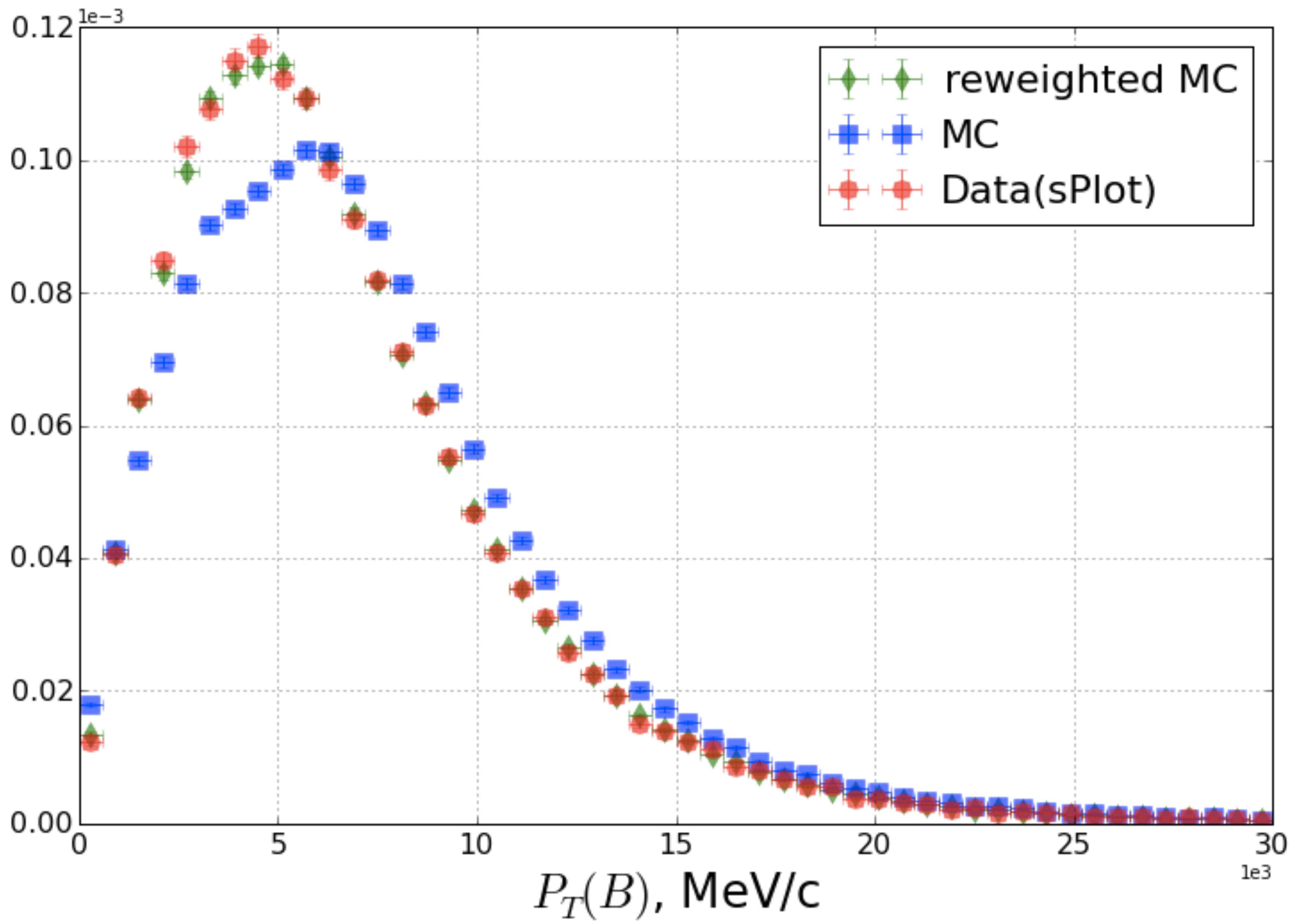
Reweighting problem in HEP

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



Applications beyond physics

- Introducing corrections to fight non-response bias: assigning higher weight to answers from groups with low response.
- See e.g. R. Kizilcec, "Reducing non-response bias with survey reweighting: Applications for online learning researchers", 2014.

Typical approach: histogram reweighting

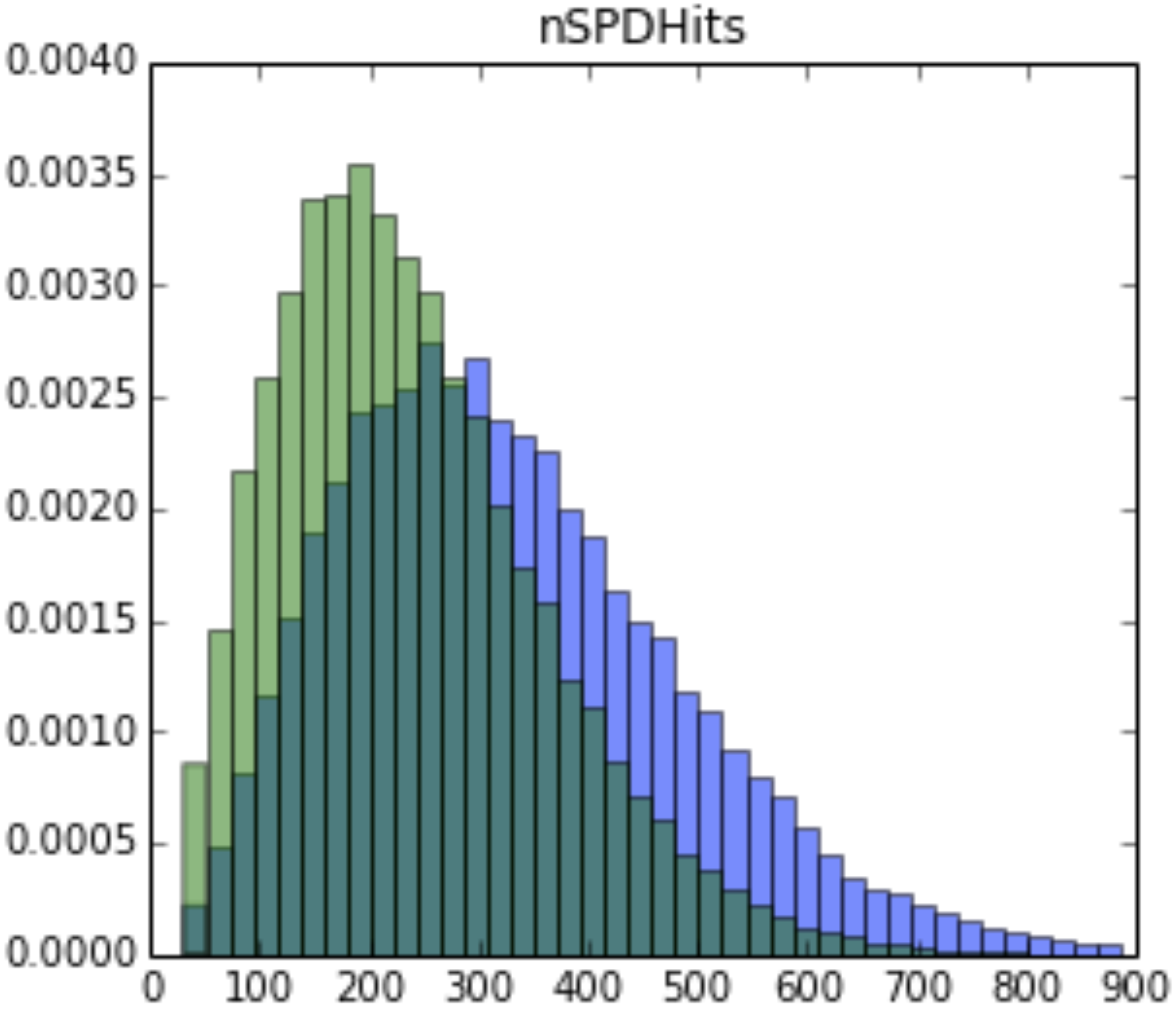
- variable(s) is split into bins
- in each bin the MC weight is multiplied by:

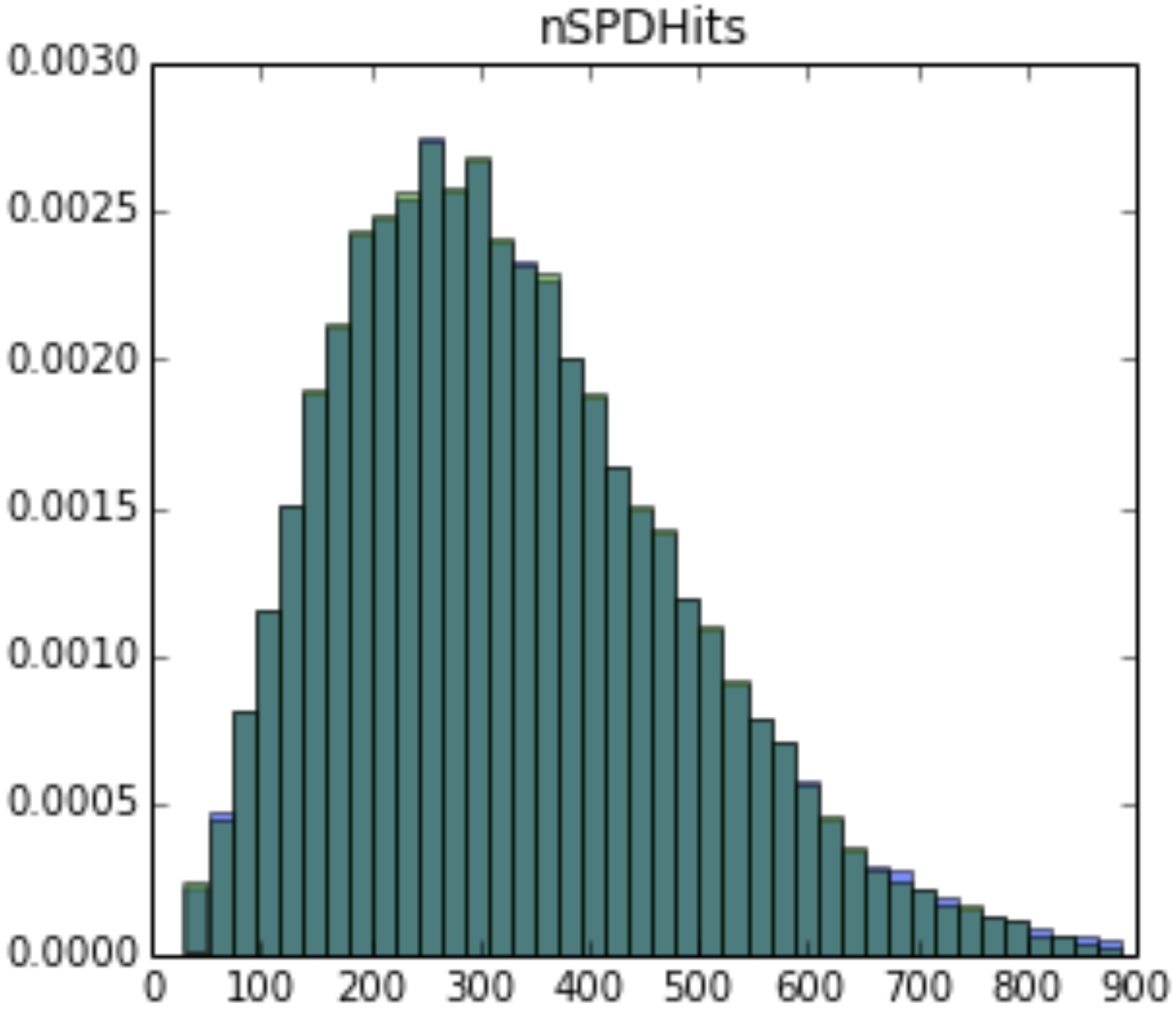
$$\text{multiplier}_{\text{bin}} = \frac{w_{\text{bin, target}}}{w_{\text{bin, original}}}$$

