SCHOOL OF DATA ANALYSIS

ML in HEP: Aspects and Applications

Tatiana Likhomanenko

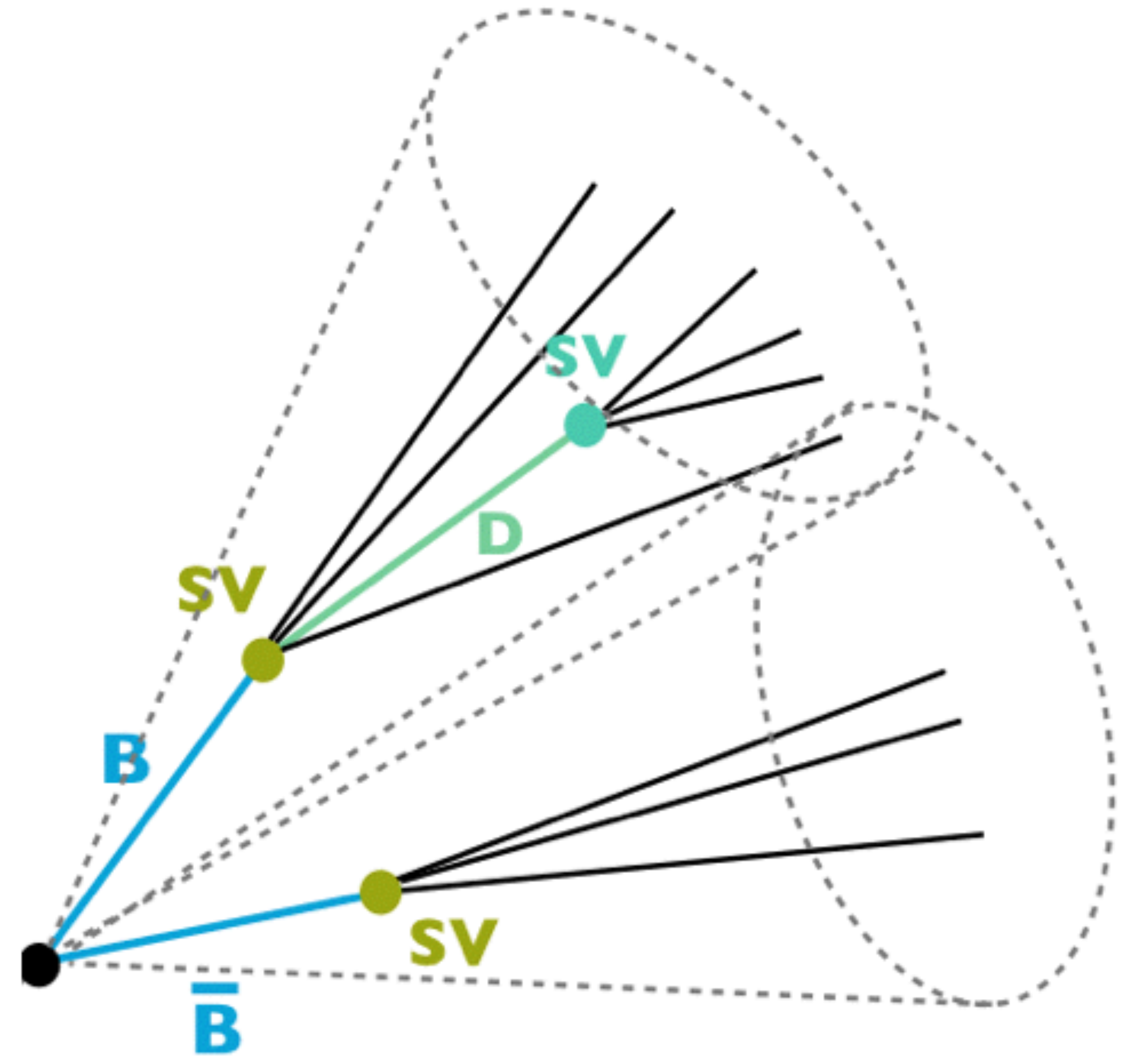
Introduction

ML in HEP

- High-level triggers
- Calibration (energy in the calorimeter)
- Particle identification
- Stripping line (rarely)
- Tagging (jet-tagging, B-tagging)
- Analysis (rare decays, ...)

LHC Event

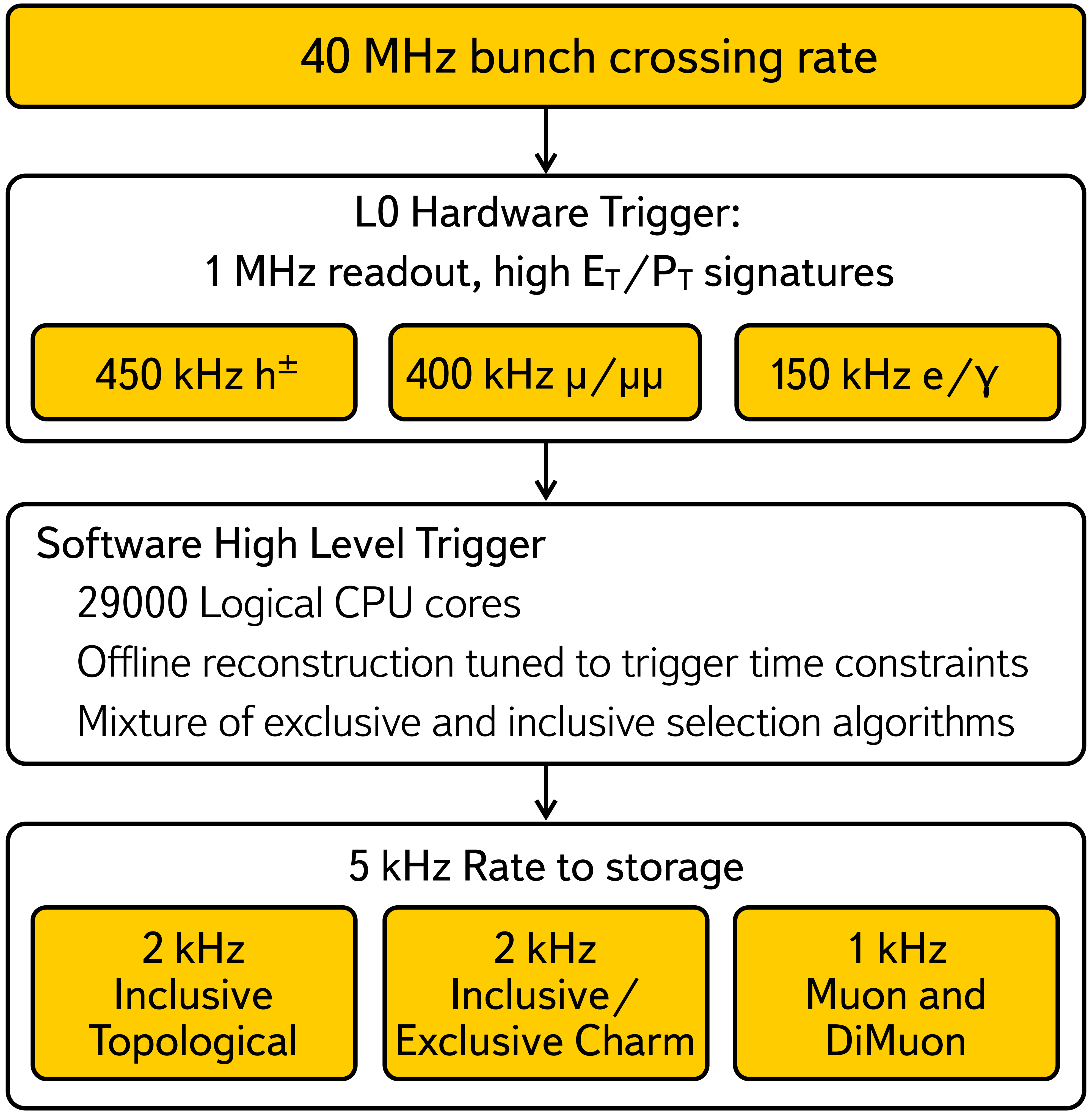
- Sample: one proton-proton bunches collision
- Event consists of:
  - tracks (track description)
  - secondary vertices (SV description)
- Questions:
  - How to describe event in ML terms?
  - How to train model on such samples?



Topological trigger

LHCb trigger system

- Select events to store them for offline processing
- Should efficiently select interesting events
- An event is interesting if it contains at least one interesting SV
- Output rate for trigger system is limited

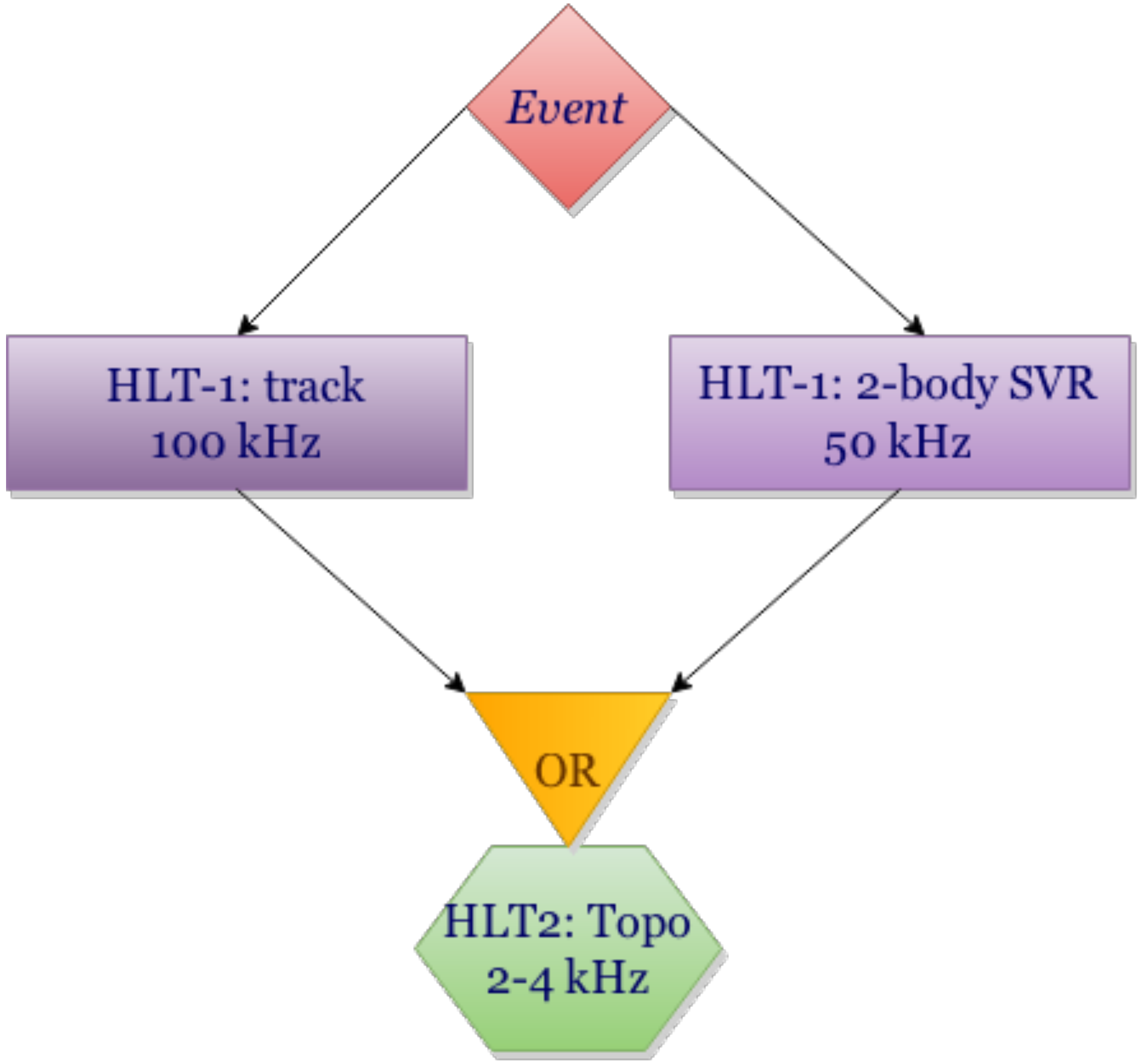


LHCb topological trigger

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

Run-II LHCb topological trigger

- HLT-1 track is looking for either one super high PT or high displacement track
- HLT-1 2-body SV classifier is looking for two tracks making a vertex
- HLT-2 improved topo classifier uses full reconstructed event to look for 2, 3, 4 and more tracks making a vertex

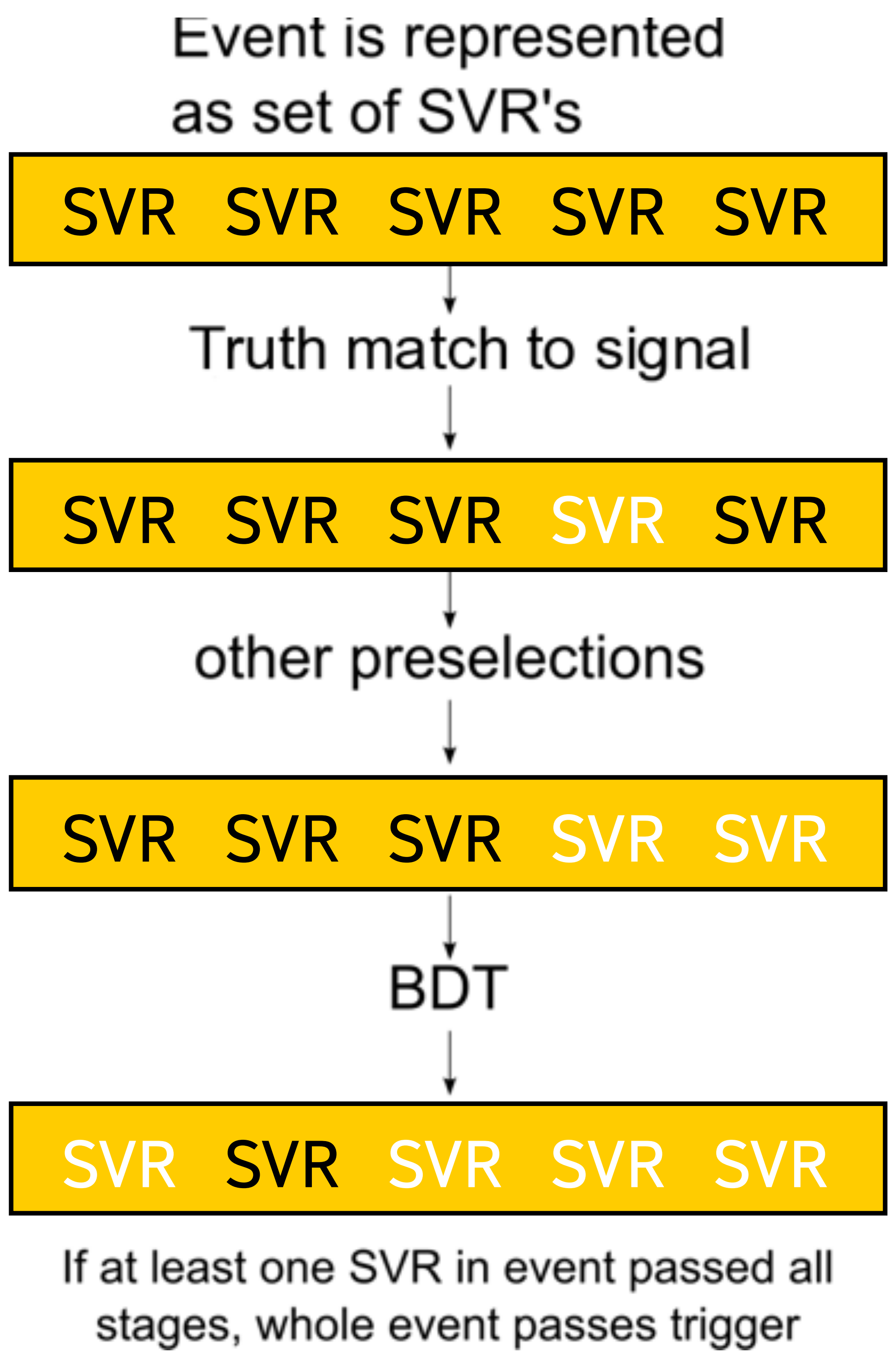


N body tracks

- Two, three or four tracks are combined to form a SVR
- Each secondary vertex in Monte Carlo data is preselected in such way, that all tracks must be matched to particles from the signal decay (true match preselection)

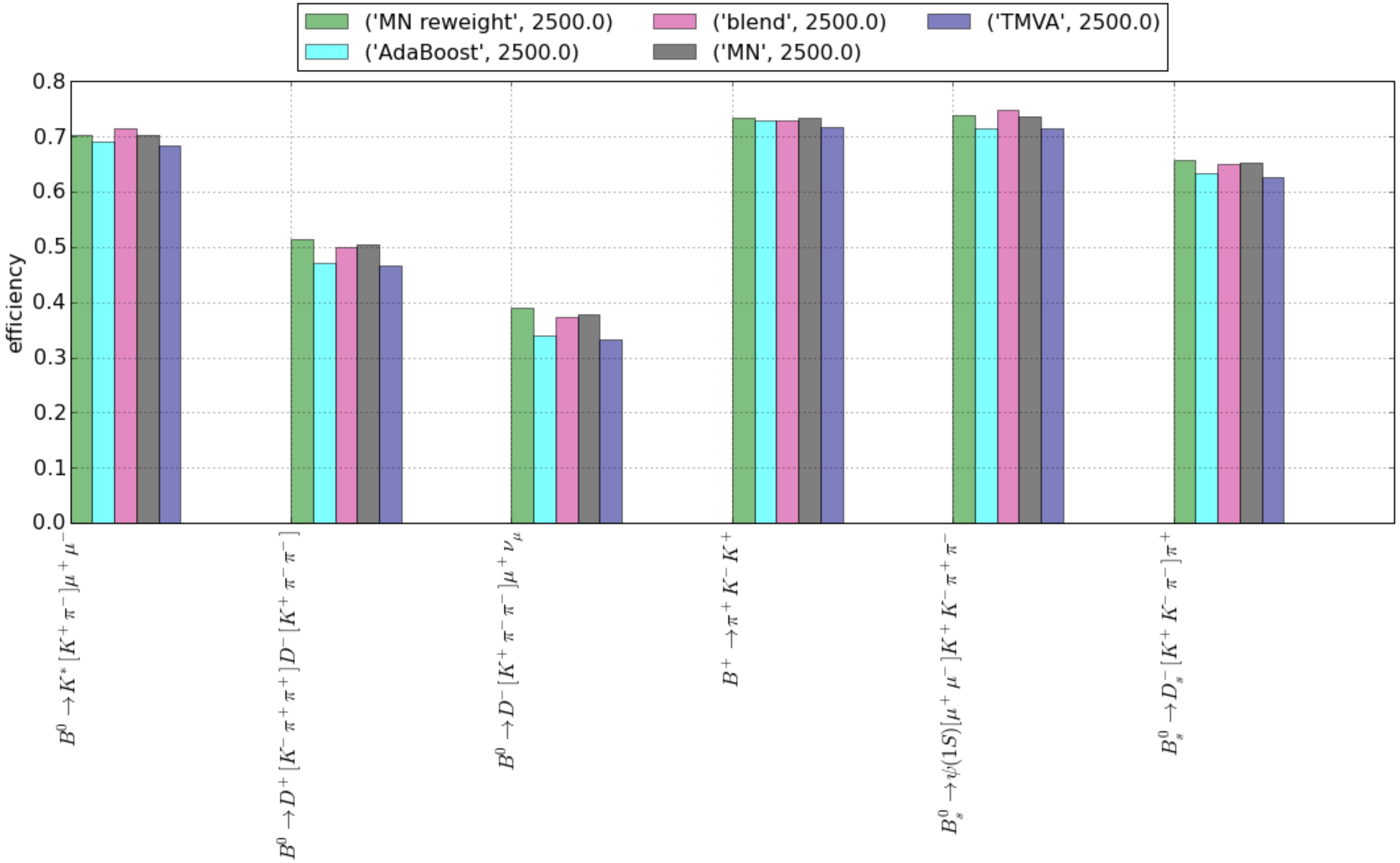
Data

- Monte Carlo samples (used as signal-like) are simulated 13-TeV B decays of various topologies
- Generic Pythia 13-TeV proton-proton collisions are used as background-like sample
- 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



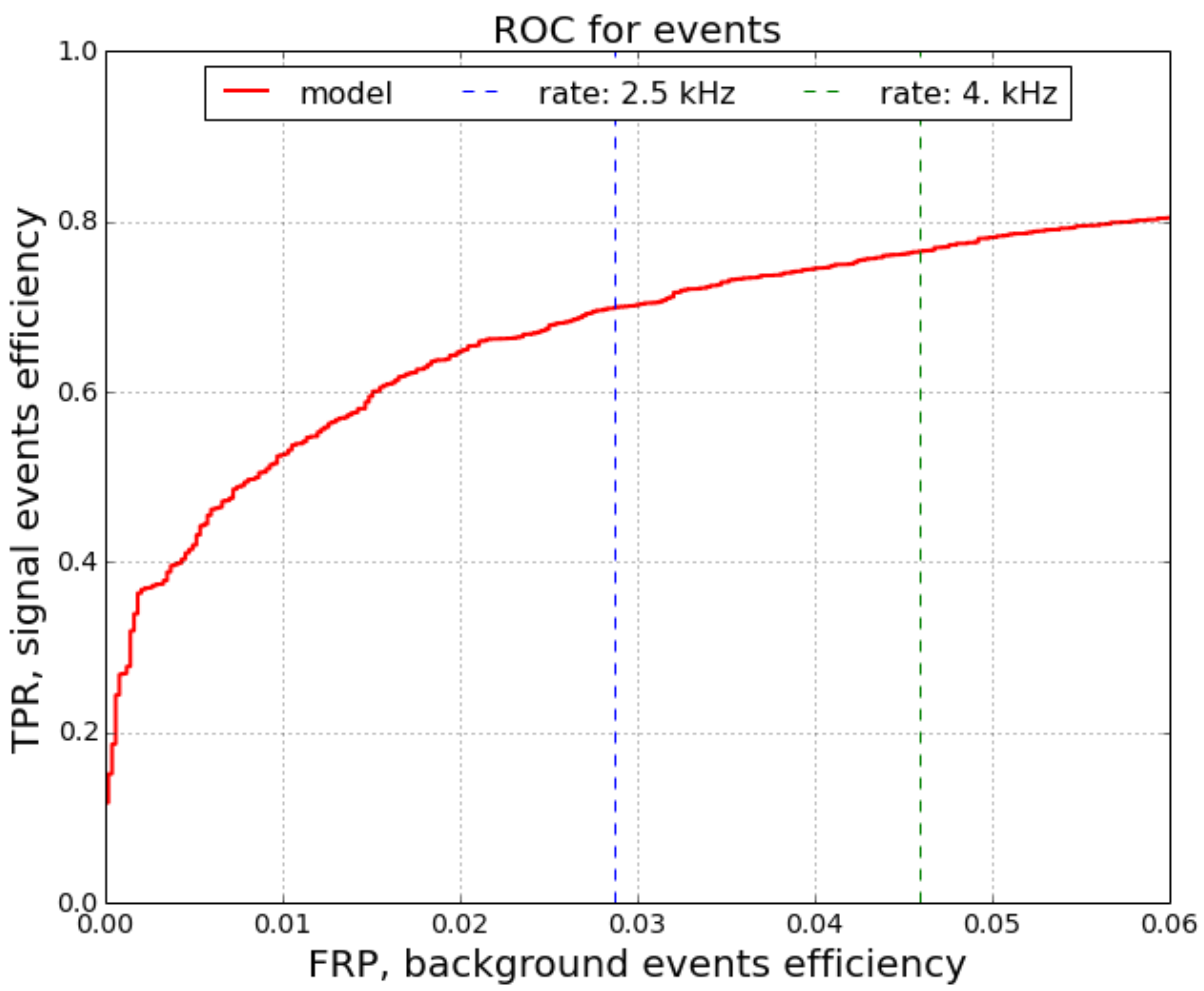
Event representation

How to measure quality?



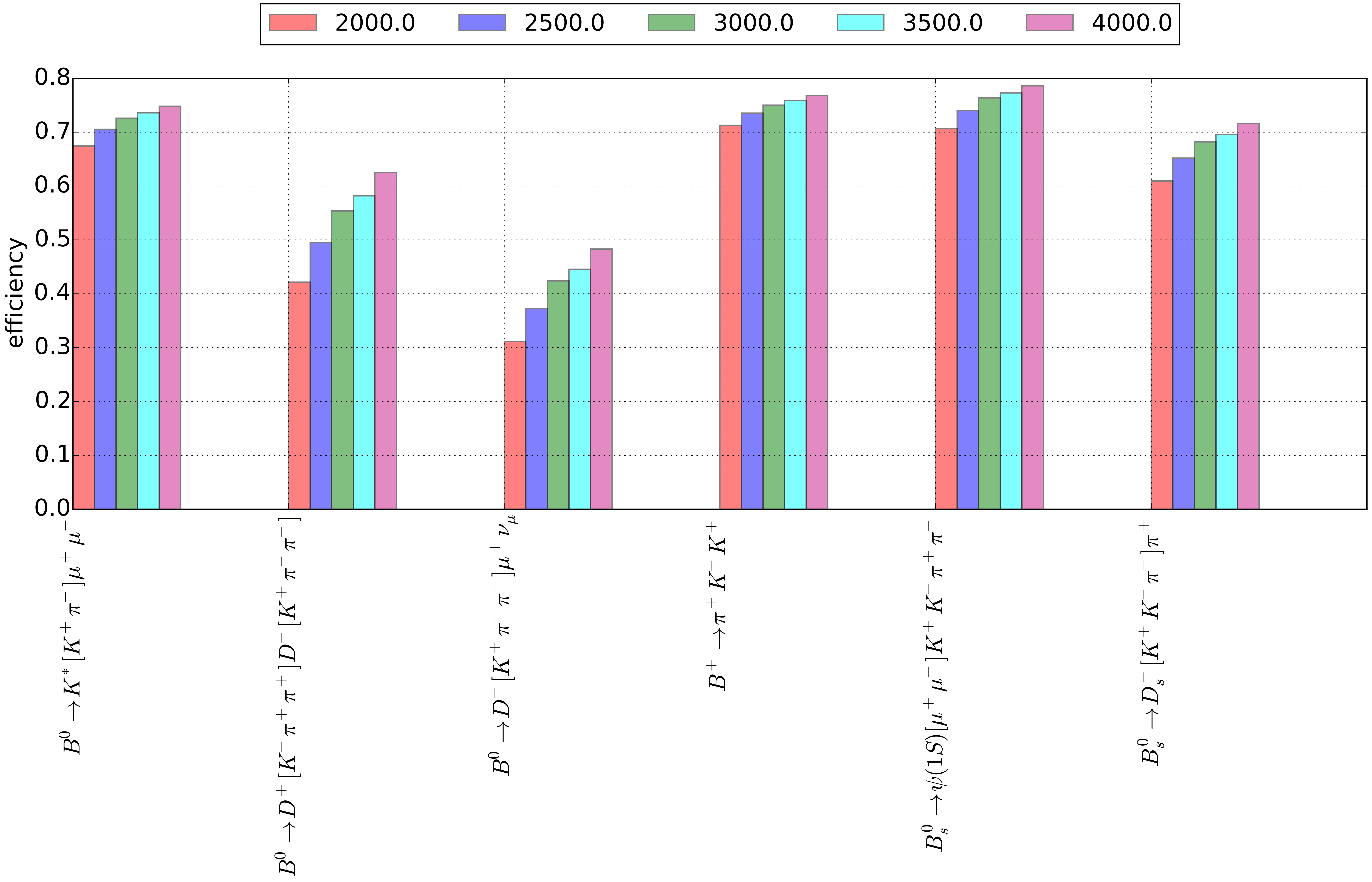
ROC curve, computed for events

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



ROC curve interpretation

Dependence on output rate



Hierarchical training

- Train separate models for:
  - each channel
  - each n-body type: 2, 3, 4
- Use them as additional features later
- Use folding scheme to train additional features or to apply additional classifier selections

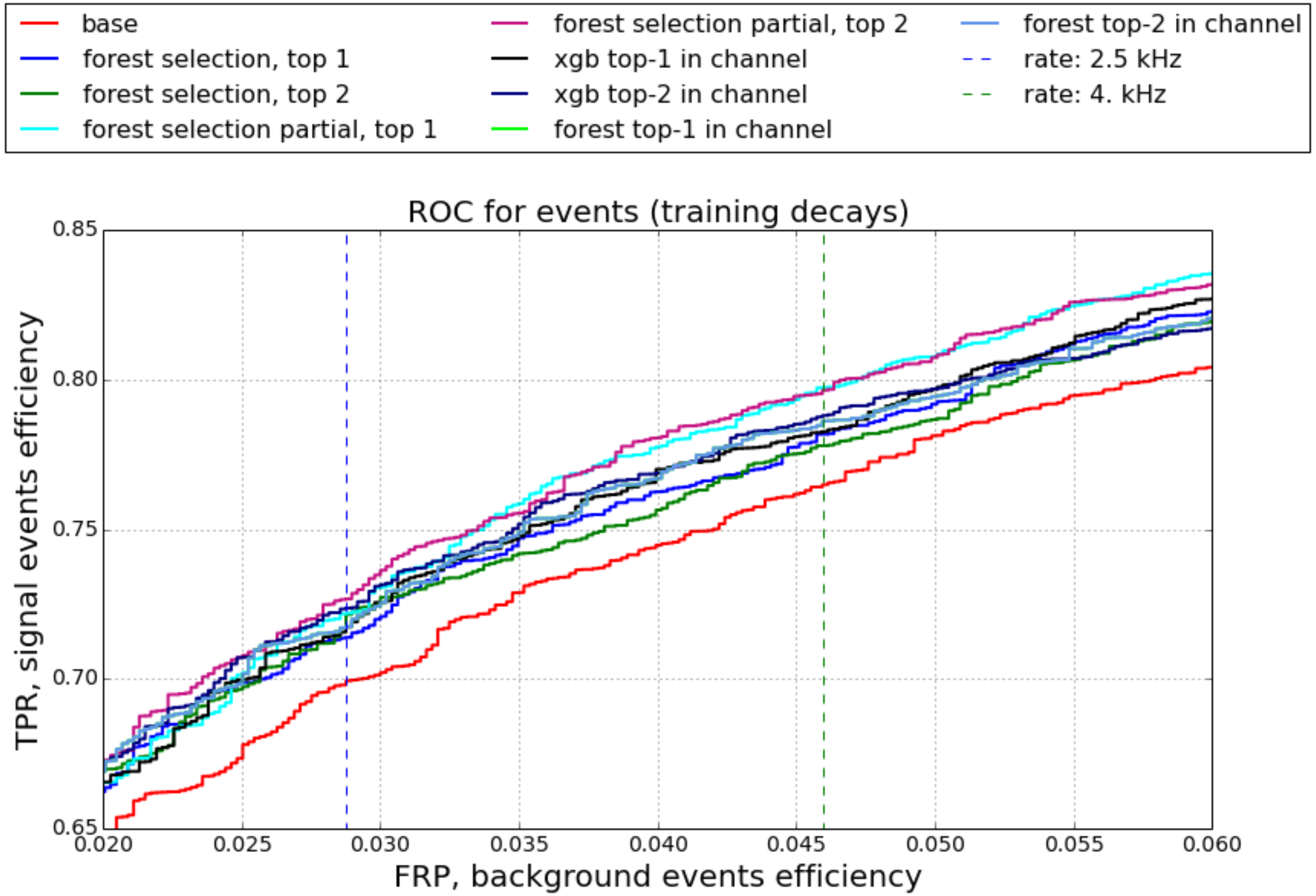
Folding scheme

Simulated signal event contains at least one interesting SV, but not each SV should be interesting

Random forest for SVs selection

- Train random forest (RF) on SVs using folding scheme
  - 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
- Try another algorithm instead of RF, maybe it will work!