$w_{\text{bin, target}}$, $w_{\text{bin, original}}$ - 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

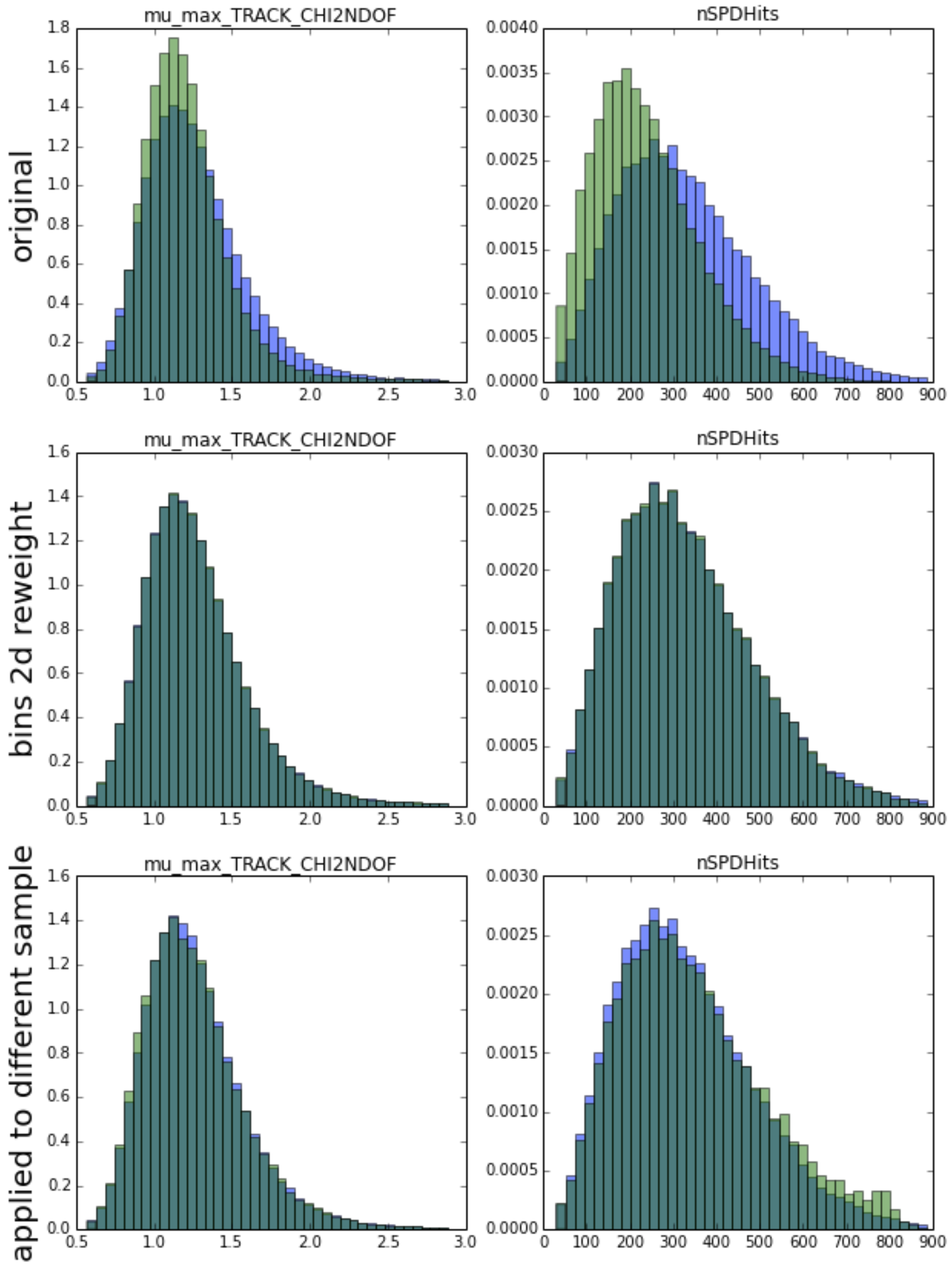




Typical approach: example

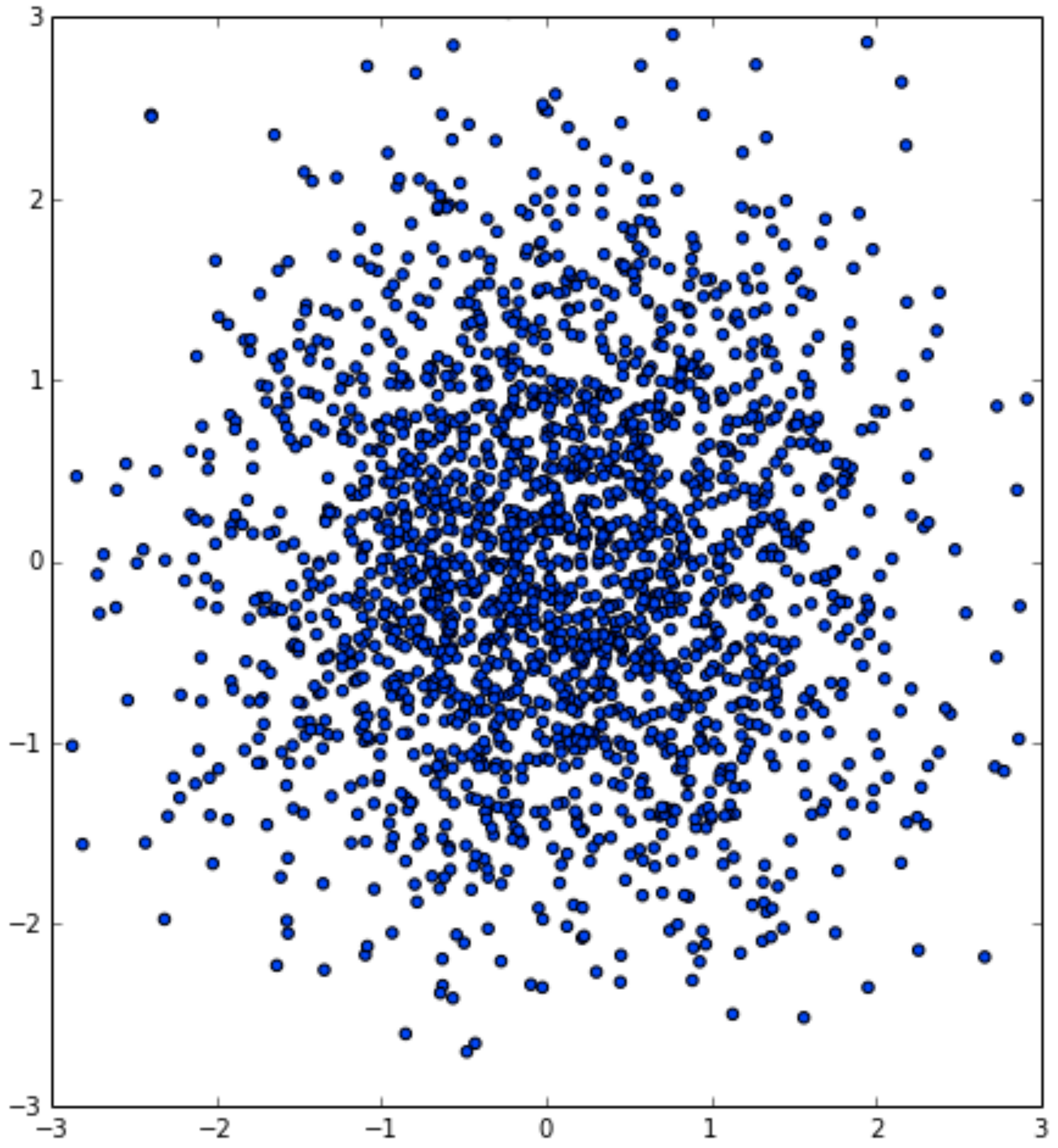
- Problems arise when there are too few events in a bin
- This can be detected on a **holdout** (see the latest row)
- Issues:
    1. few bins - rule is rough
    2. many bins - rule is not reliable

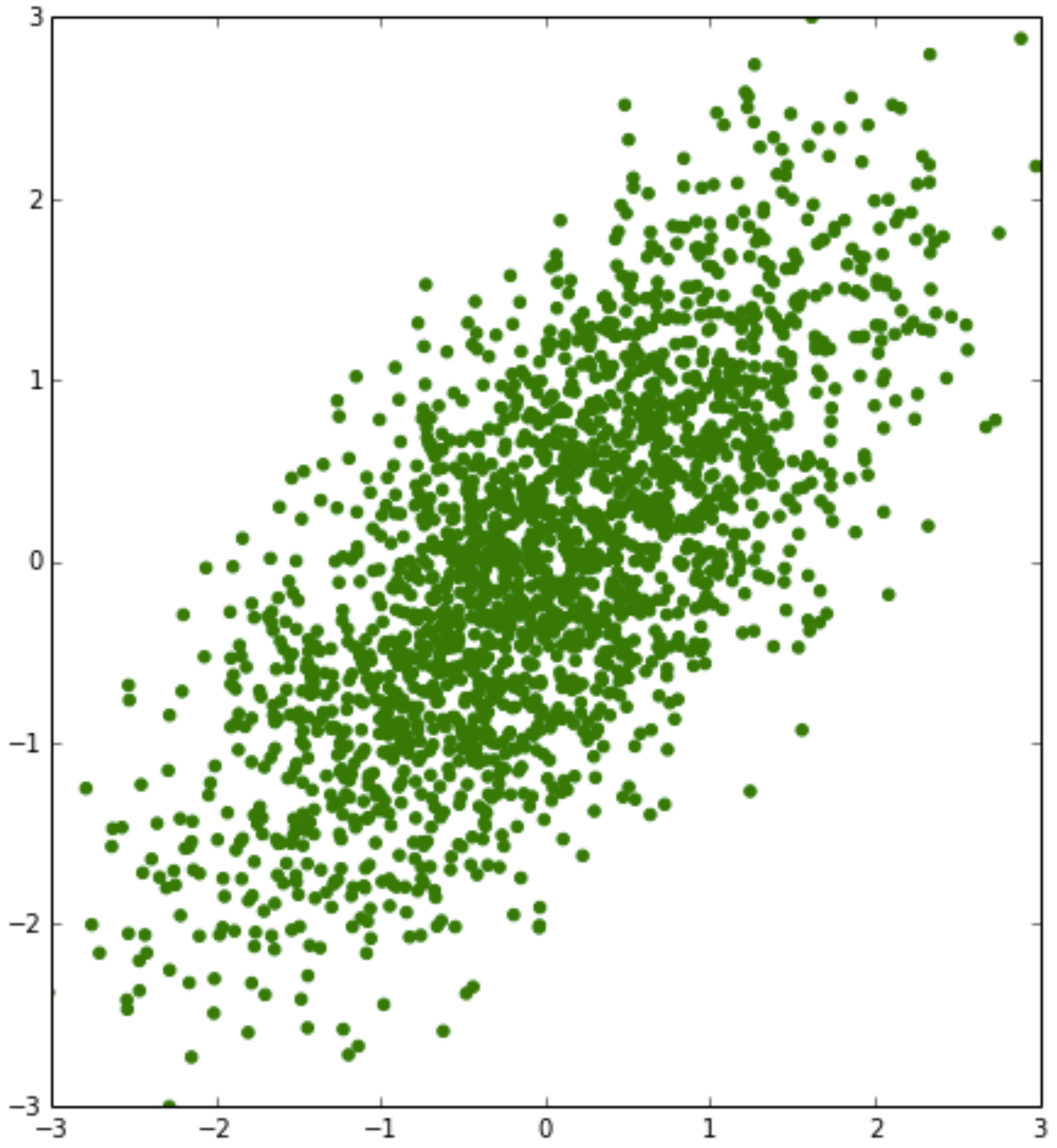
Reweighting rule must be checked on a holdout!