Random forest for SVs selection



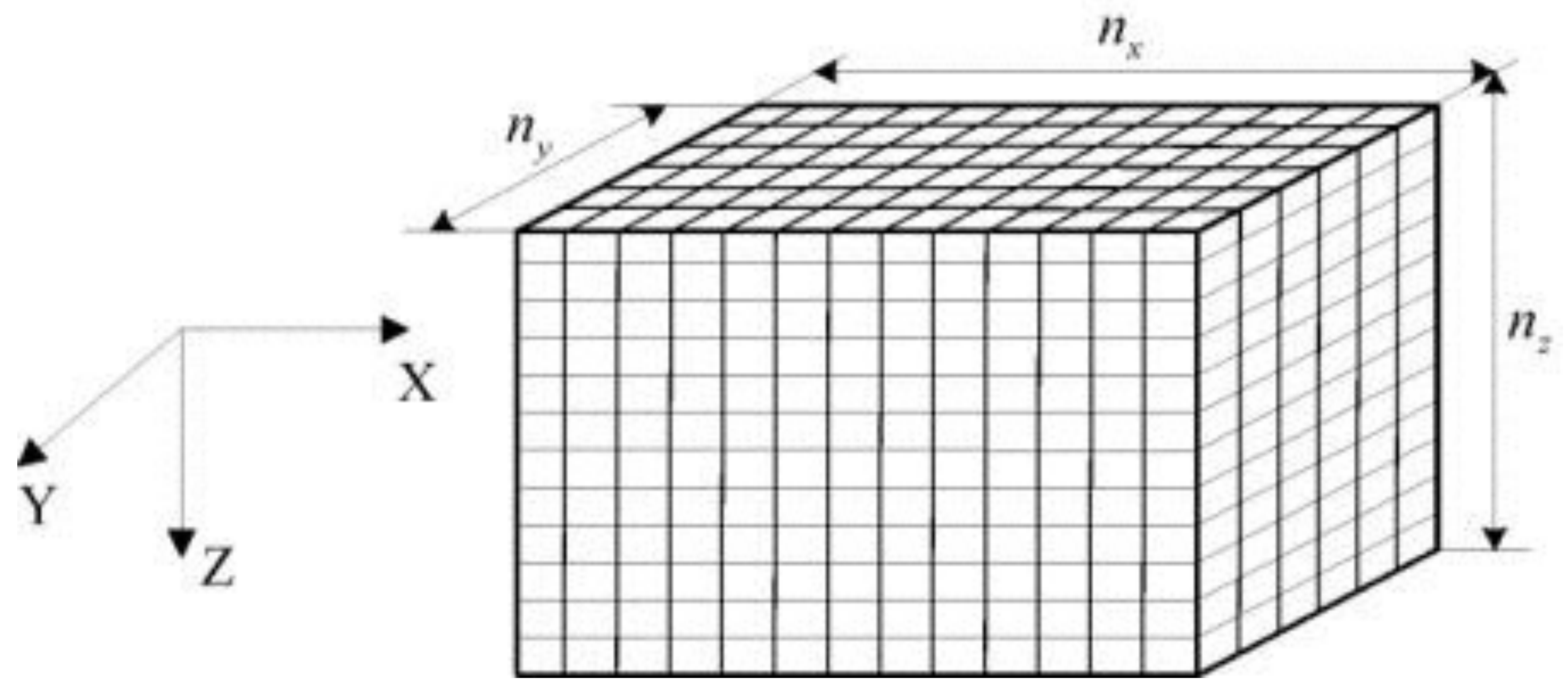
Online processing

There are two possibilities to speed up prediction operation:

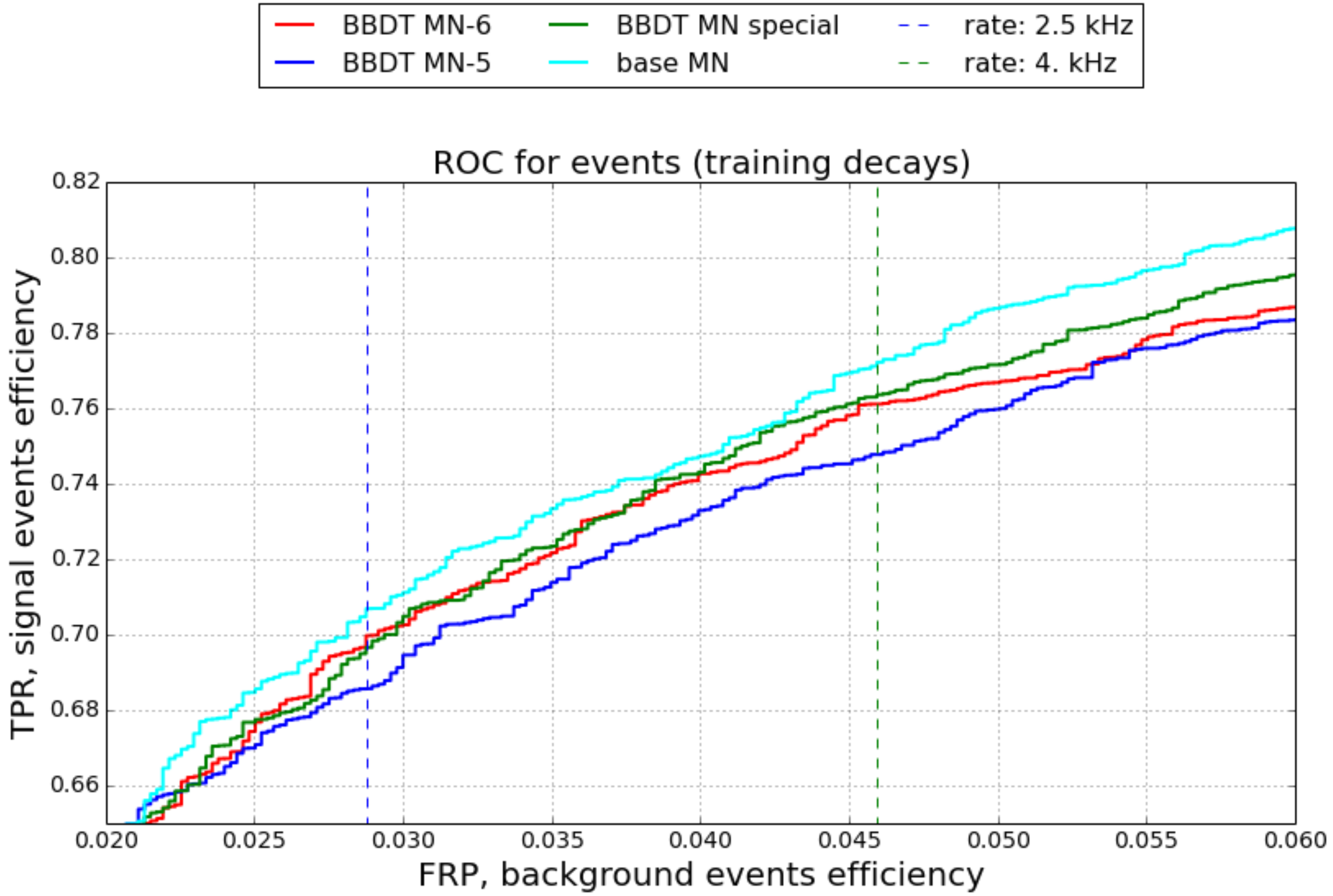
- Bonsai boosted decision tree format (BBDT)
- Post-pruning

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



BBDT, results



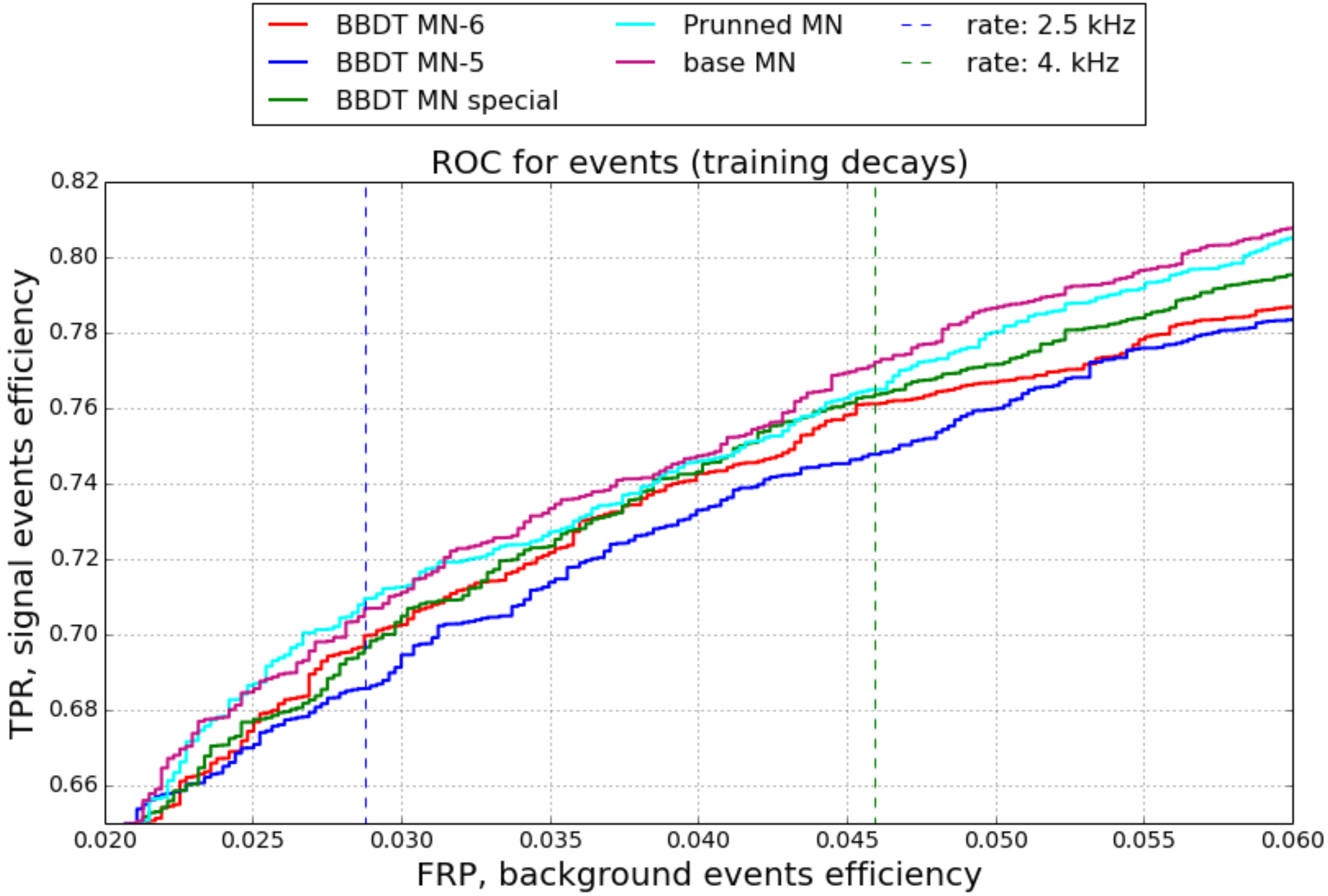
Post-pruning

- Train GB over oblivious trees with several thousands trees
- Reduce this amount of trees to a hundred
- Greedily choose trees in a sequence from the initial ensemble to minimize a modified loss function:

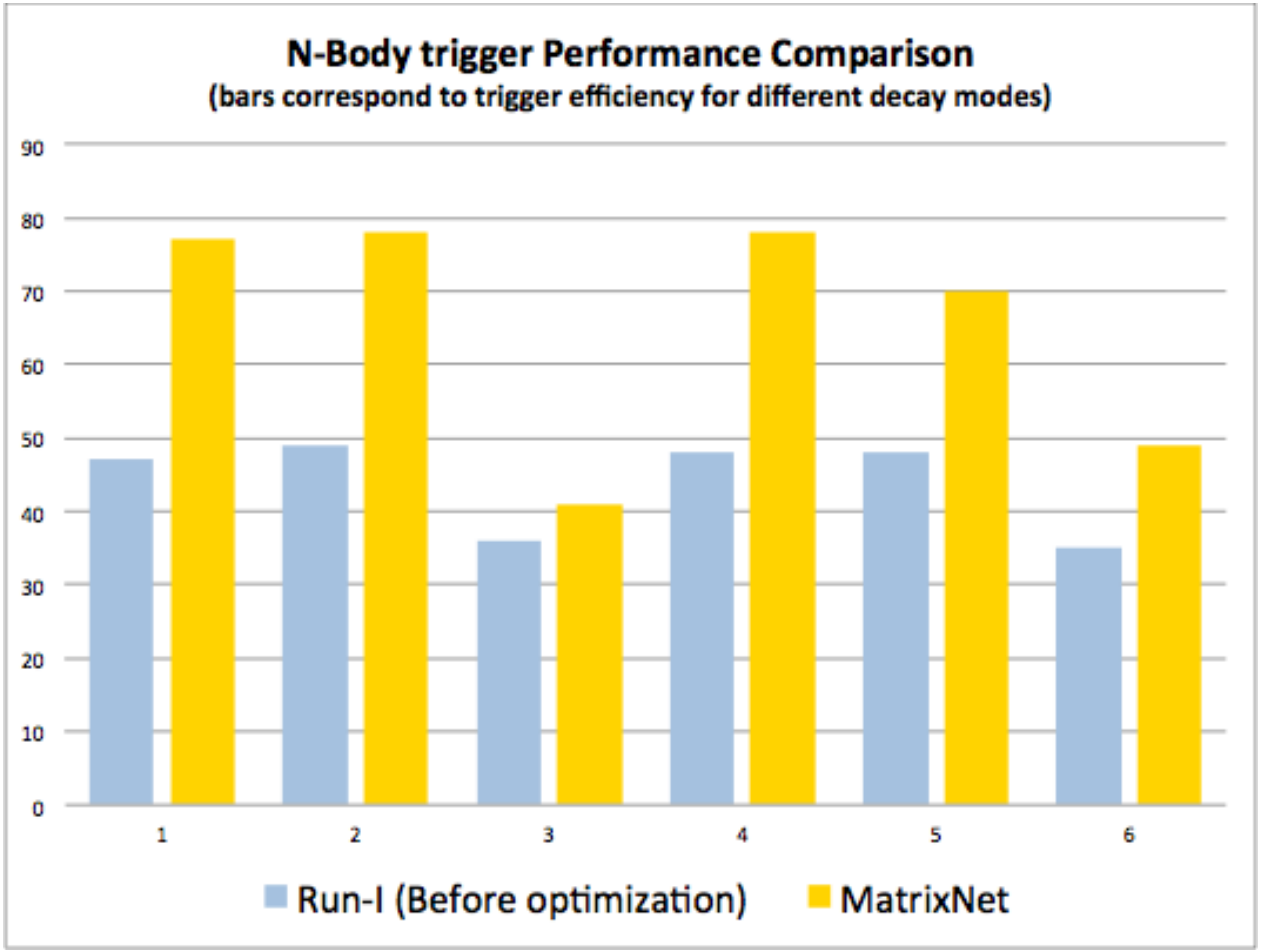
$$\sum_{signal} \log\left(1 + e^{-F(x)}\right) + \sum_{background} e^{F(x)}$$

- At the same time change values in leaves (tree structure is preserved)

Post-pruning, results



Topological trigger results (without RF trick)