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

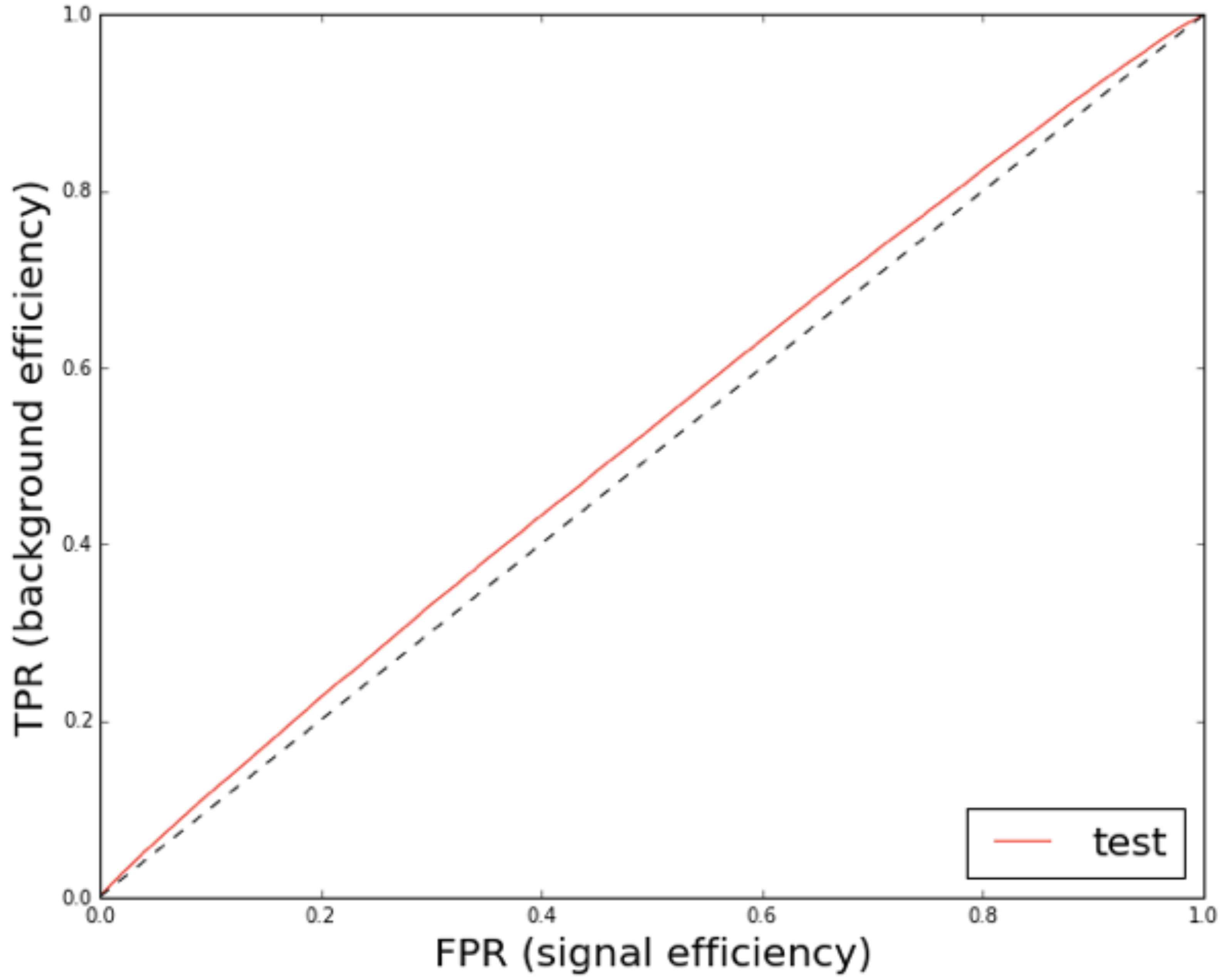




Comparing nDim distributions using ML

- 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: $\dfrac{f_{RD}(x)}{f_{MC}(x)}$
- Classifier trained to discriminate MC and RD should reconstruct probabilities $p_{MC}(x)$ and $p_{RD}(x)$
- For reweighting we can use $\dfrac{f_{RD}(x)}{f_{MC}(x)} \sim \dfrac{p_{RD}(x)}{p_{MC}(x)}$

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

•••

- 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:

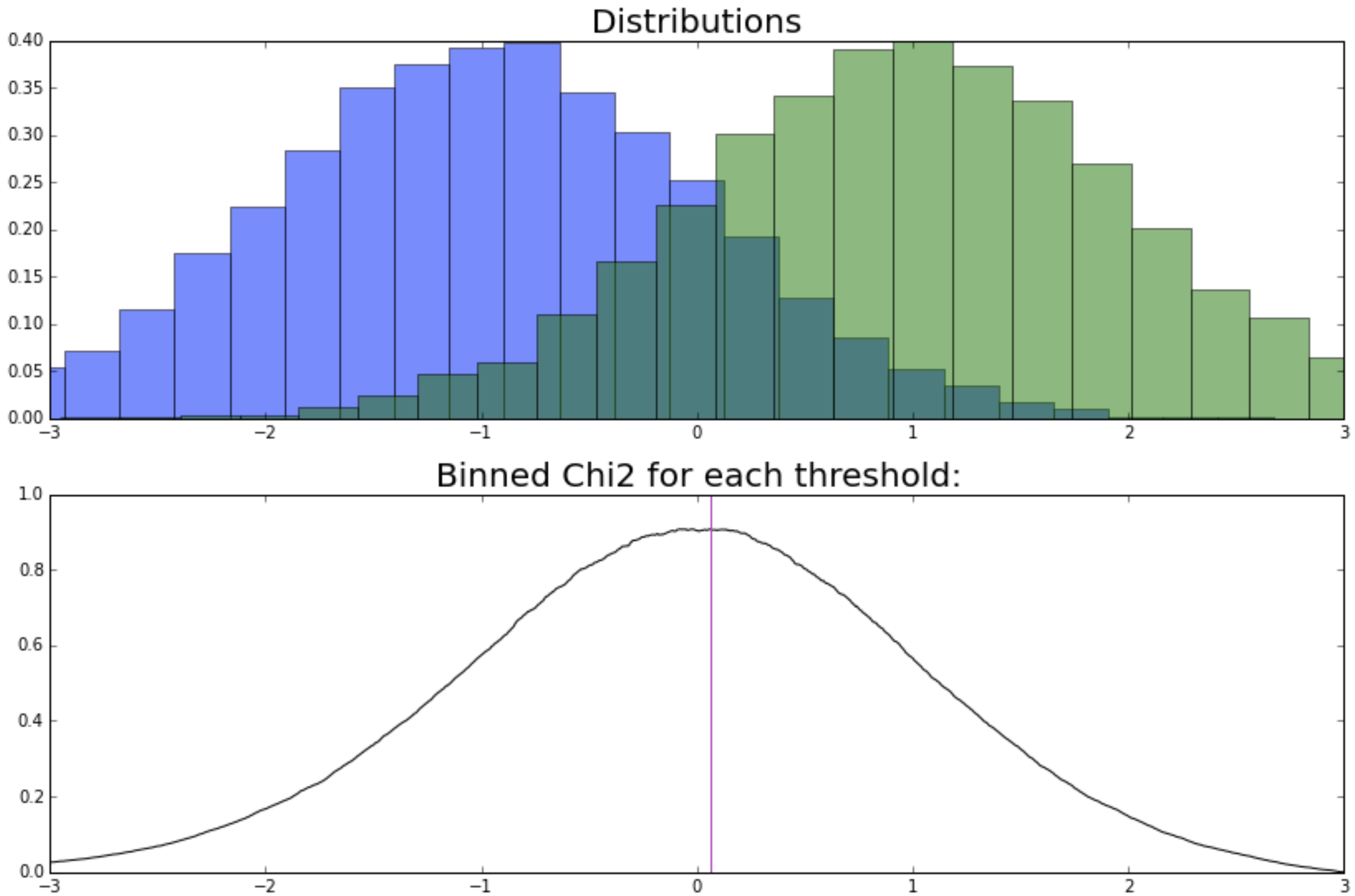
- Tree splits the space of variables with orthogonal cuts (each tree leaf is a region, or bin)
- There are different criteria to construct a tree (MSE, Gini index, entropy, …)
- Find regions with the highest difference between original and target distribution

Spitting criteria

Finding regions with high difference between original and target distribution by maximizing symmetrized χ^2:

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

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.



AdaBoost (Adaptive Boosting) recall

- building of weak learners one-by-one, predictions are summed:

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

- each time increase weights of events incorrectly classified by a tree $d(x)$

$$w_i \leftarrow w_i \exp(-\alpha y_i d(x_i))), \qquad y_i = \pm 1$$

- main idea: provide base estimator (weak learner) with information about which samples have higher importance

BDT reweighter

Many times repeat the following steps:

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

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

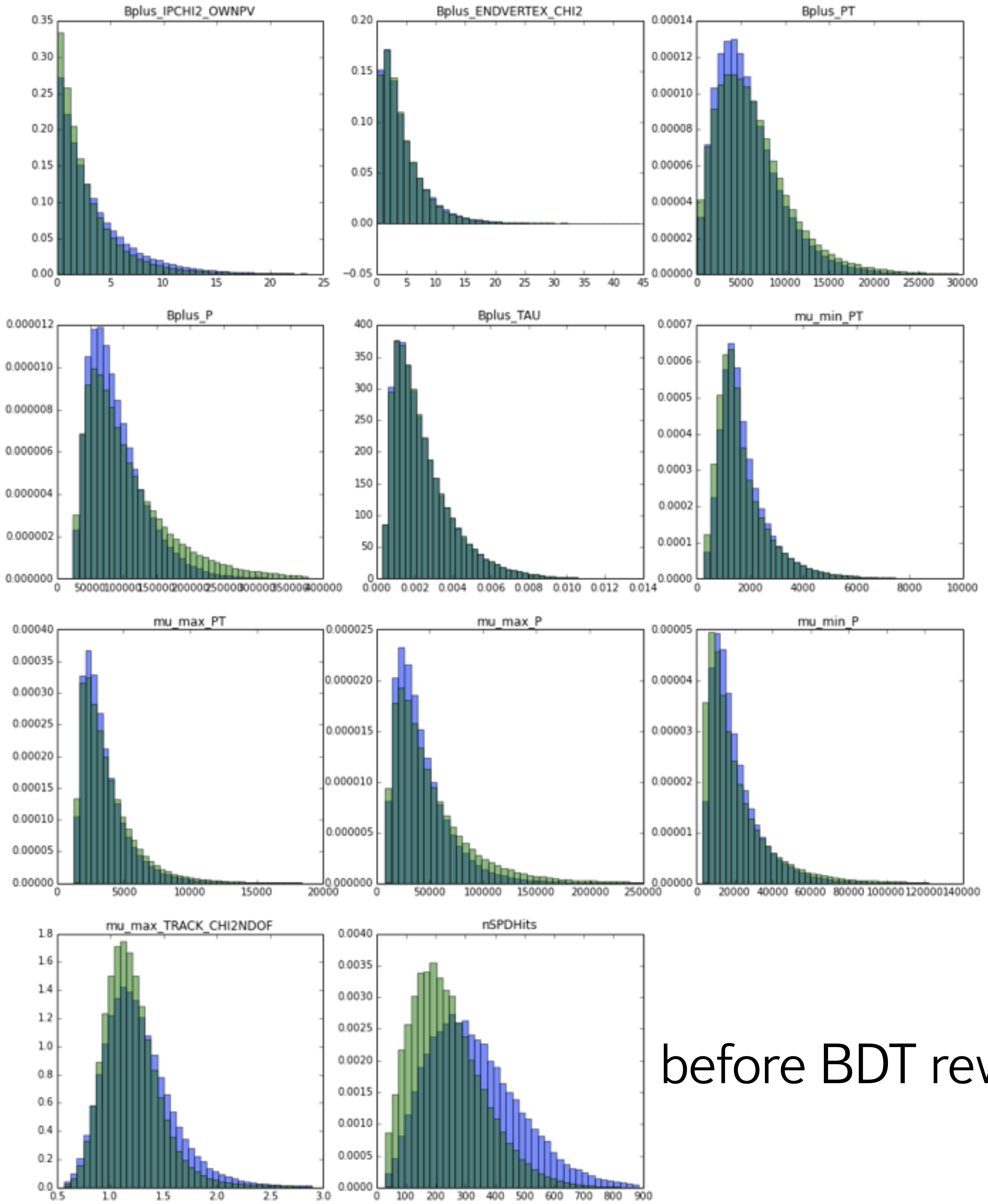
- reweight distributions (compare with AdaBoost):

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

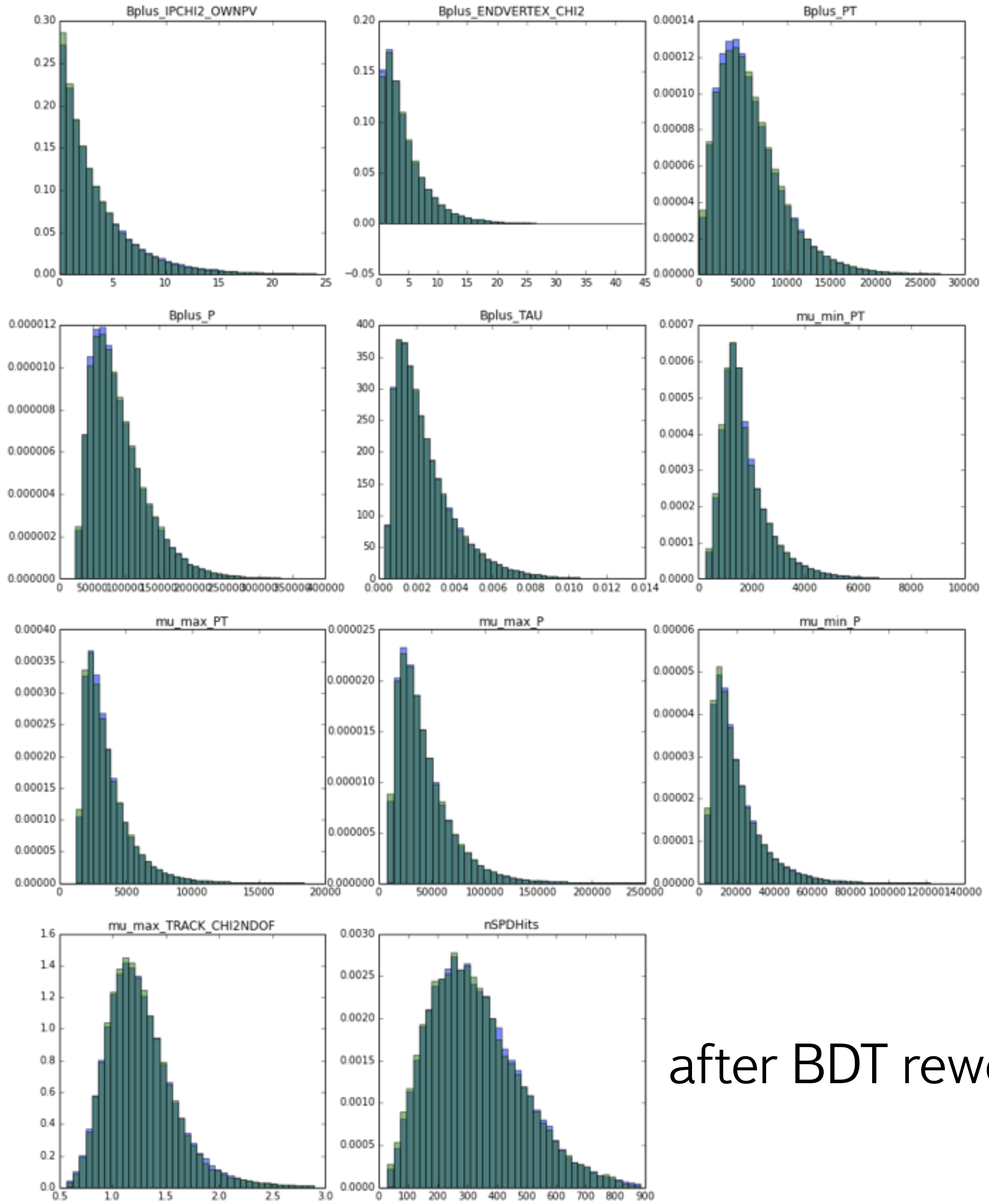
Comparison with GBDT:

- different tree splitting criterion
- different boosting procedure

BDT reweighter DEMO



before BDT reweighting



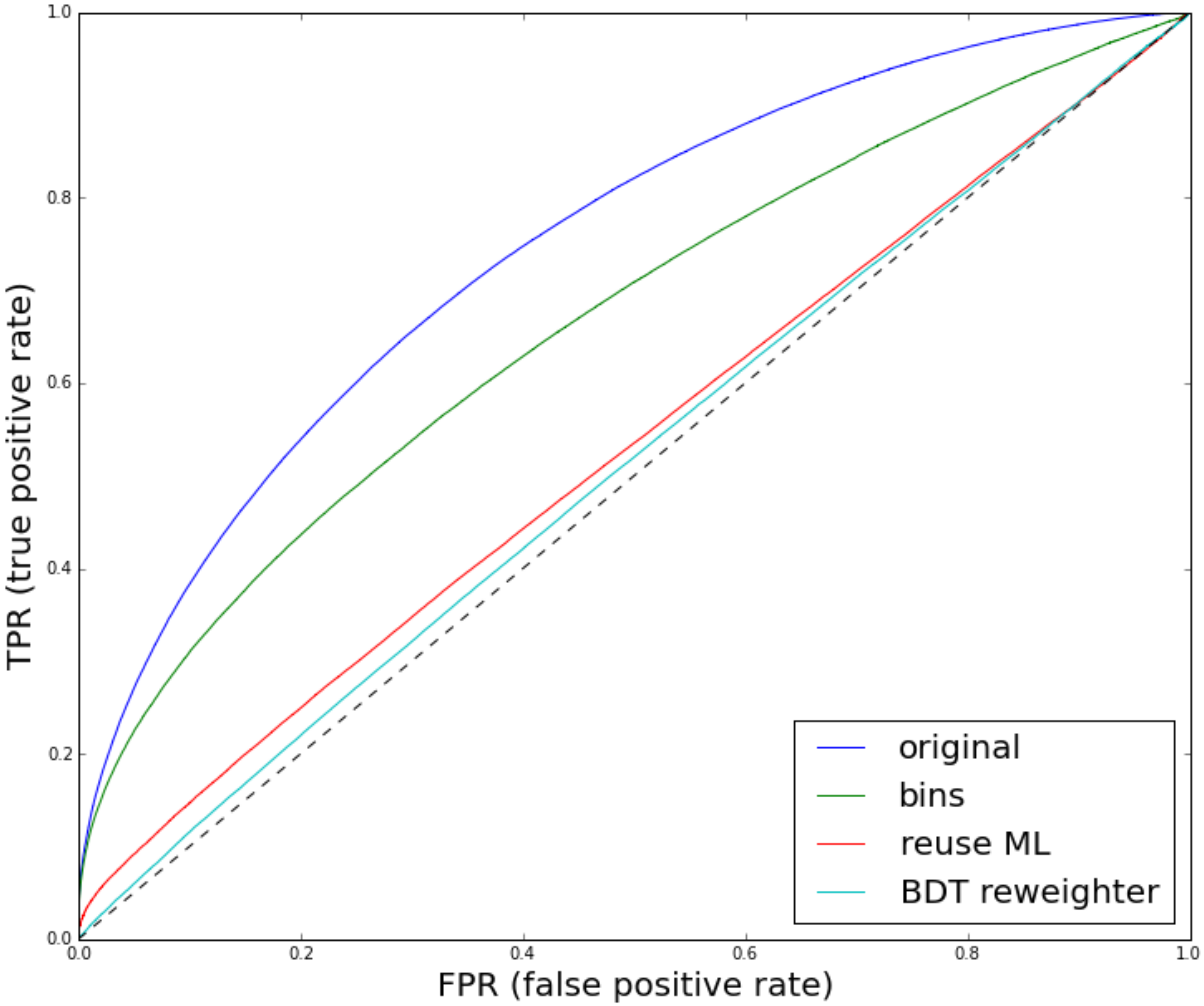
after BDT reweighting

KS 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

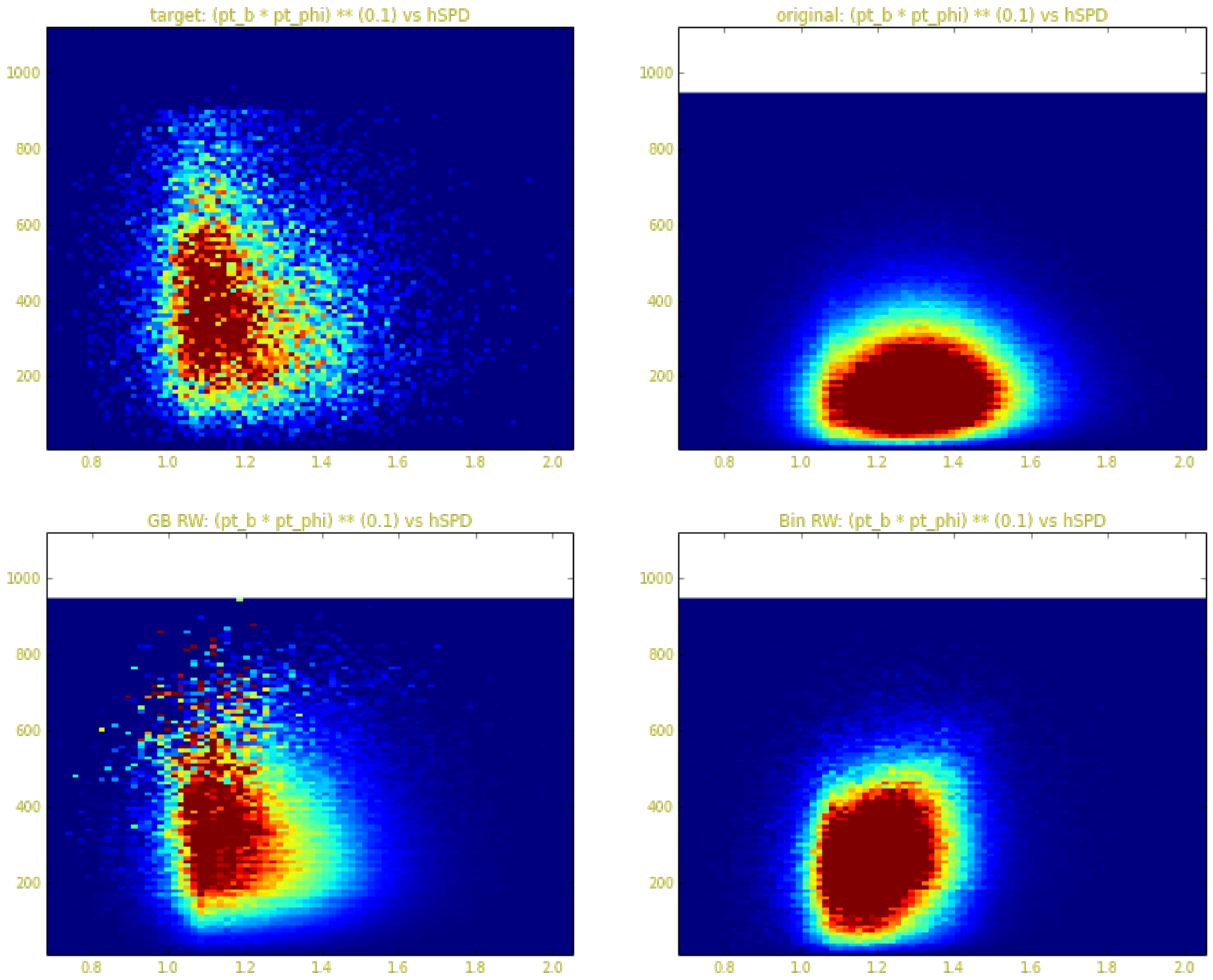
```
from hep_ml.reweight import GBReweighter
gb = GBReweighter()
gb.fit(mc_data, real_data, target_weight=real_data_sweights)
gb.predict_weights(mc_other_channel)
```