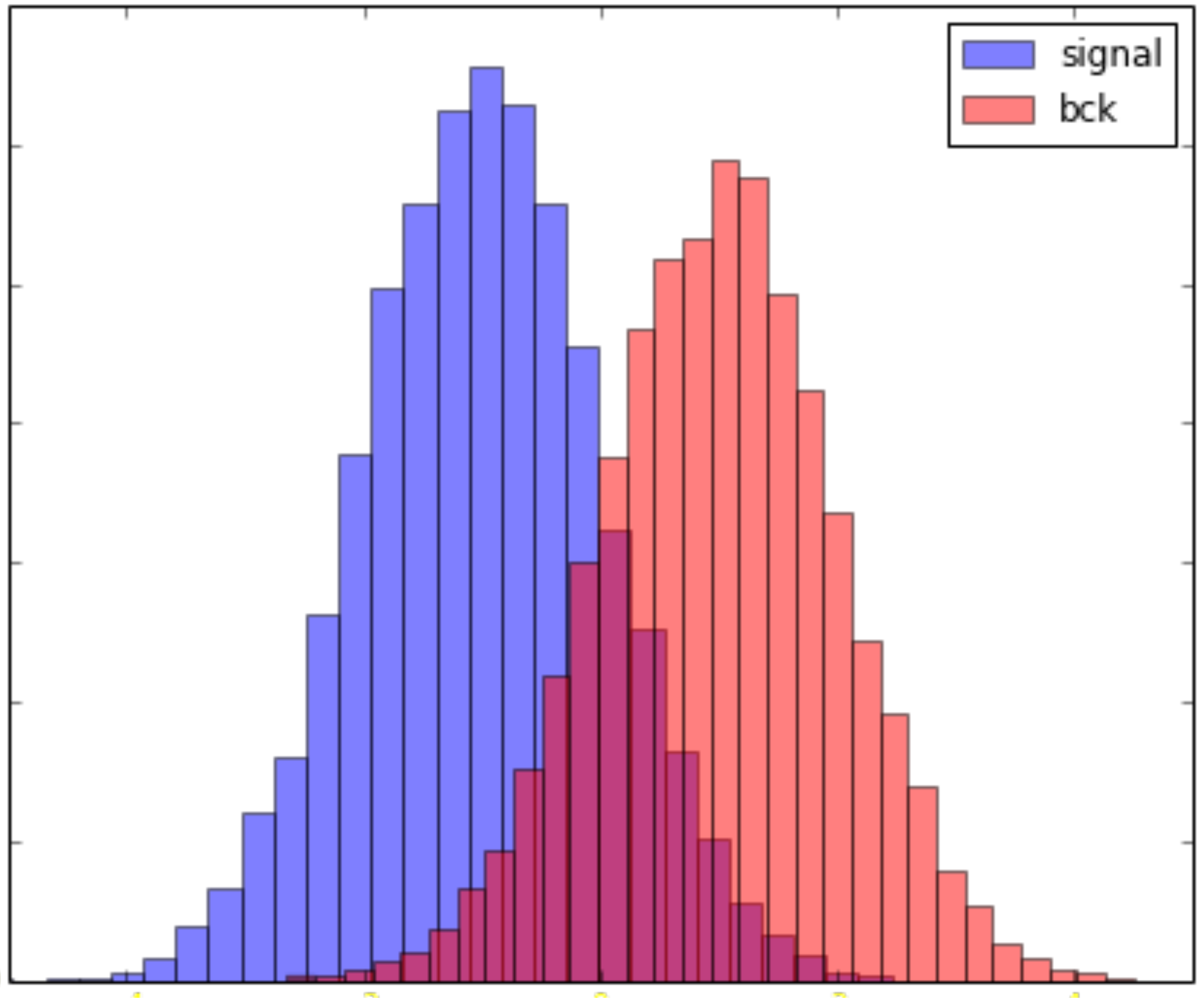
https://github.com/yandexdataschool/LHCb-topo-trigger

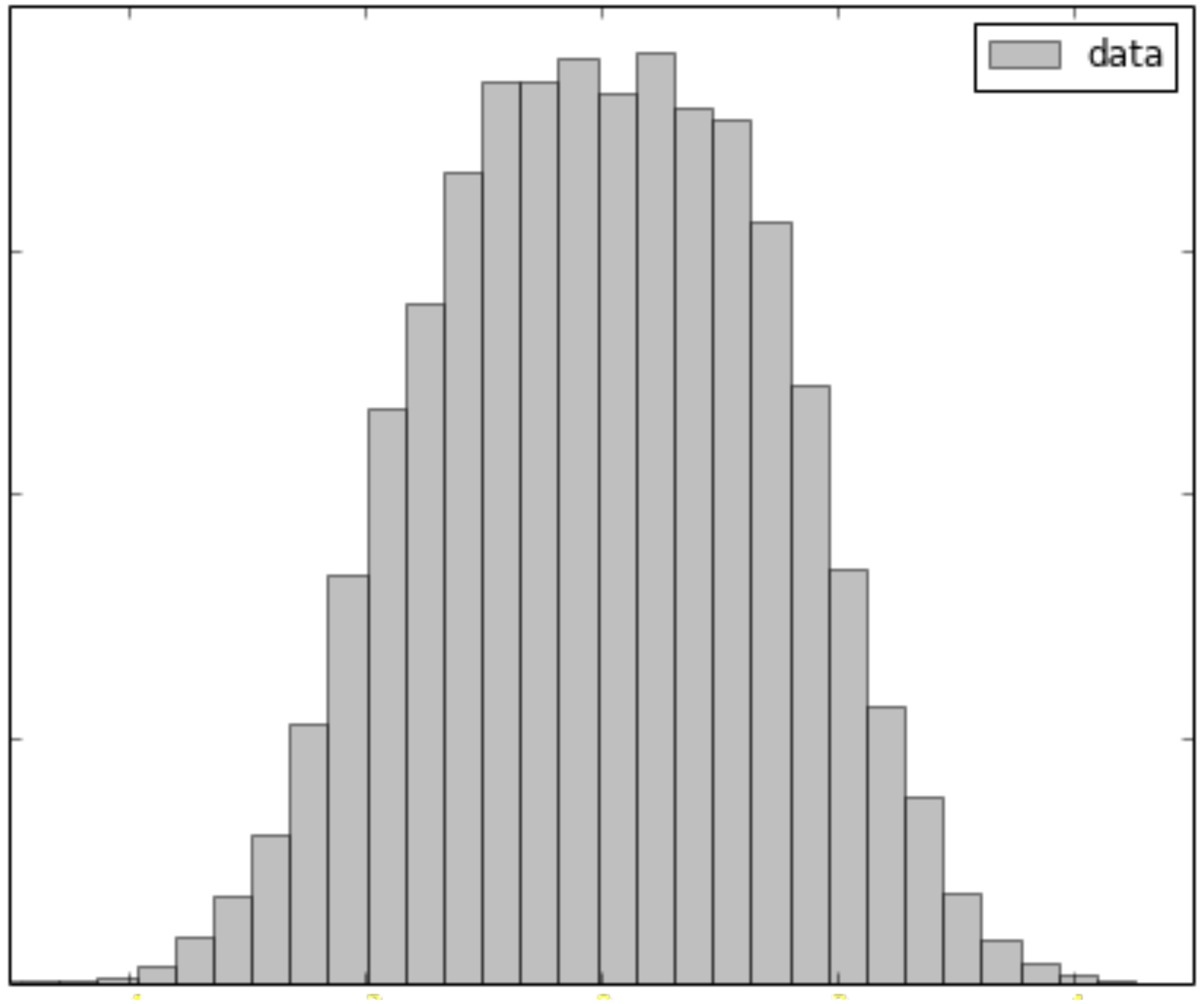
sPlot technique

Why using it?

- Monte Carlo is not-well simulated
- Need to work with real unlabeled data
- Need somehow to label real data: want to restore for features their distributions for the signal and background data
- Our main knowledge is the mass distribution for real data from which we can extract the mass pdfs for signal and background.
- How to restore signal/bck pdfs for other features?