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

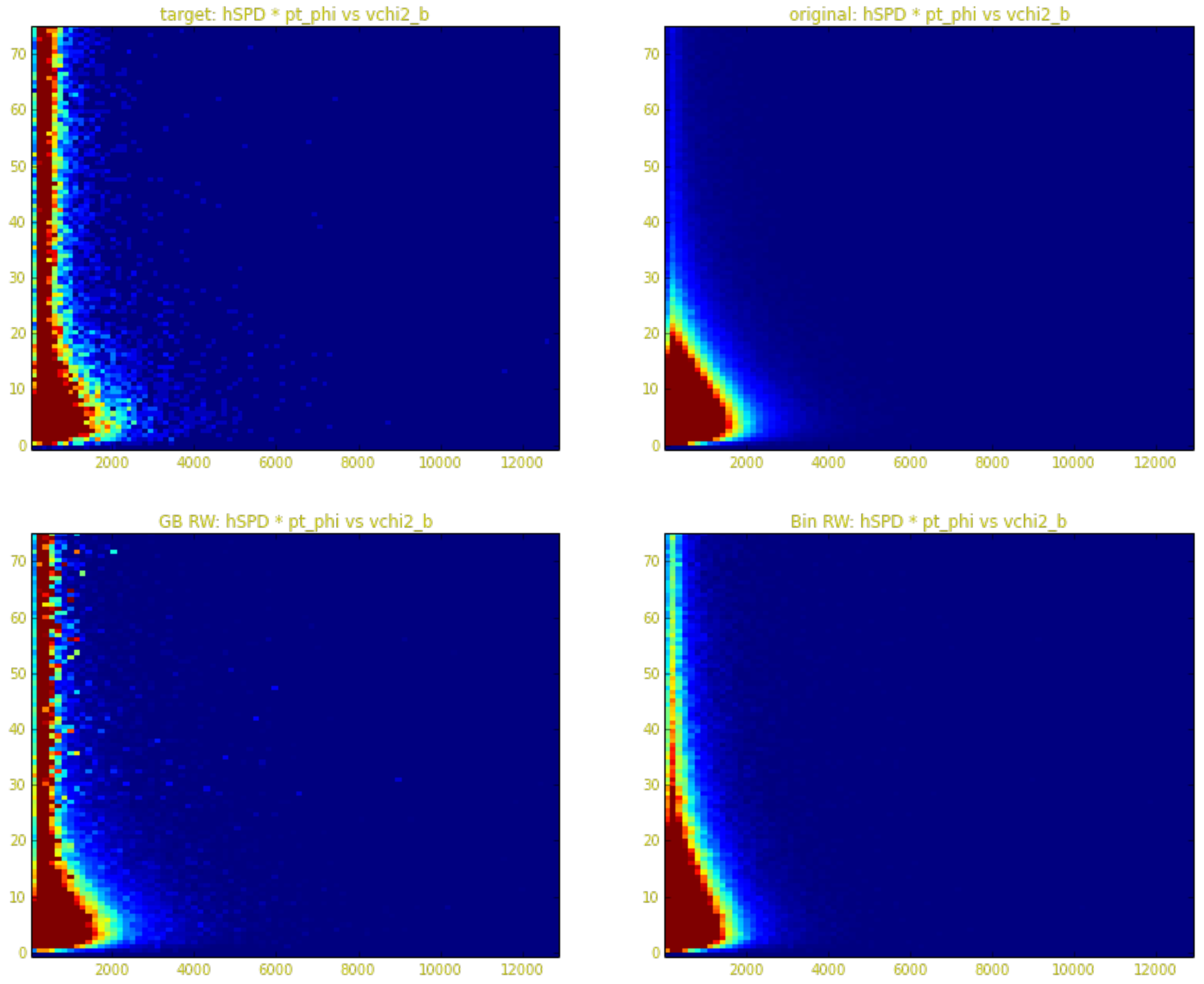
More DEMO:1

Bin vs GB reweighting:
feature combinations



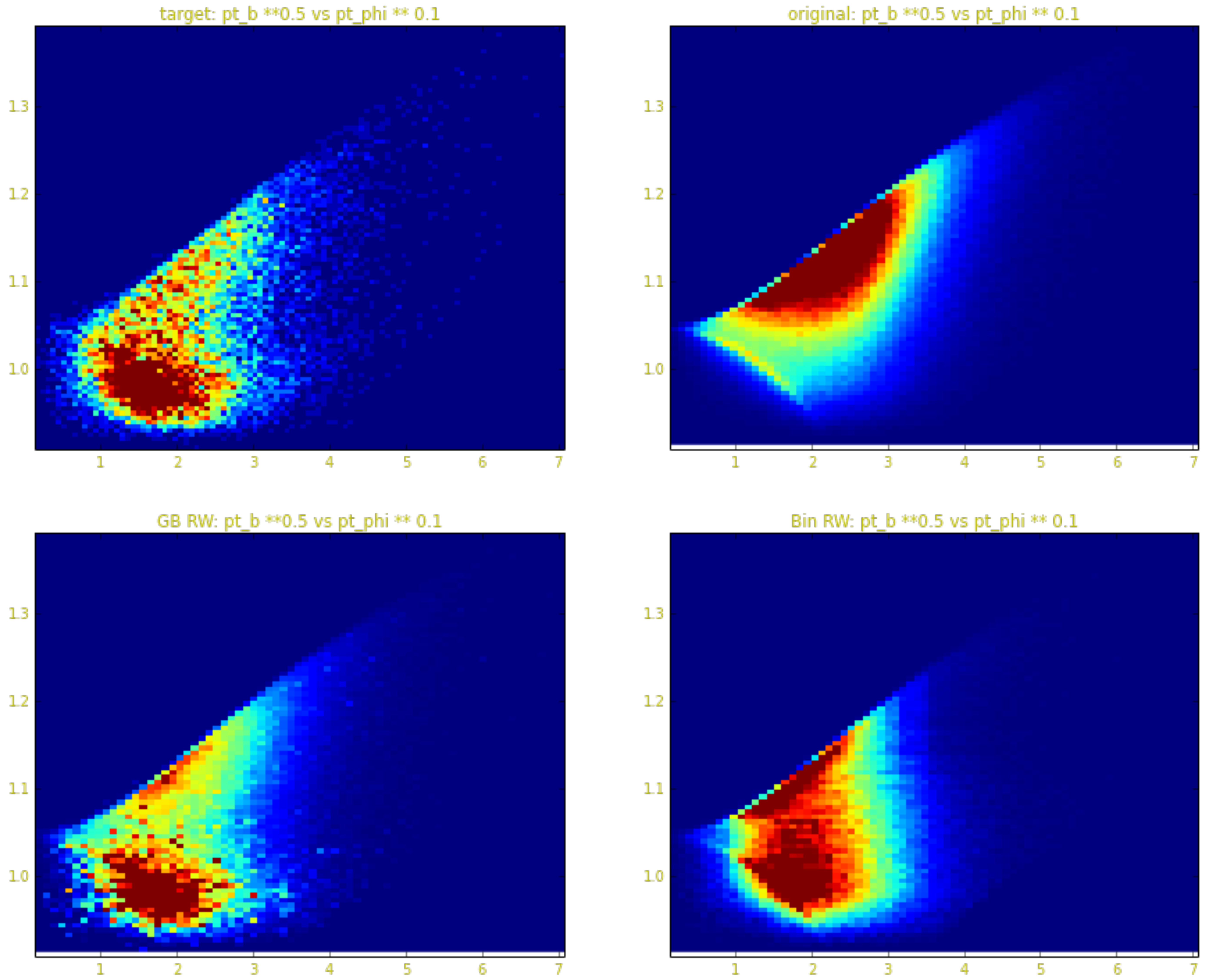
More DEMO:2

Bin vs GB reweighting:
feature combinations



More DEMO:3

Bin vs GB reweighting:
feature combinations



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)

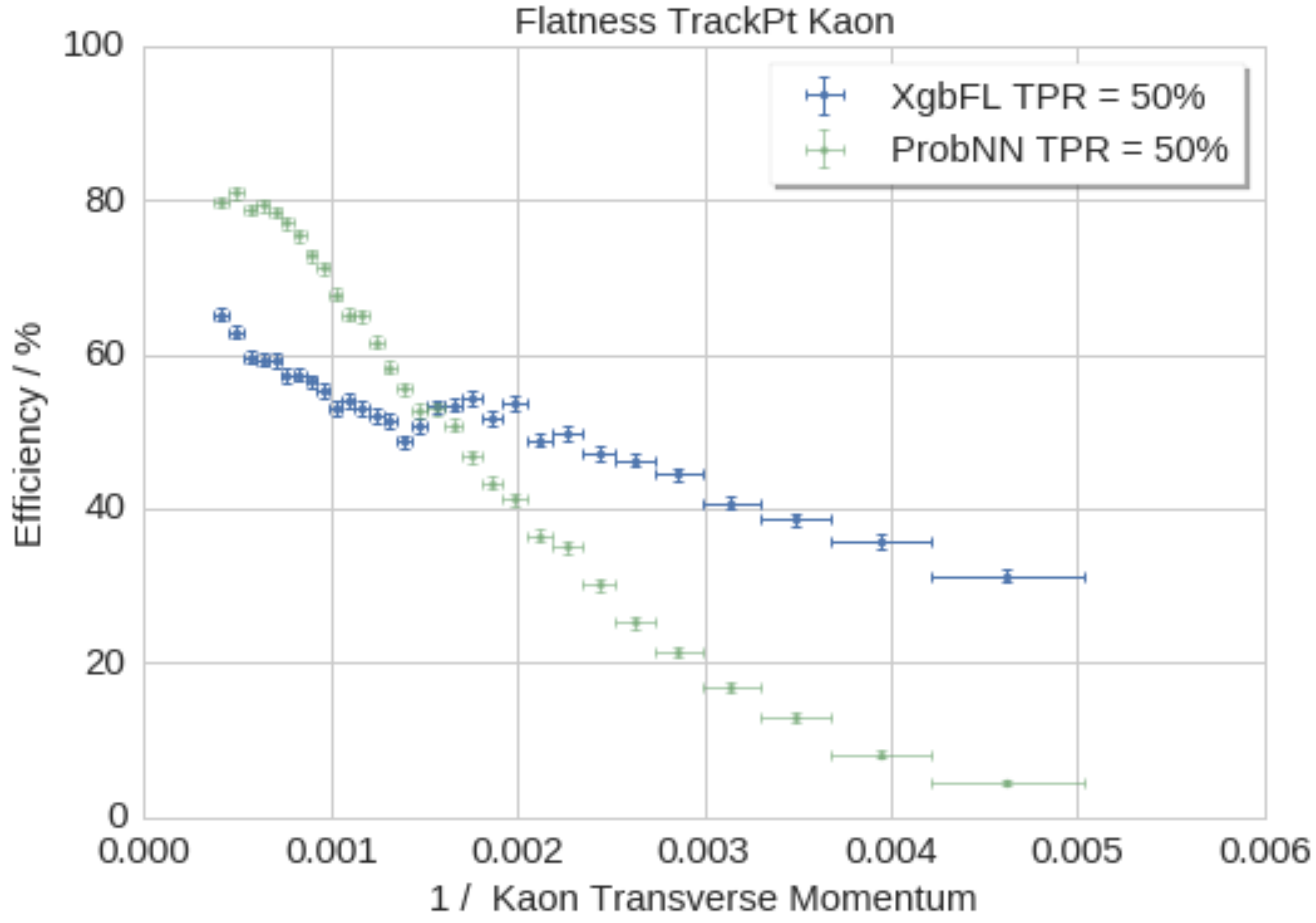
Boosting to uniformity

Uniformity

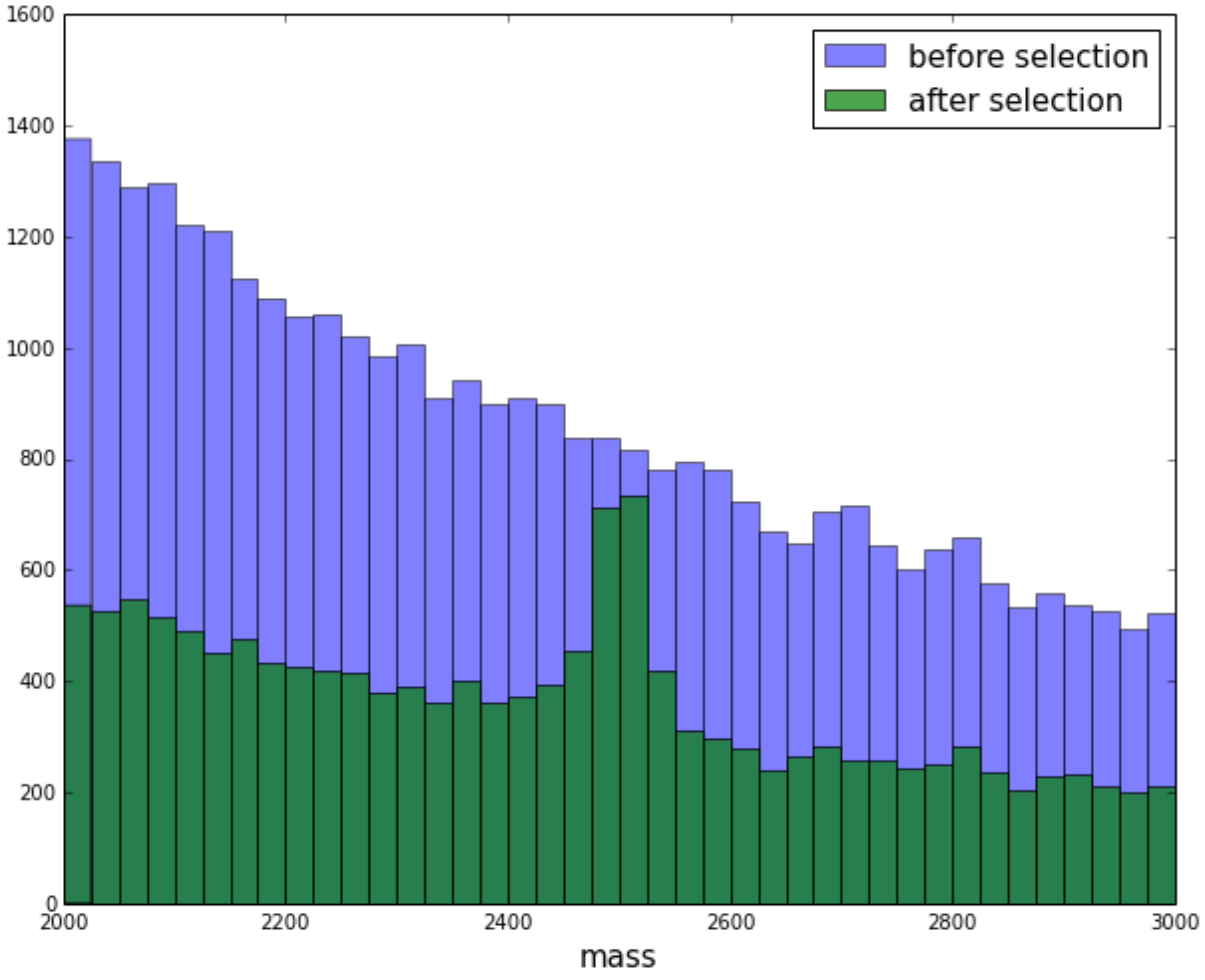
Uniformity means that we have constant efficiency (FPR/TPR) against some variable.

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 loose 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

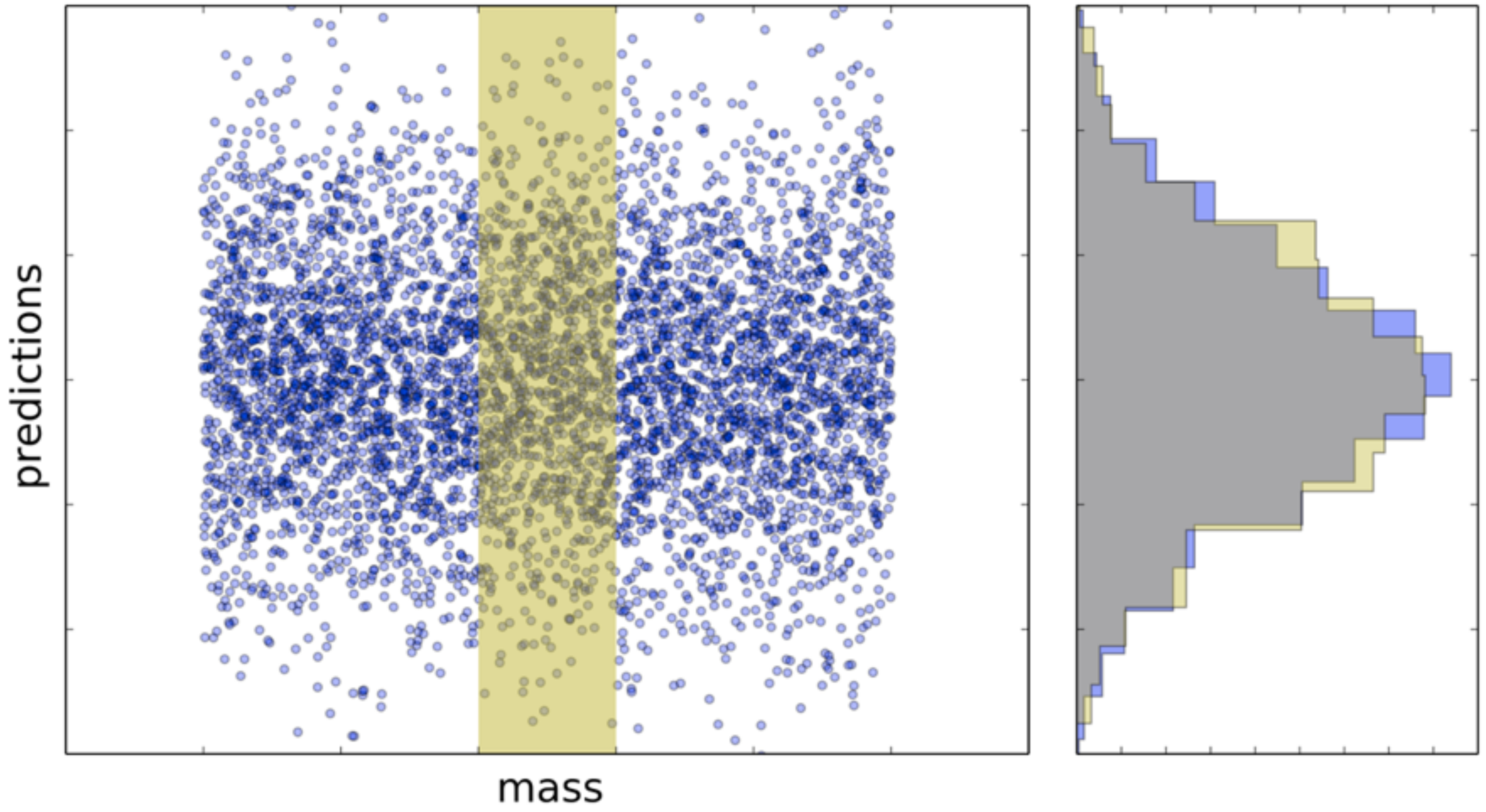
$$D(x) = \sum_j \alpha_j d_j(x)$$

by optimizing some loss function. Those could be:

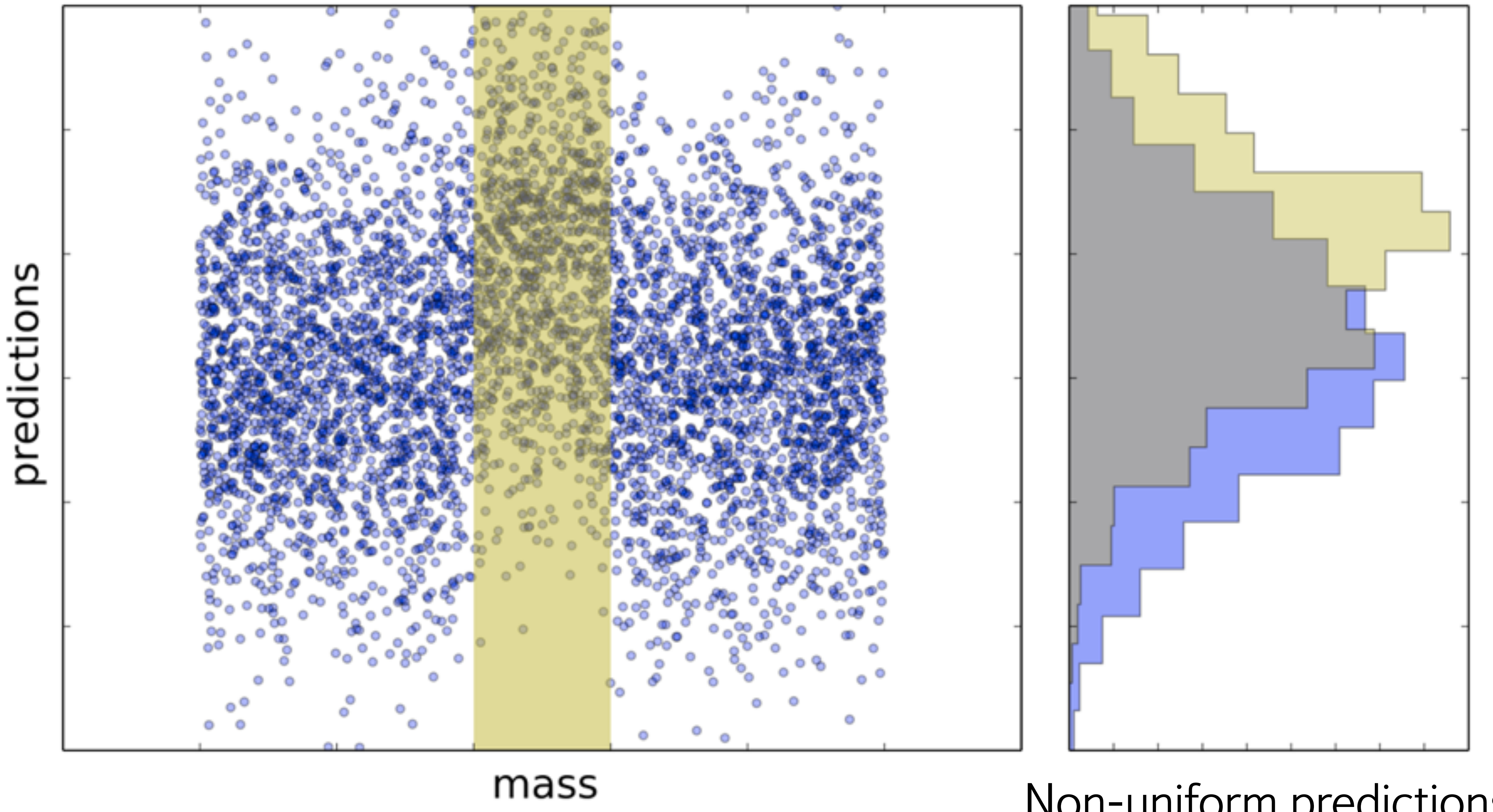
- MSE: $\mathcal{L} = \sum_i (y_i - D(x_i))^2$
- AdaLoss: $\mathcal{L} = \sum_i e^{-y_i D(x_i)}, \qquad y_i = \pm 1$
- LogLoss: $\mathcal{L} = \sum_i \log(1 + e^{-y_i D(x_i)}), \qquad y_i = \pm 1$