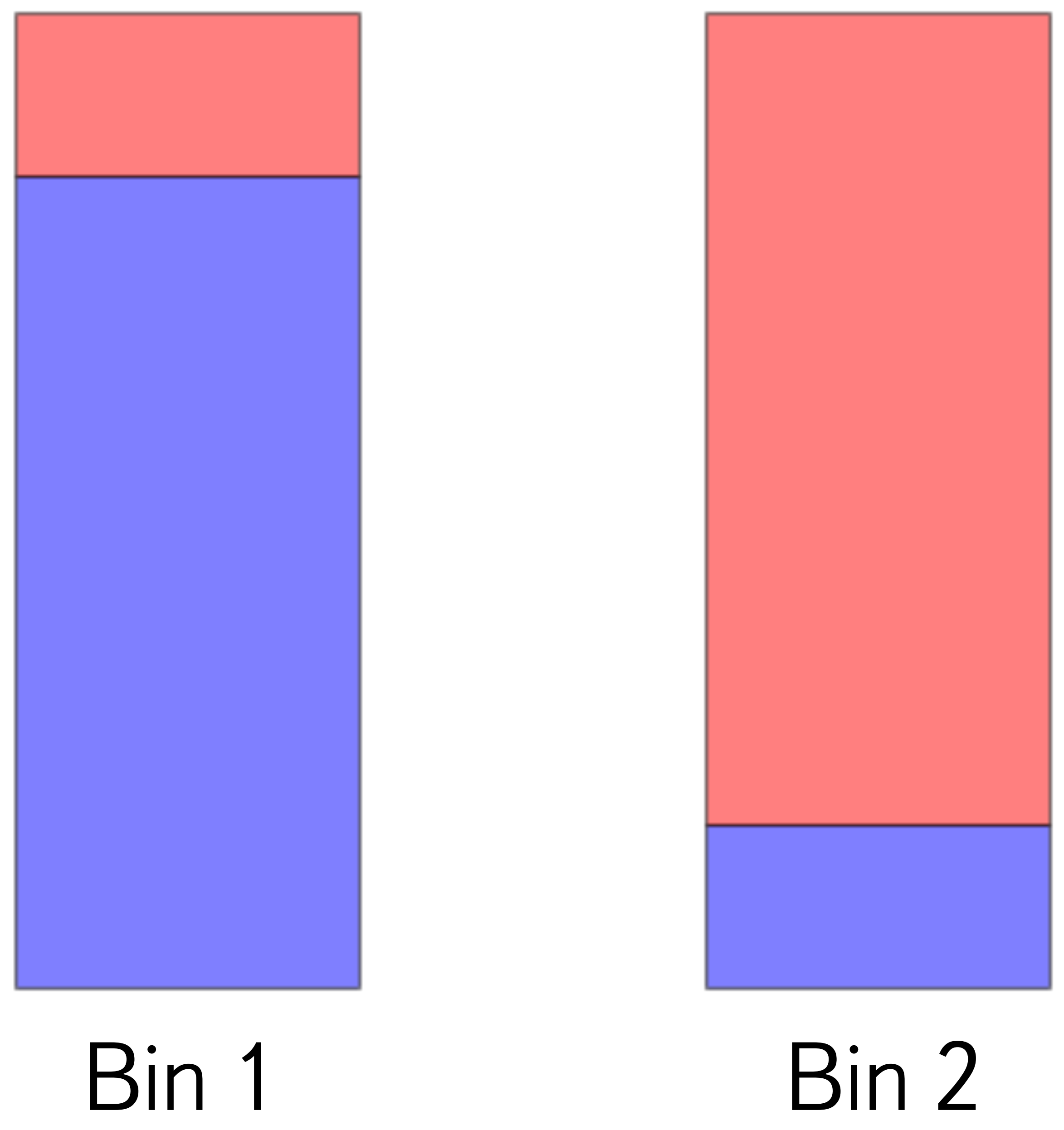
Feature initial distributions

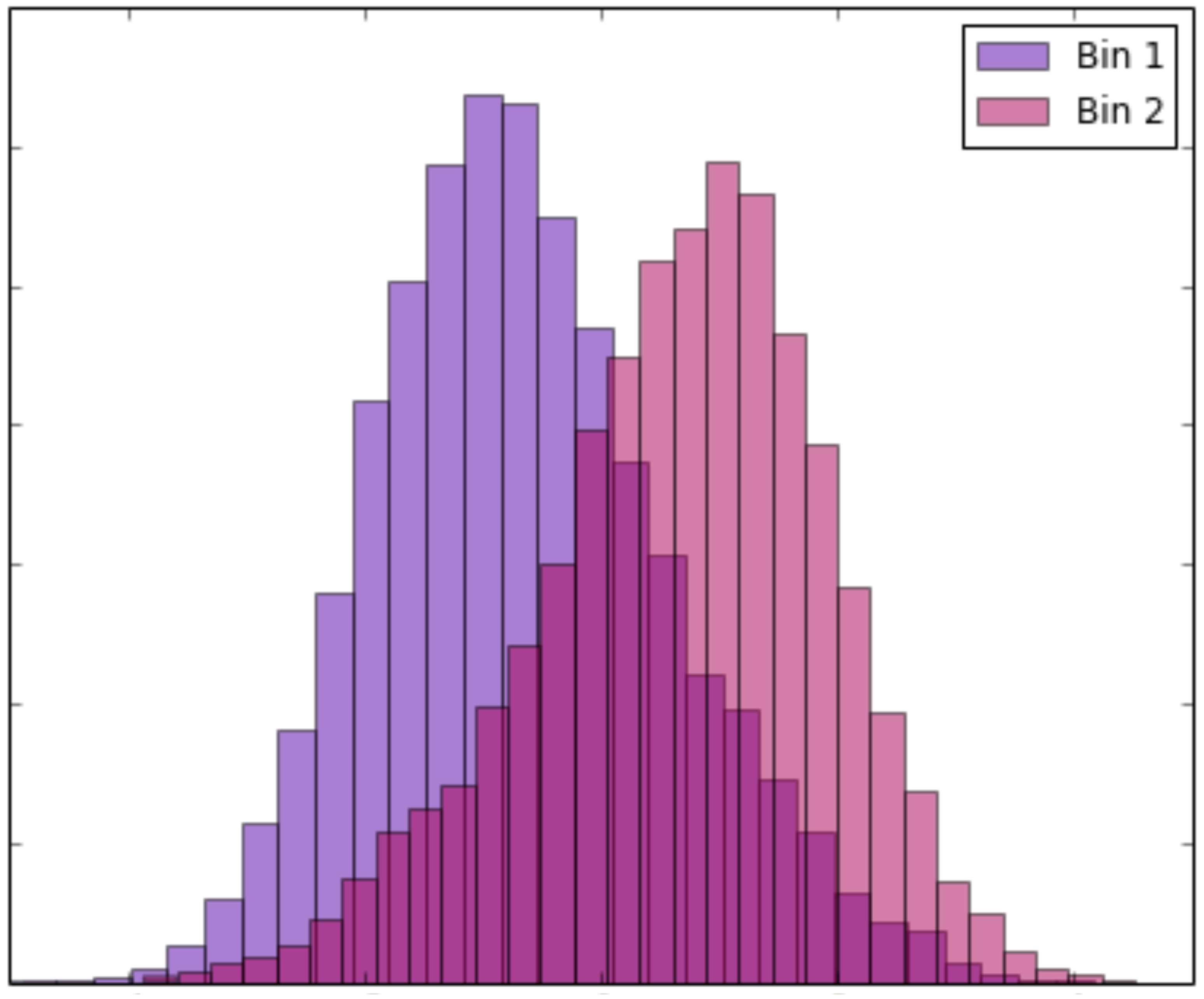




Two mass bins

Proportion of events inside bins

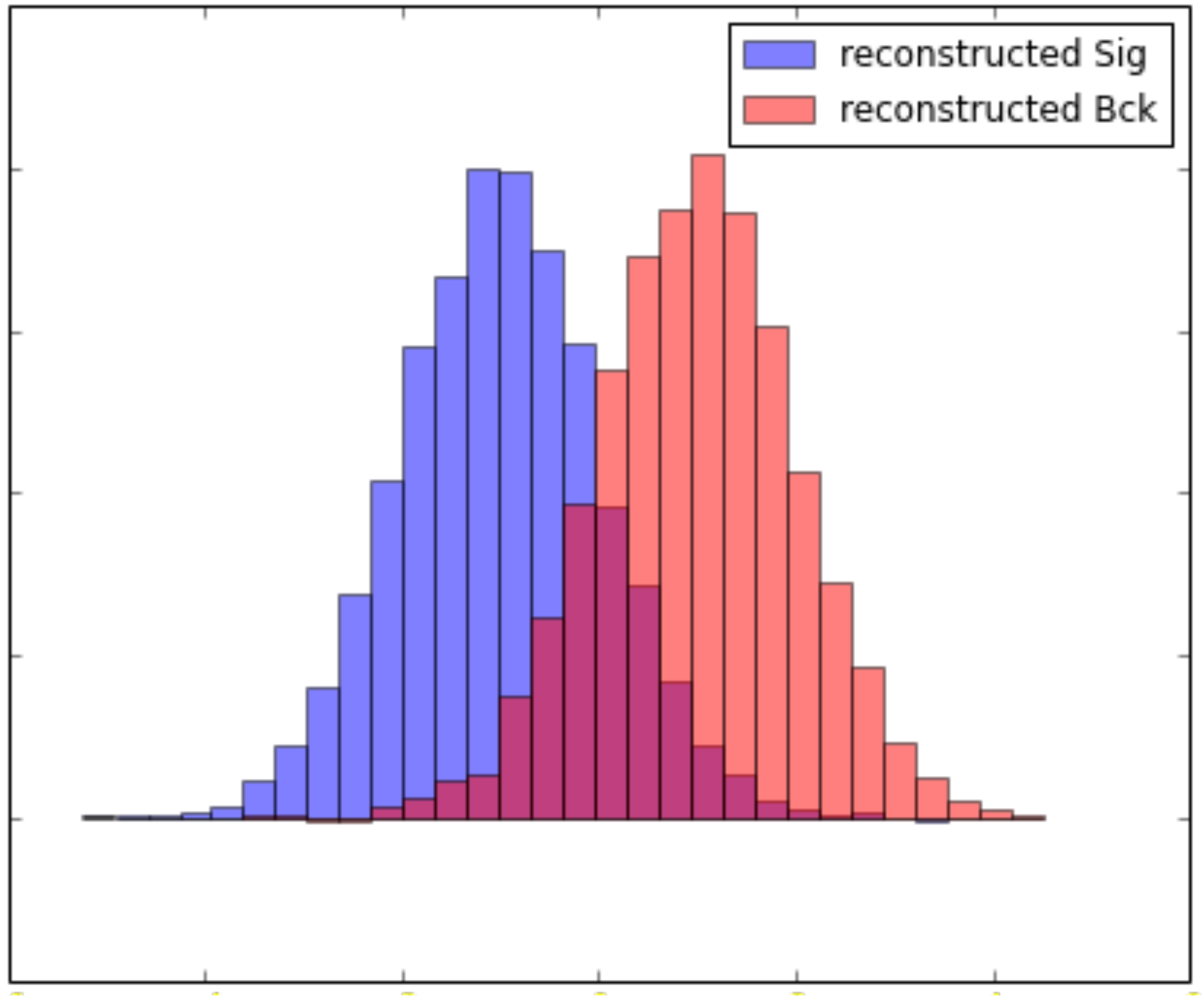


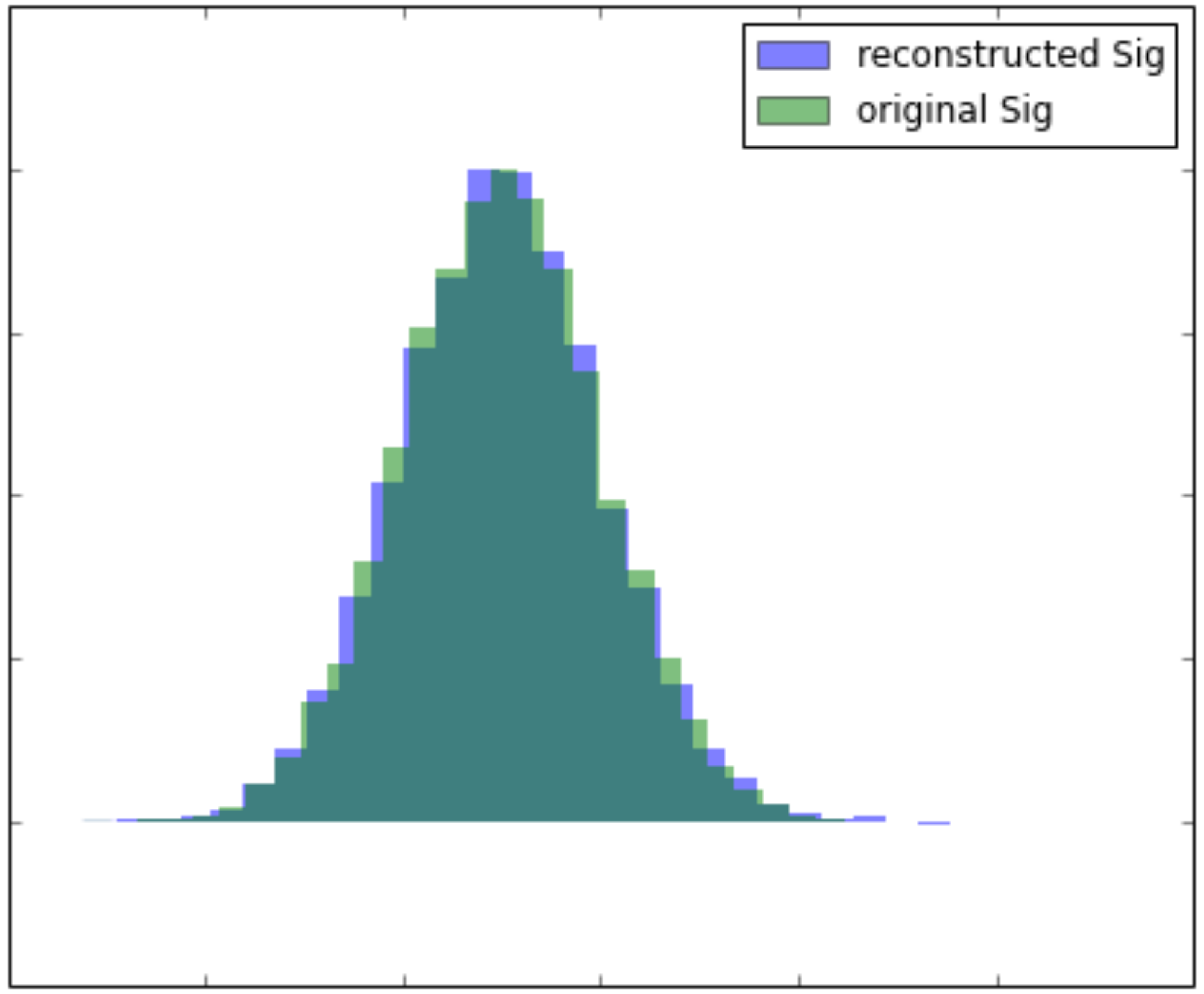


$$Bin1 : w_{b_1} f_b + w_{s_1} f_s \quad *w_{b_2}$$
$$Bin2 : w_{b_2} f_b + w_{s_2} f_s \quad *(-w_{b_1})$$

+ will obtain initial signal distribution

Reconstruction

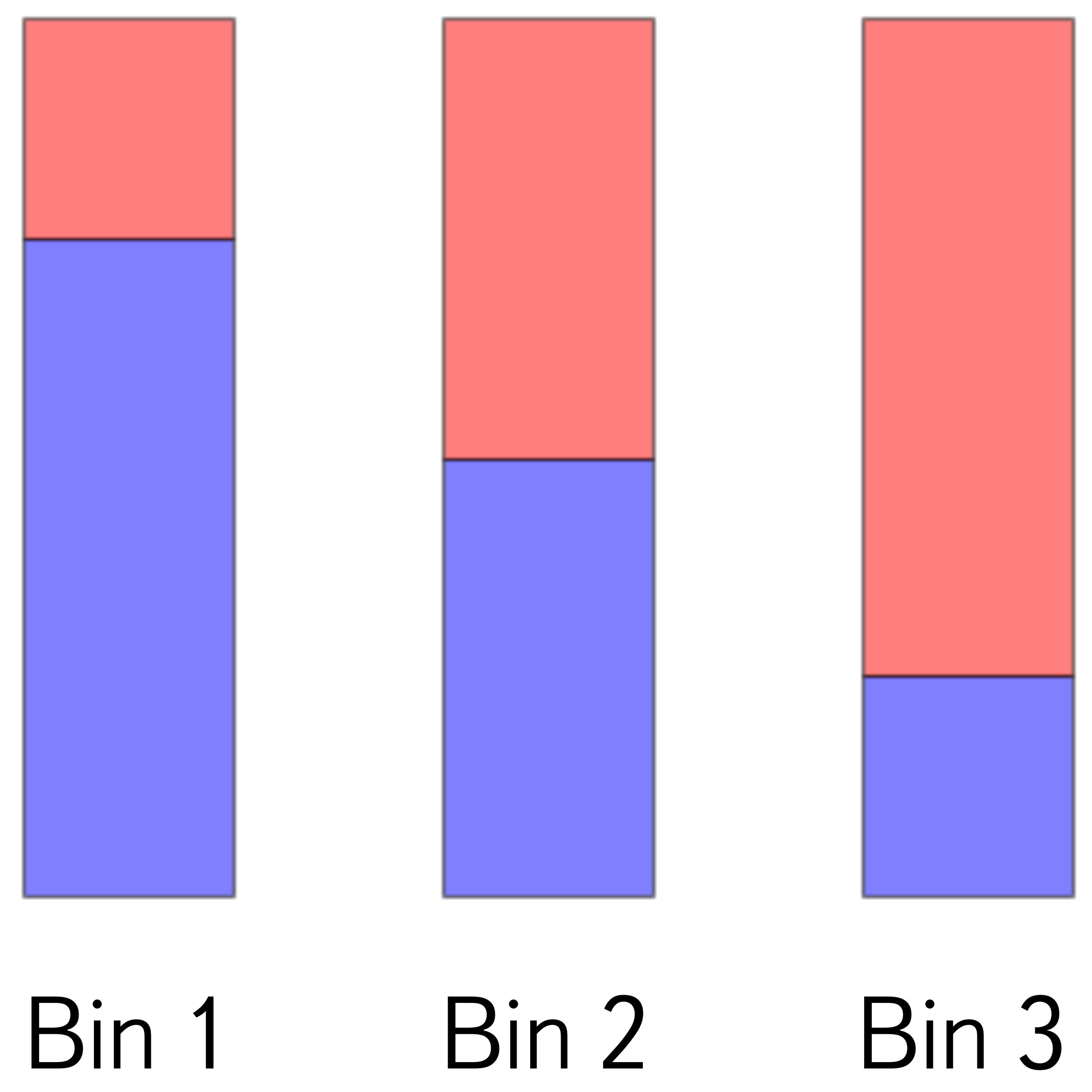




More bins: sWeight

- Equivalent to some optimization problem
- Has simple explicit solution
- Produce weights (sWeight) for each event
- Feature pdf with sWeight = signal pdf
- Details for sPlot technique
- Blogpost about sPlot (simplified explanation)

Proportion of events inside bins



How to reweight?

ML on sPlot data

Properties and problems

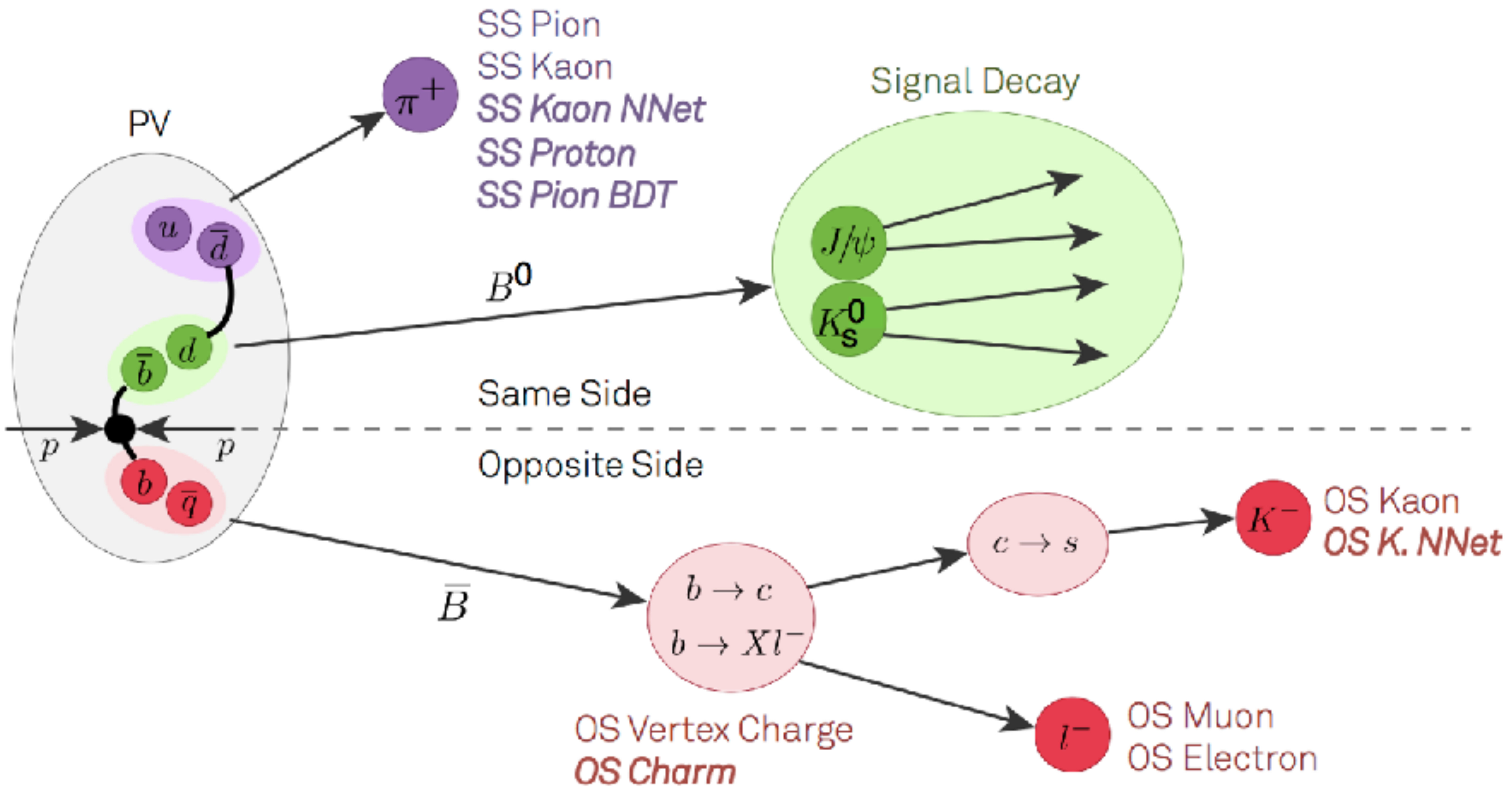
- There are always negative values among sWeights
- The correspondence between probability to be signal/bck and sWeight
- Standard ML algorithms cannot be applied* (loss function is not convex)

Approaches

- Ignore negative weights
- Take sWeight>0 as signal, sWeight <=0 as bck (with -sWeight)
- Put sample with weight=P(to be signal) as signal and with weight=P(to be bck) as bck
- Introduce another loss function

B-Tagging

B-Tagging

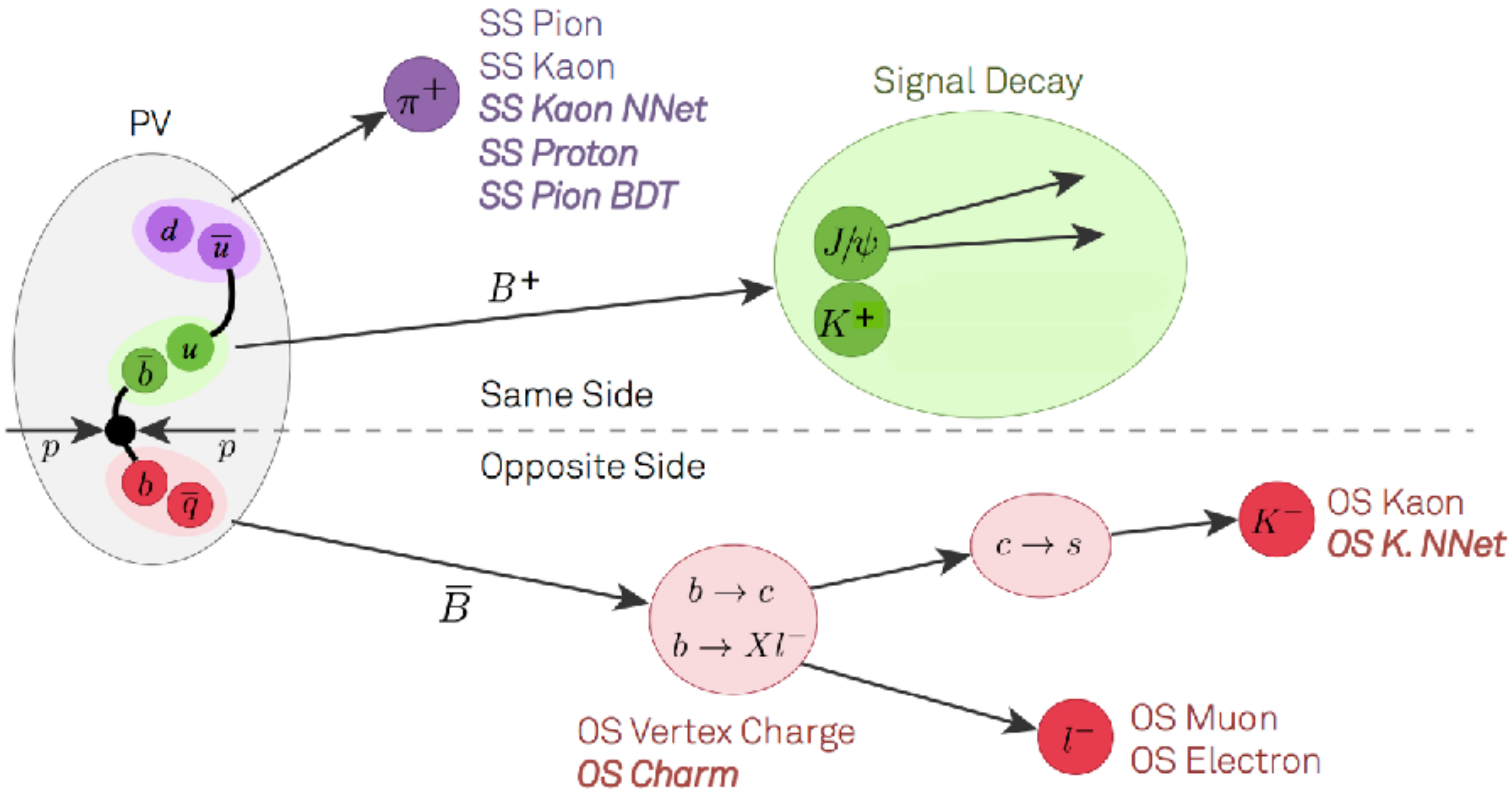


SS Pion
SS Kaon
SS Kaon NNet
SS Proton
SS Pion BDT
PV
π^+
u $\bar{d}$
$\bar{b}$ d
B^0
Signal Decay
J/ψ
K_S^0
Same Side
p p
Opposite Side
b $\bar{q}$
$\bar{B}$
$b \to c$
$b \to Xl^-$
$c \to s$
K^-
OS Kaon
OS K. NNet
OS Vertex Charge
OS Charm
l^-
OS Muon
OS Electron

What is FT?

- The Flavour Tagging (FT) algorithm determines the flavour of each reconstructed signal B meson at production
- The B meson consists either of a $\boldsymbol{b}$ or a $\bar{\boldsymbol{b}}$ quark, which defines its flavour
- The FT algorithm should predict probabilities $P(b)$ and $P(\bar{b})$

B-Tagging (ground truth data)



$$P(\bar{b}) = P(B^+),\ P(b) = P(B^-);$$

Current tagging system

- Take $B^{\pm} \rightarrow J/\psi\ K^{\pm}$
- Apply sPlot technique to real data to extract signal-like data (sWeights)
- Choose:
  - one tagging track (PID selection and, if necessary, max PT track) for each event
  - one secondary vertex, which produces a tagging particle
- Train exclusive taggers for SS (same side) tagging particles and OS (opposite side)
- Target for classifier is ‘right tagged’ label for chosen track/vertex
- Combine all taggers to one (probabilistic model) to obtain P(anti-b quark)

Inclusive probabilistic model

- Let s_p – charge of track or vertex (+1 or -1), s_B – quark flavour (+1 for b and -1 for $\bar{b}$), components={tracks, vertices}, then assume that:

$$\frac{P(\bar{b})}{P(b)} = \prod_{\text{components}} \frac{P(\bar{b}|B, \text{component}, \mathrm{s_p})}{P(b|B, \text{component}, \mathrm{s_p})} =$$

$$= \prod_{\text{components}} \left(\frac{P(s_B \cdot s_p > 0|B, \text{component})}{P(s_B \cdot s_p < 0|B, \text{component})} \right)^{s_p}$$

$$flavour = \begin{cases} \bar{b}, & \text{if } P(\bar{b}) \geq P(b) \\ b, & \text{if } P(\bar{b}) < P(b) \end{cases} \qquad \omega = \min\left[P(\bar{b}), P(b)\right]$$

Why Inclusive?

- does not depend on the tagging particle type (pion, kaon, electron, muon, proton);
- is not split into the opposite and same side;
- combines full available information in the event.

Inclusive training

- LHCb data samples with reconstructed signal decay $B^+ \to J/\psi K^+$ *or* $B^- \to J/\psi K^-$
- set of all tracks/vertices for all events form the tracks/vertices datasets (we except for ones forming the reconstructed signal decay)
- train gradient boosted decision trees (GBDT)
- target for a classifier is a label that B meson has the same sign as the track/vertex and output is a probability

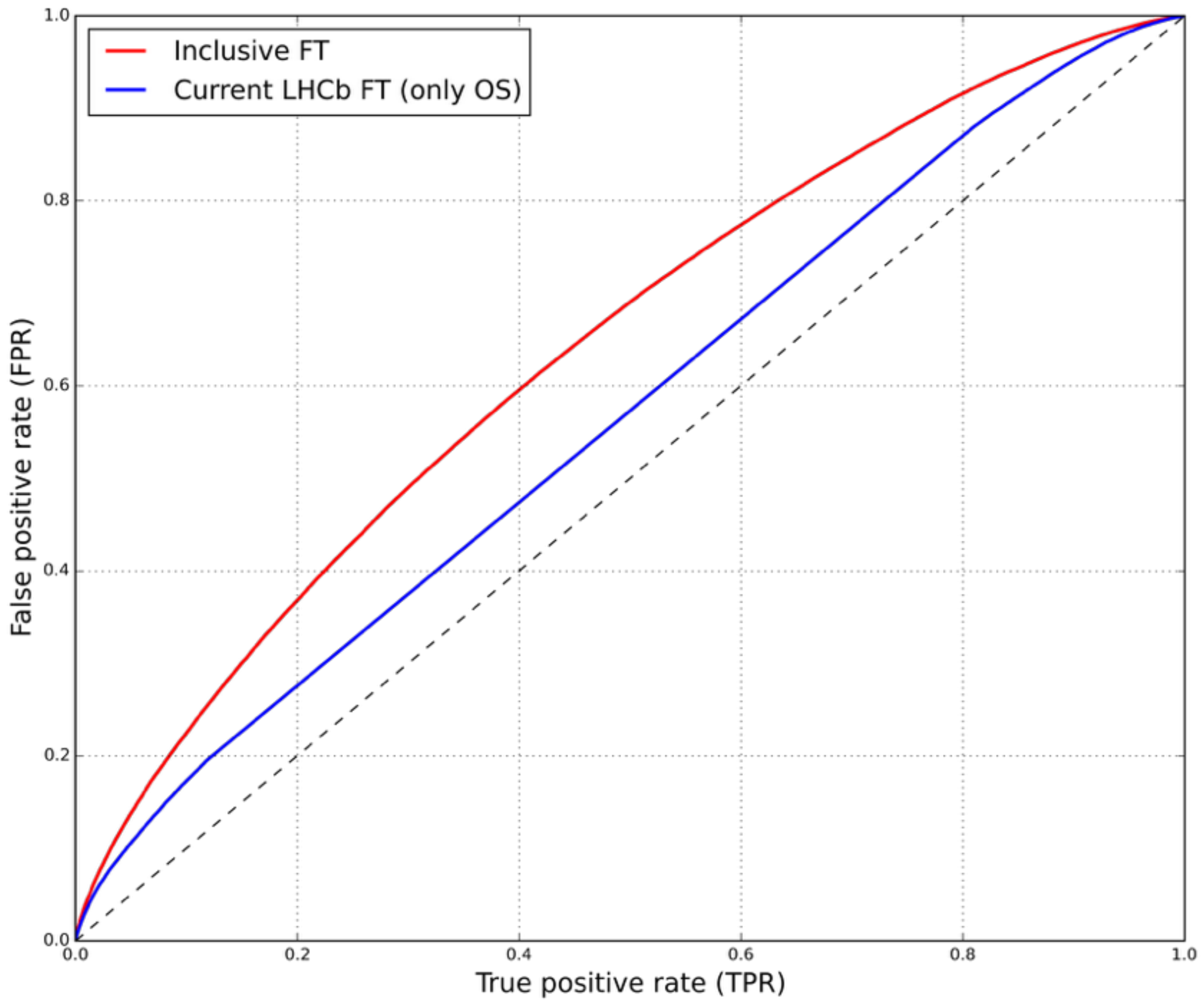
$$target = \begin{cases} 1, & \text{if } s_B \cdot s_p > 0 \\ 0, & \text{if } s_B \cdot s_p < 0 \end{cases}$$

$$P(s_B \cdot s_p > 0 | B, \text{component})$$

Result

- ROC AUC score 0.641
- ROC AUC for current tagging system 0.566

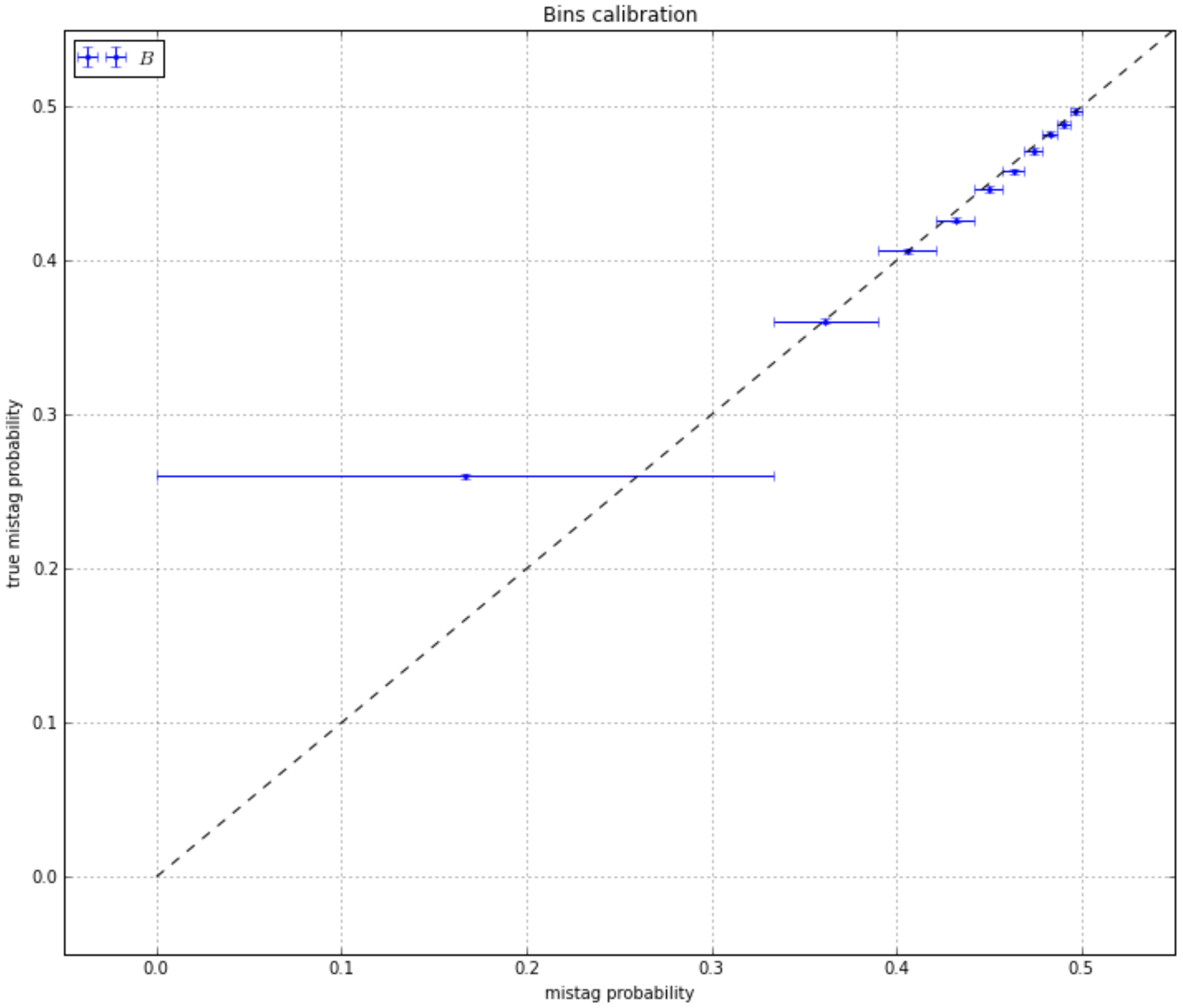
ROC curve is computed including untagged events (events which didn't pass preselections; for them probability is set to 0.5)



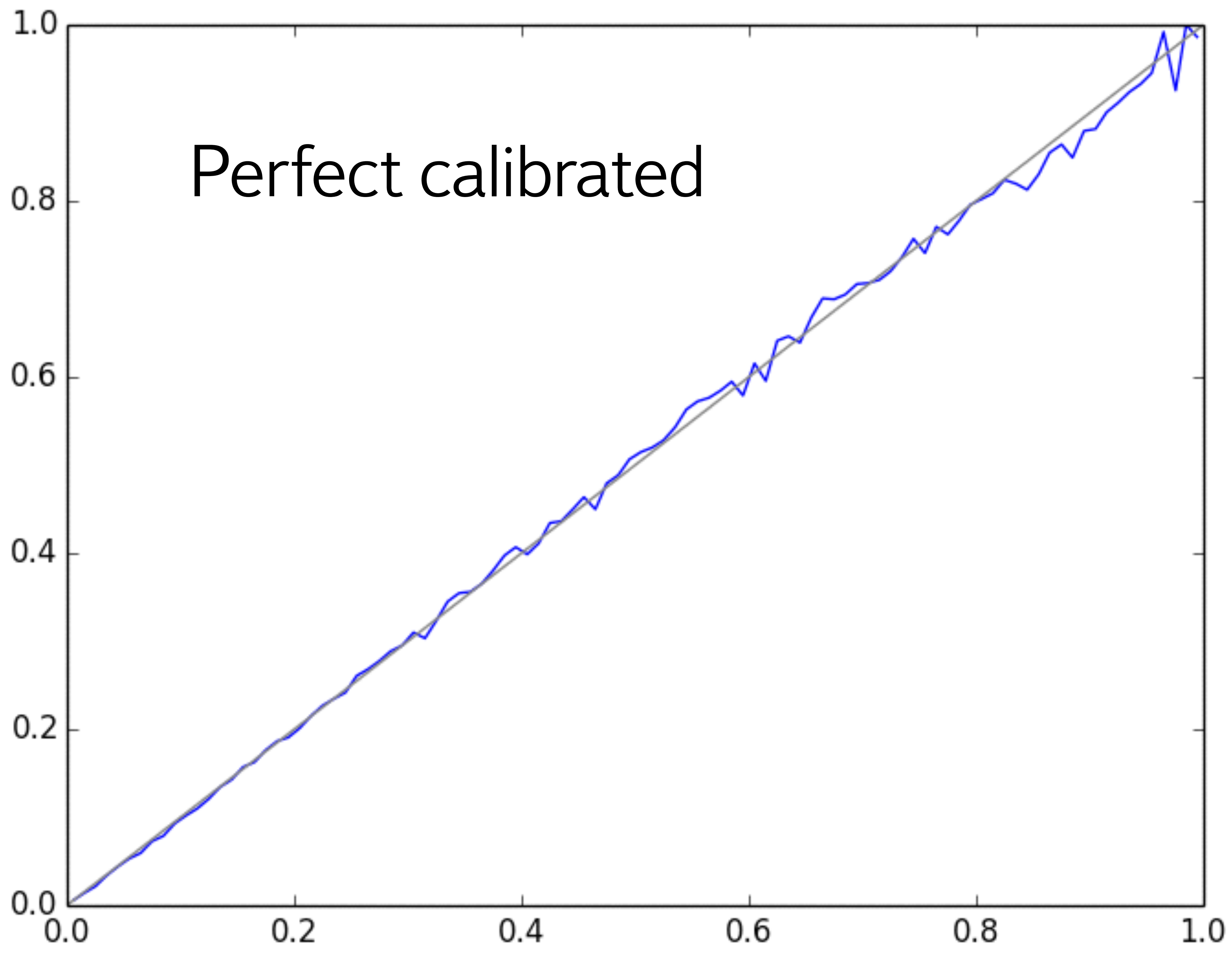
Classifiers output calibration to probabilities