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

Non-uniformity measure



Uniform predictions



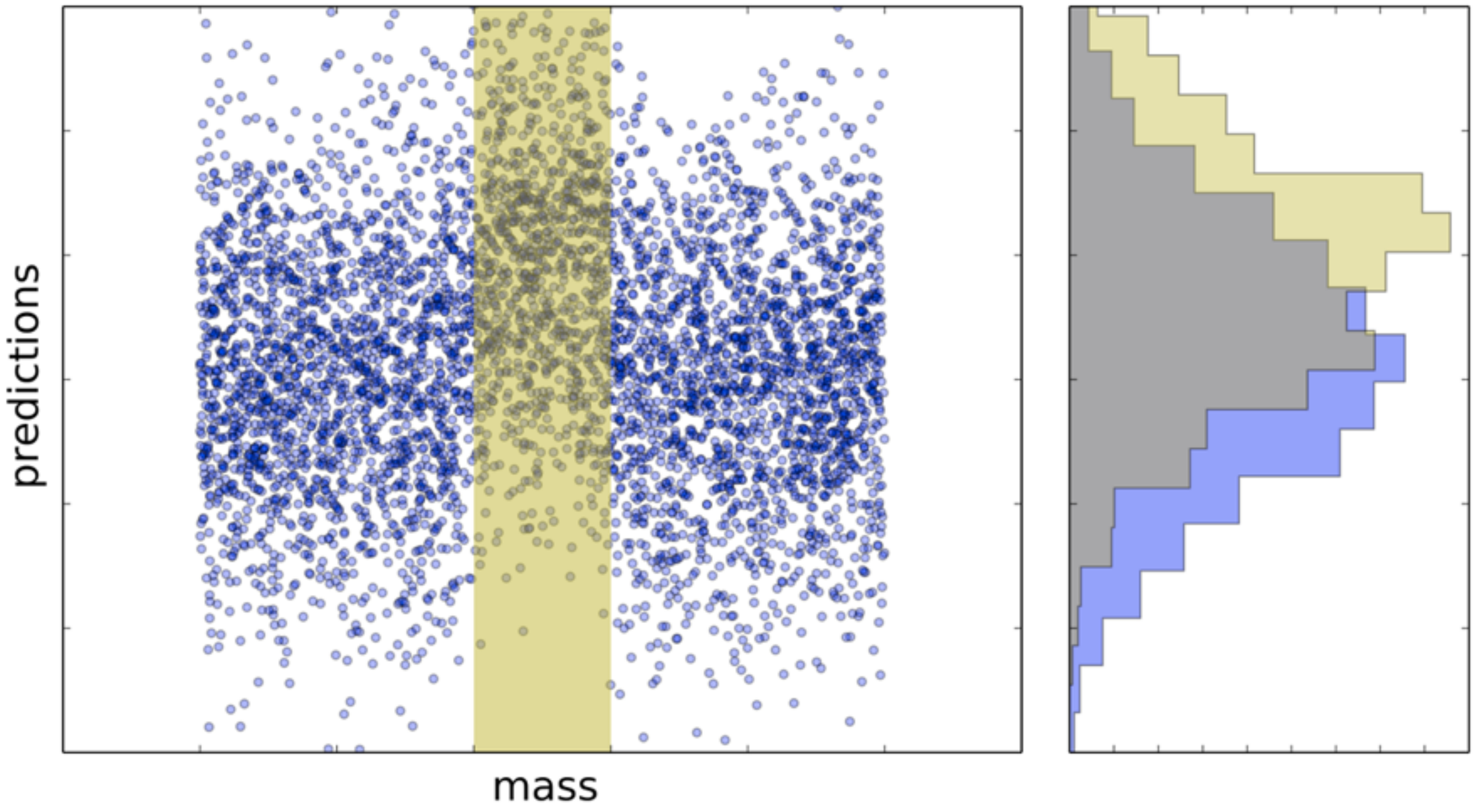
Non-uniform predictions (peak in highlighted region)

- difference in the efficiency can be detected analyzing distributions
- uniformity = no dependence between 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
- 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:

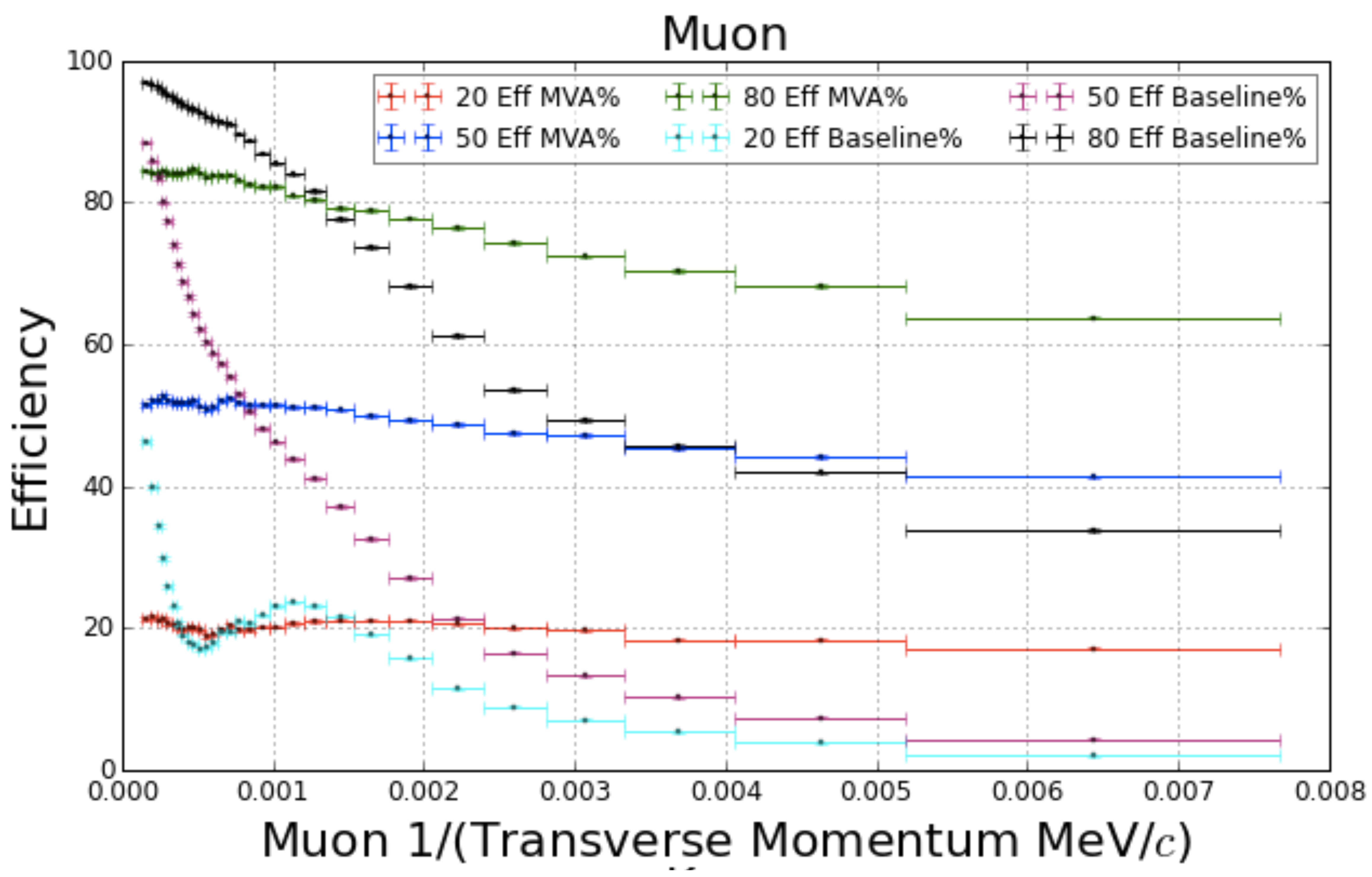
$$\mathcal{L} = \mathcal{L}_{\text{adaloss}} + \alpha \mathcal{L}_{\text{FL}}$$

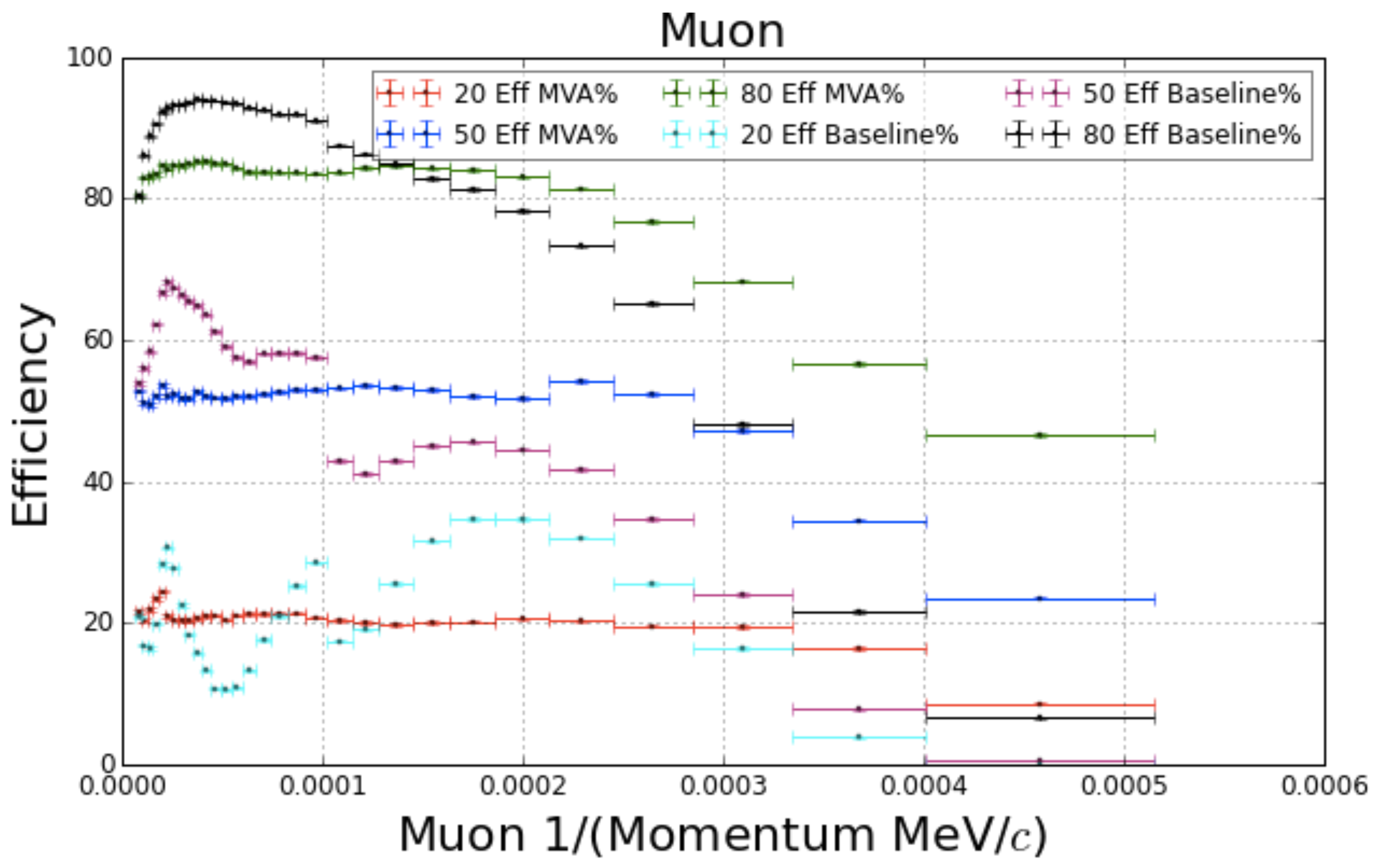
- Flatness loss approximates non-differentiable CvM measure:

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

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

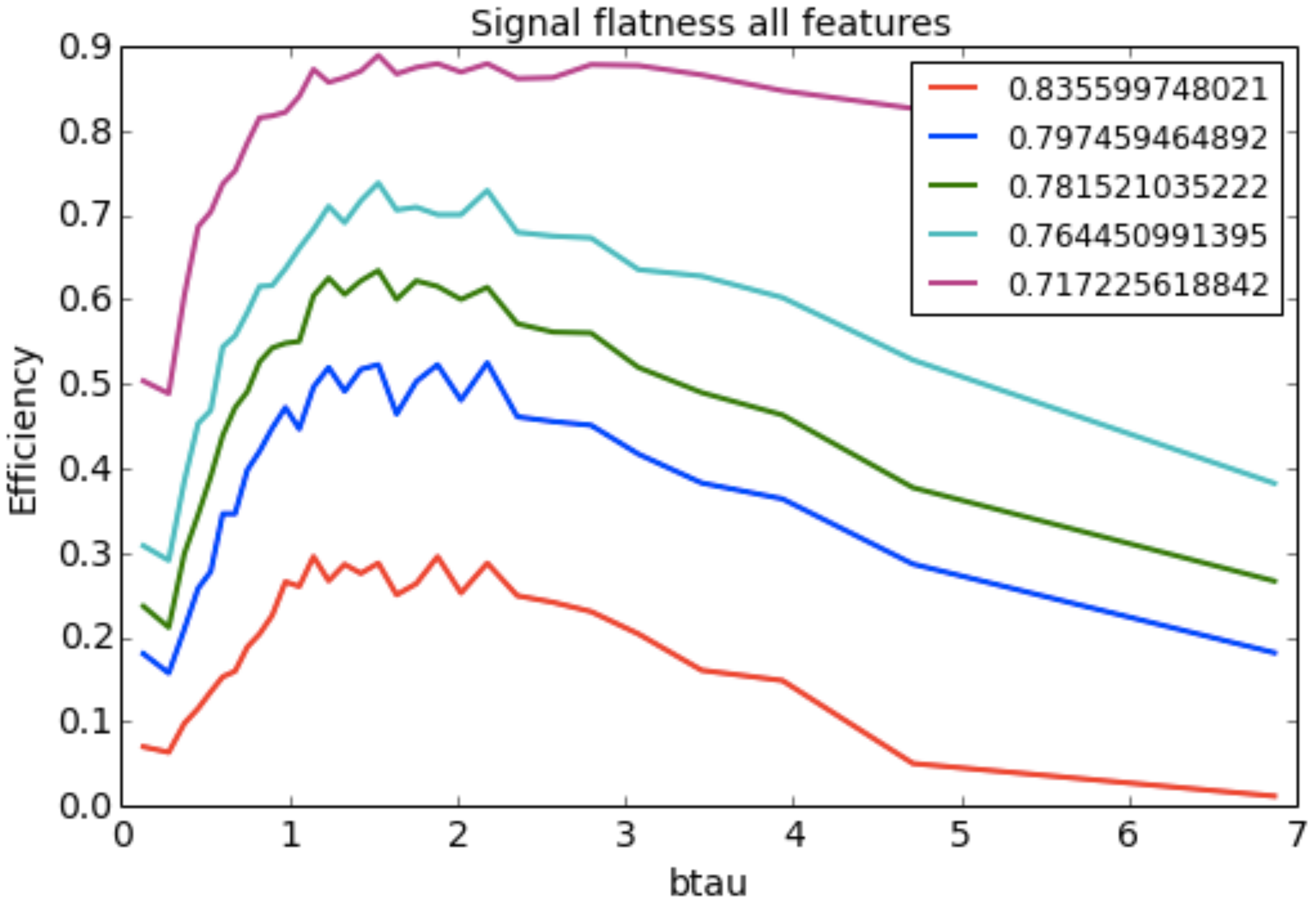
PID DEMO

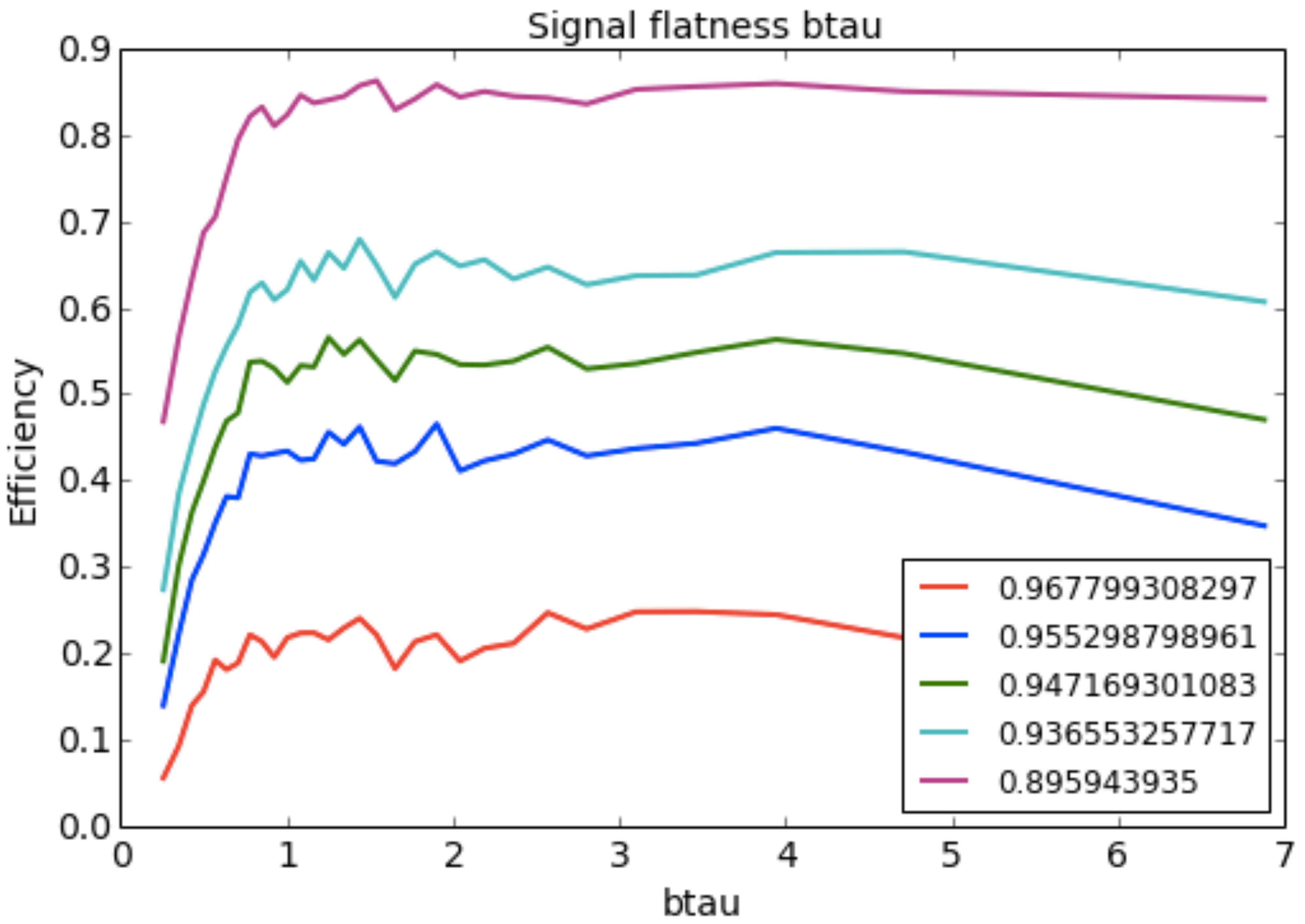




Features strongly depend on the momentum and transverse momentum. Both algorithms use the same set of features. Used MVA is a specific BDT implementation with flatness loss.

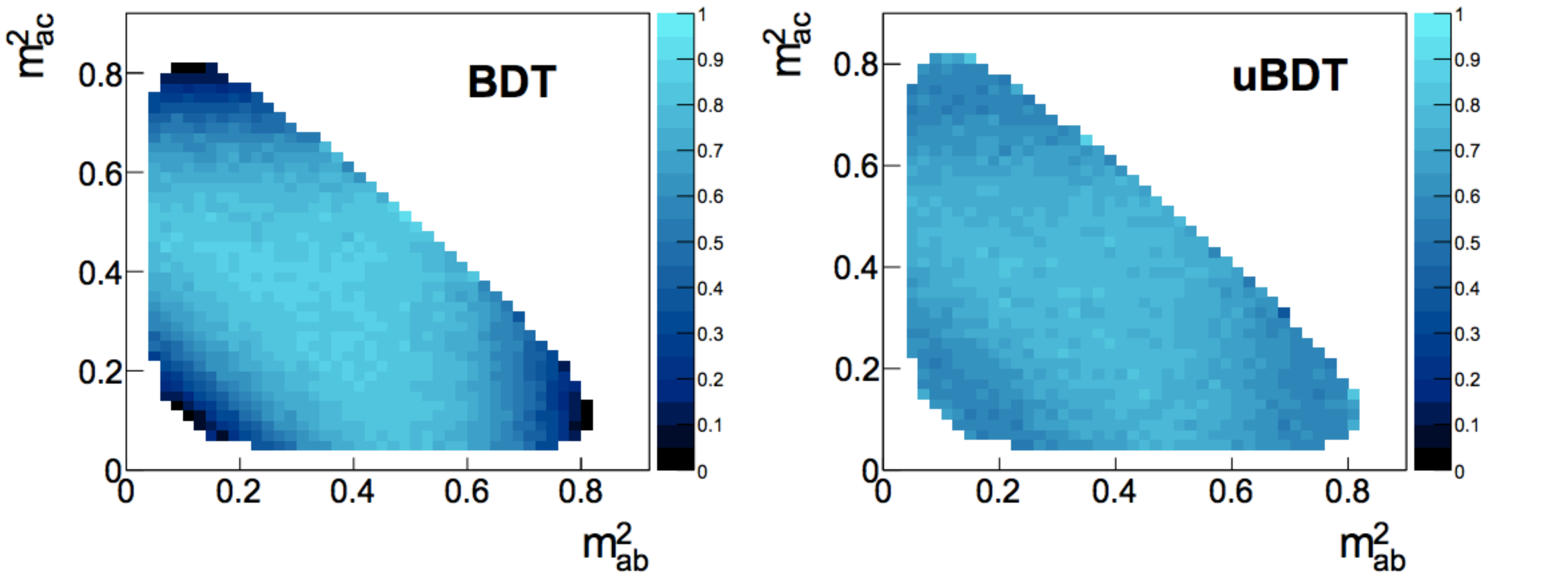
Trigger DEMO





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

Boosting summary

- powerful general-purpose algorithm
- most known applications: classification, regression and ranking
- widely used, considered to be well-studied
- can be adapted to different specific scientific problems

Thanks for attention