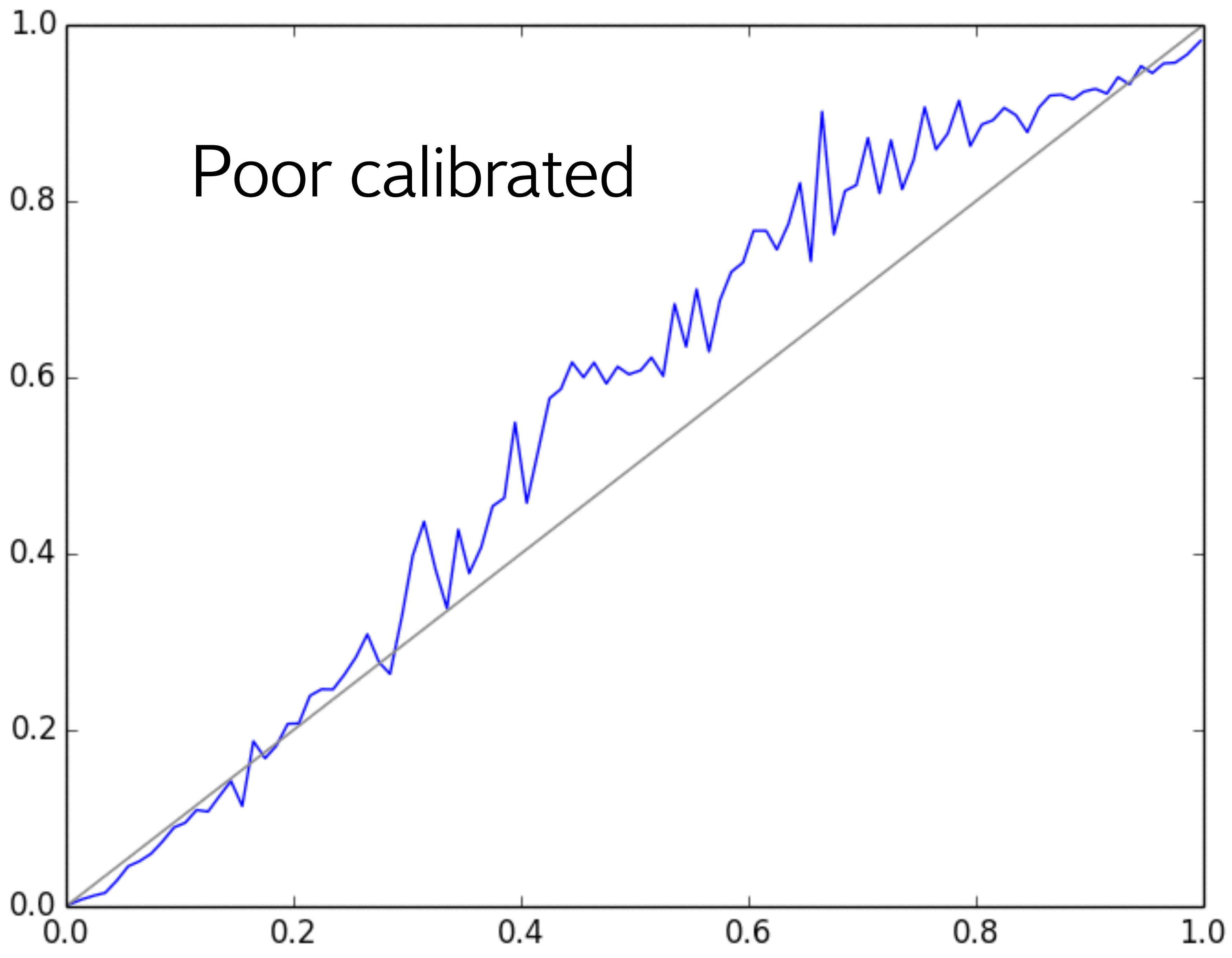
How to check calibration?

- Enough statistics
- Divide into many bins output of the classifier (probability to be 1-labeled)
- Compute in each bin #(label=1) / (#(label=1) + #(label=0))
- Compare it with the mean of predictions in a bin



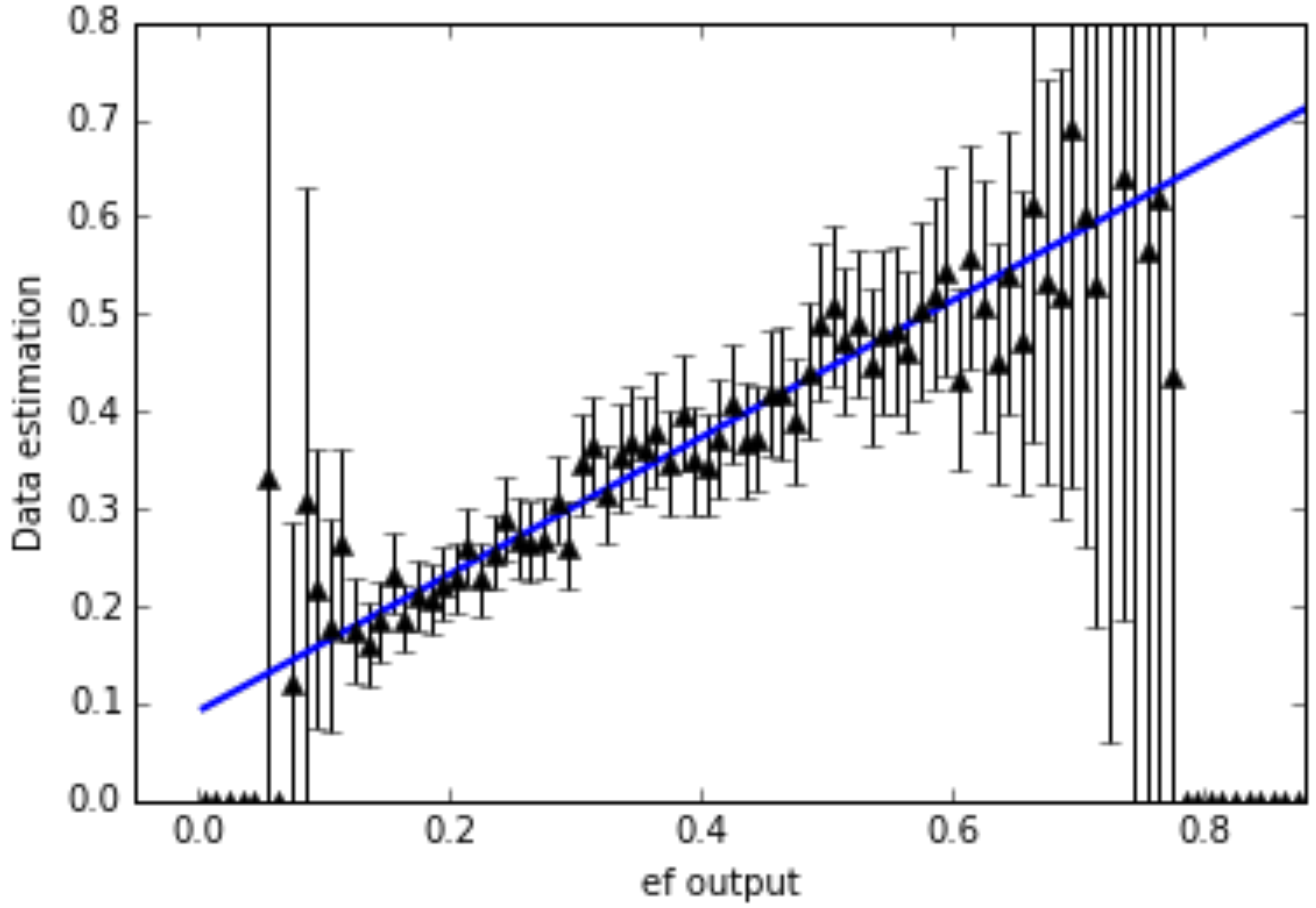
Examples


Perfect calibrated
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

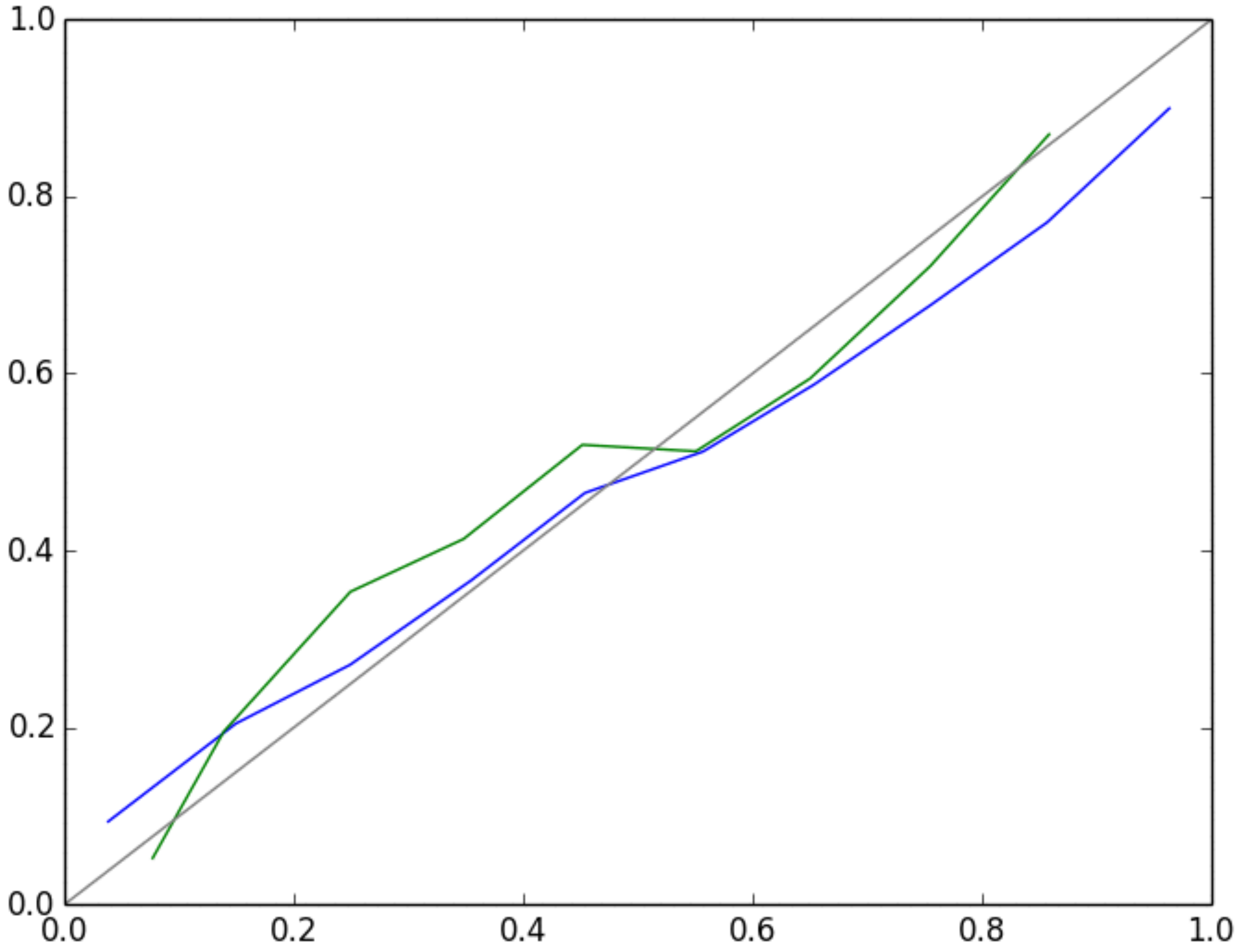

Poor calibrated
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

Bins approach

- Bins method:
  - #(1-labled data in bin) / #(data in bin)
  - fit with linear function
- Problems:
  - #bins, their thresholds
  - why linear fit?

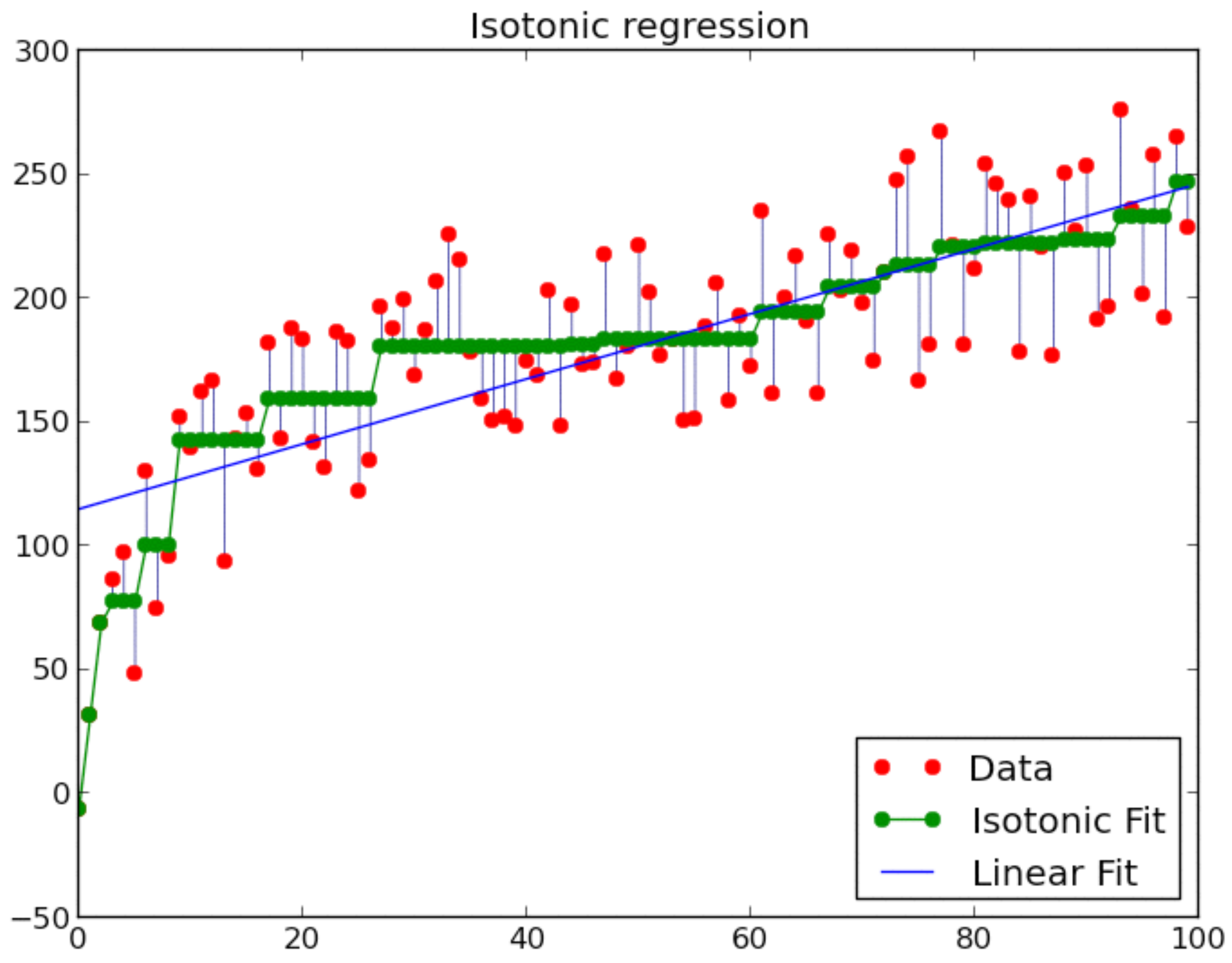


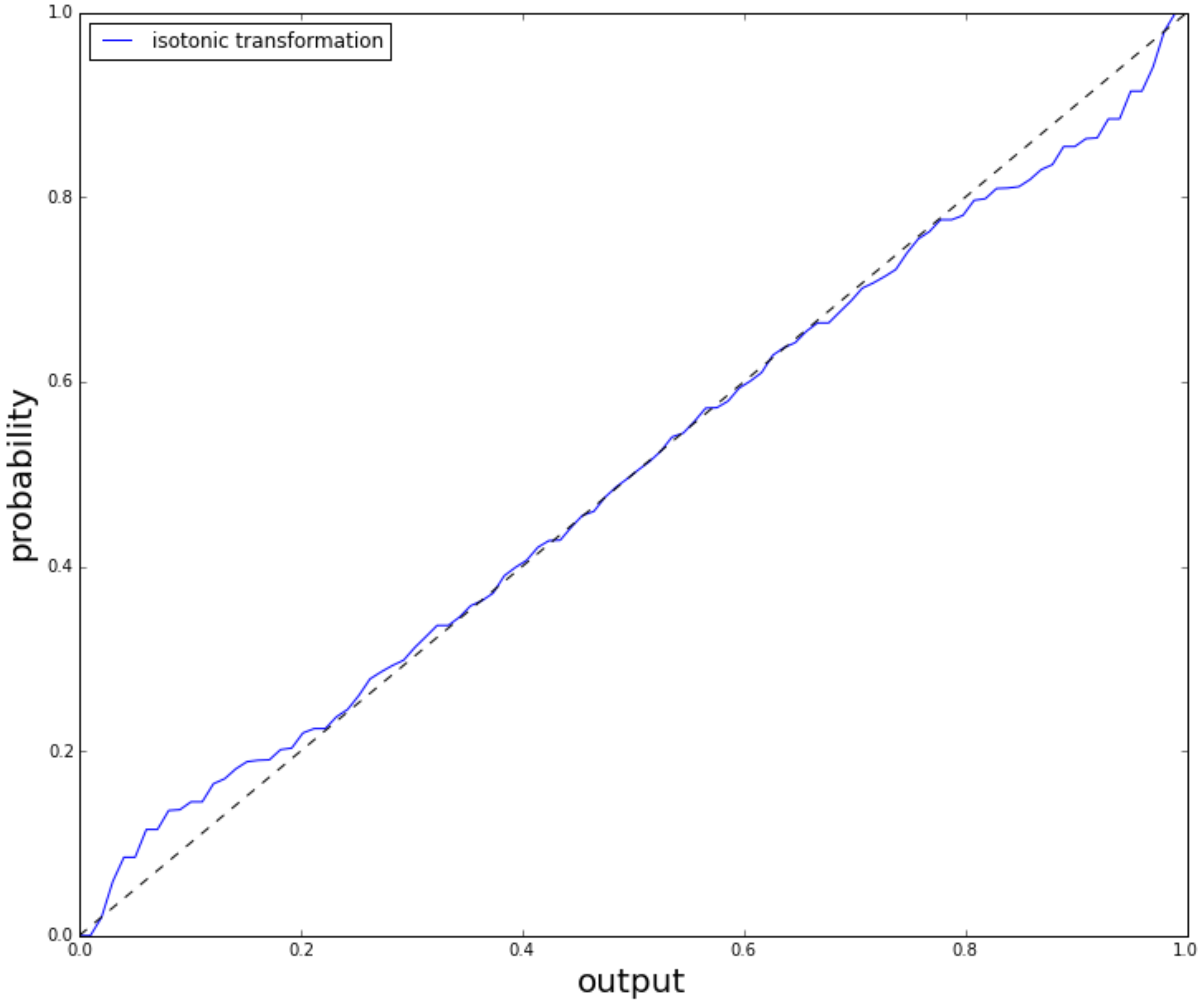
Platt’s scaling (logistic regression)




```
from sklearn.linear_model import LogisticRegression as LR

lr = LR()

lr.fit(p_train.reshape( -1, 1 ), y_train )

p_calibrated = lr.predict_proba( p_test.reshape( -1, 1 ))[:,1]
```

Isotonic regression





Sklearn interface

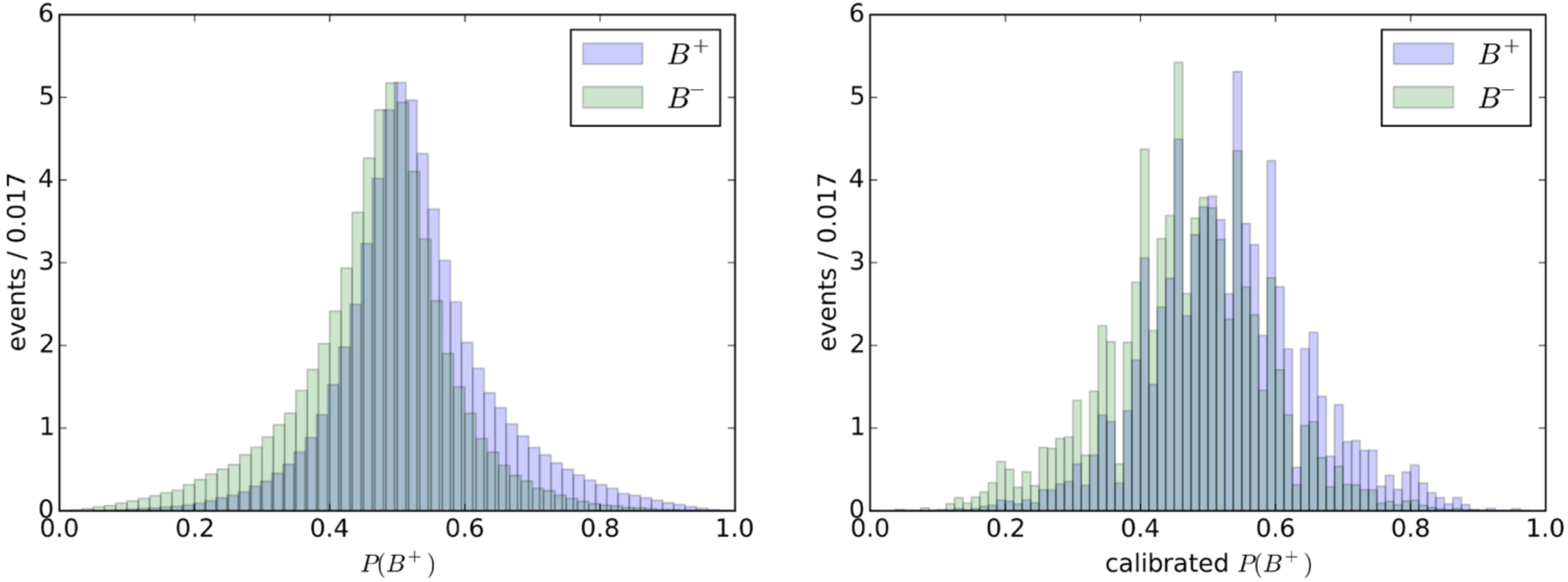
Summary

- Use isotonic calibration
- Use holdout to check your calibration rule

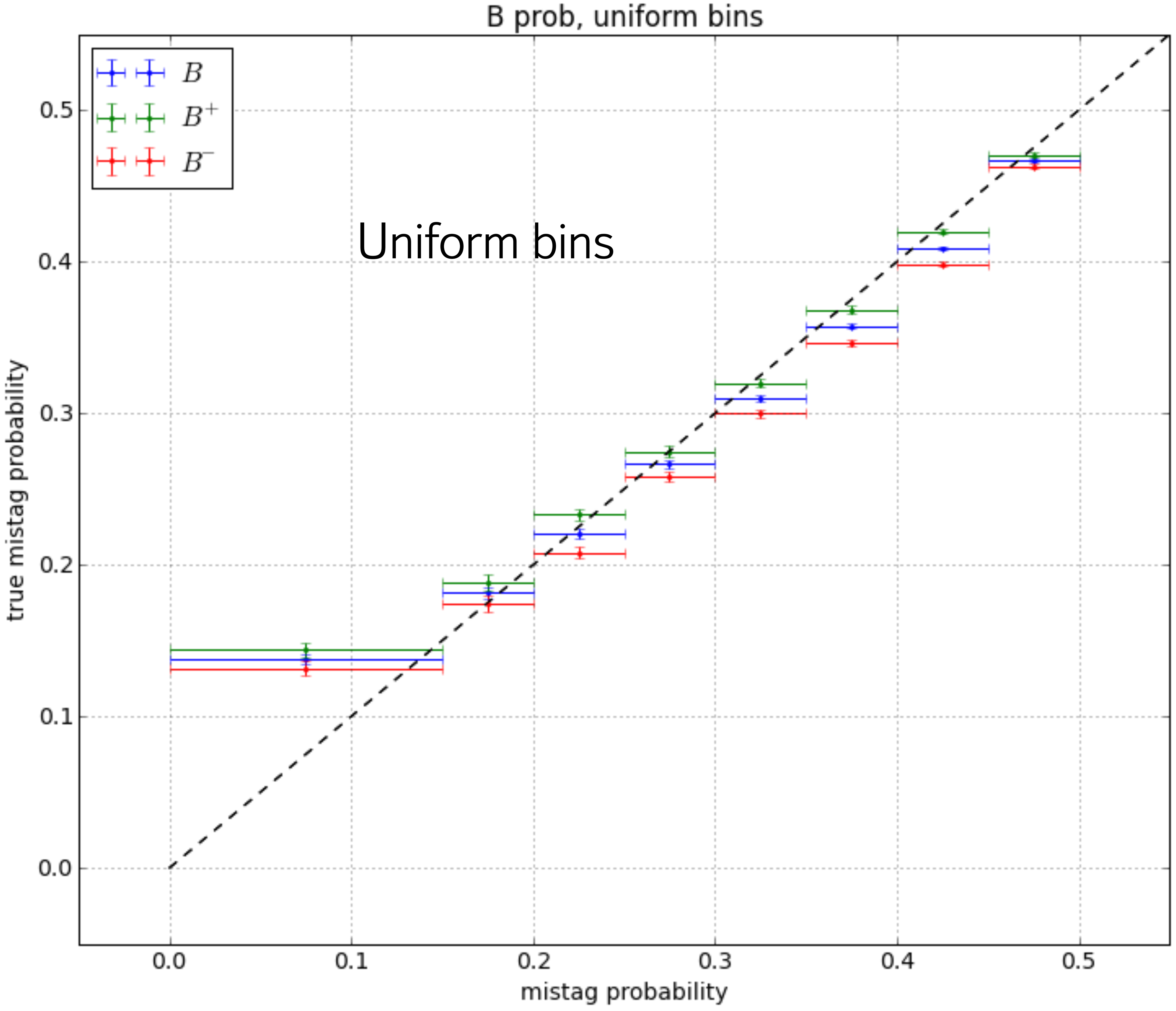
Calibration in B-tagging

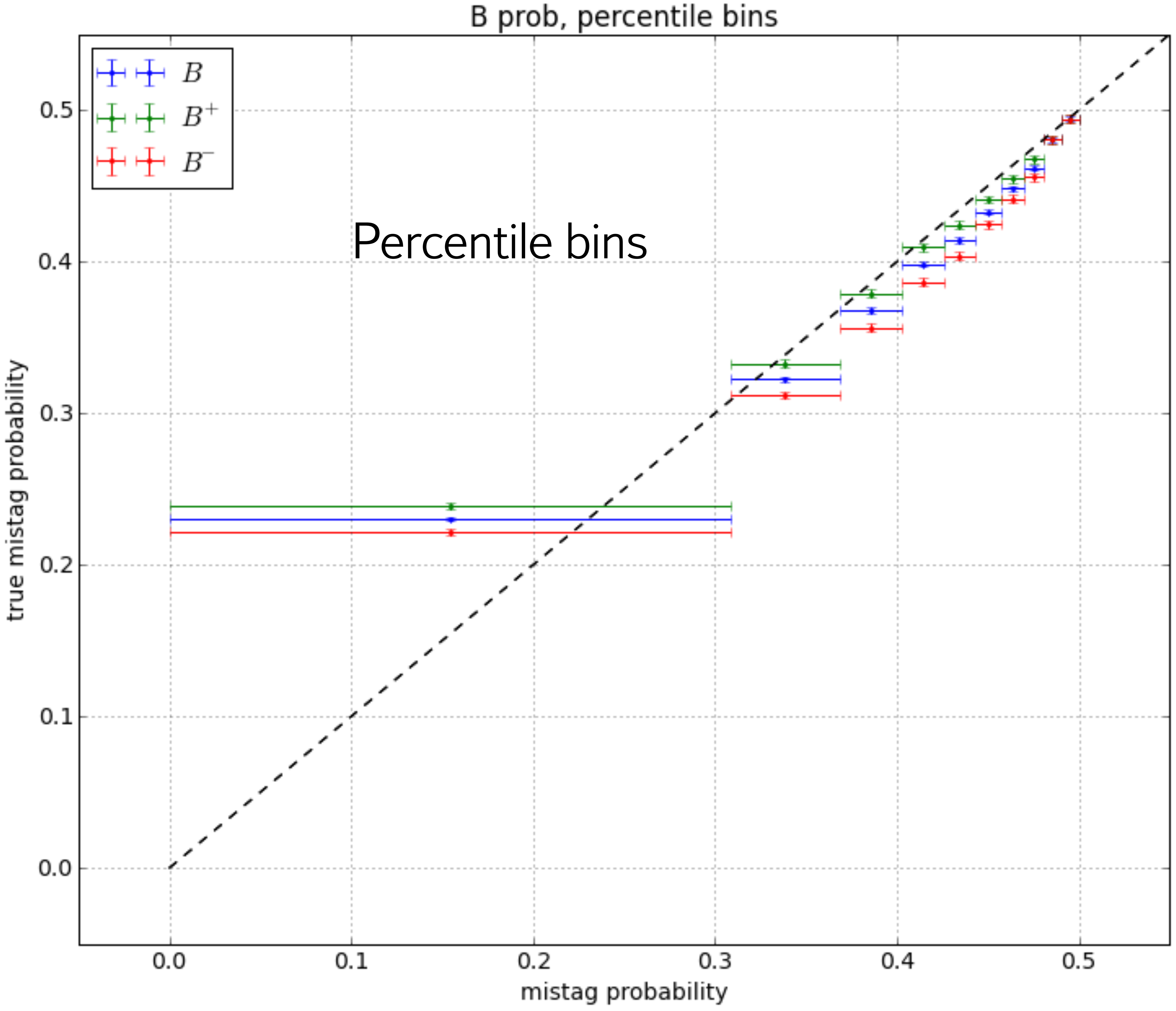
Symmetry problem

- symmetric isotonic regression (add inverse labels with inverse probability)
- add random noise after calibration for stability (to avoid clipped predictions).

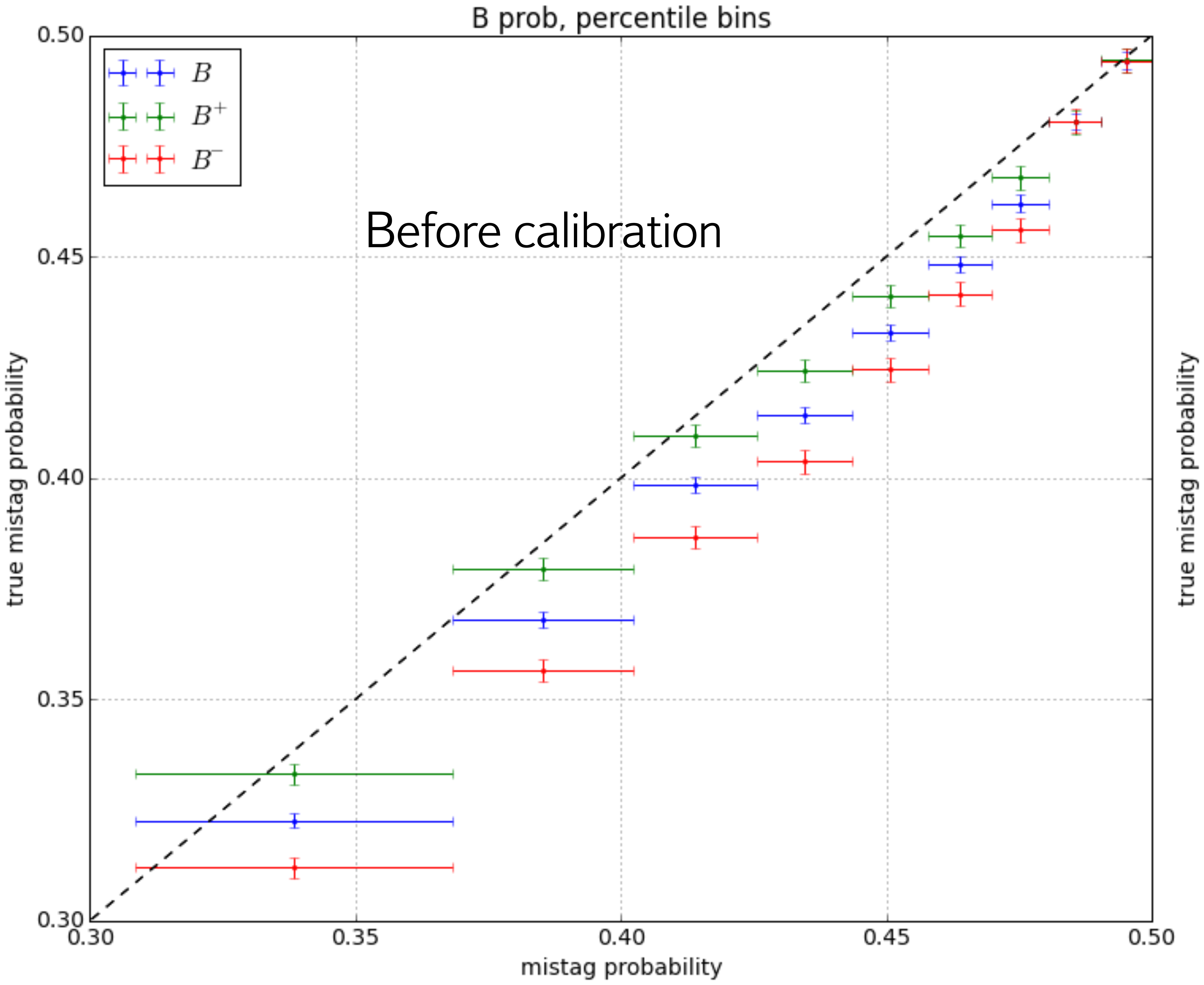


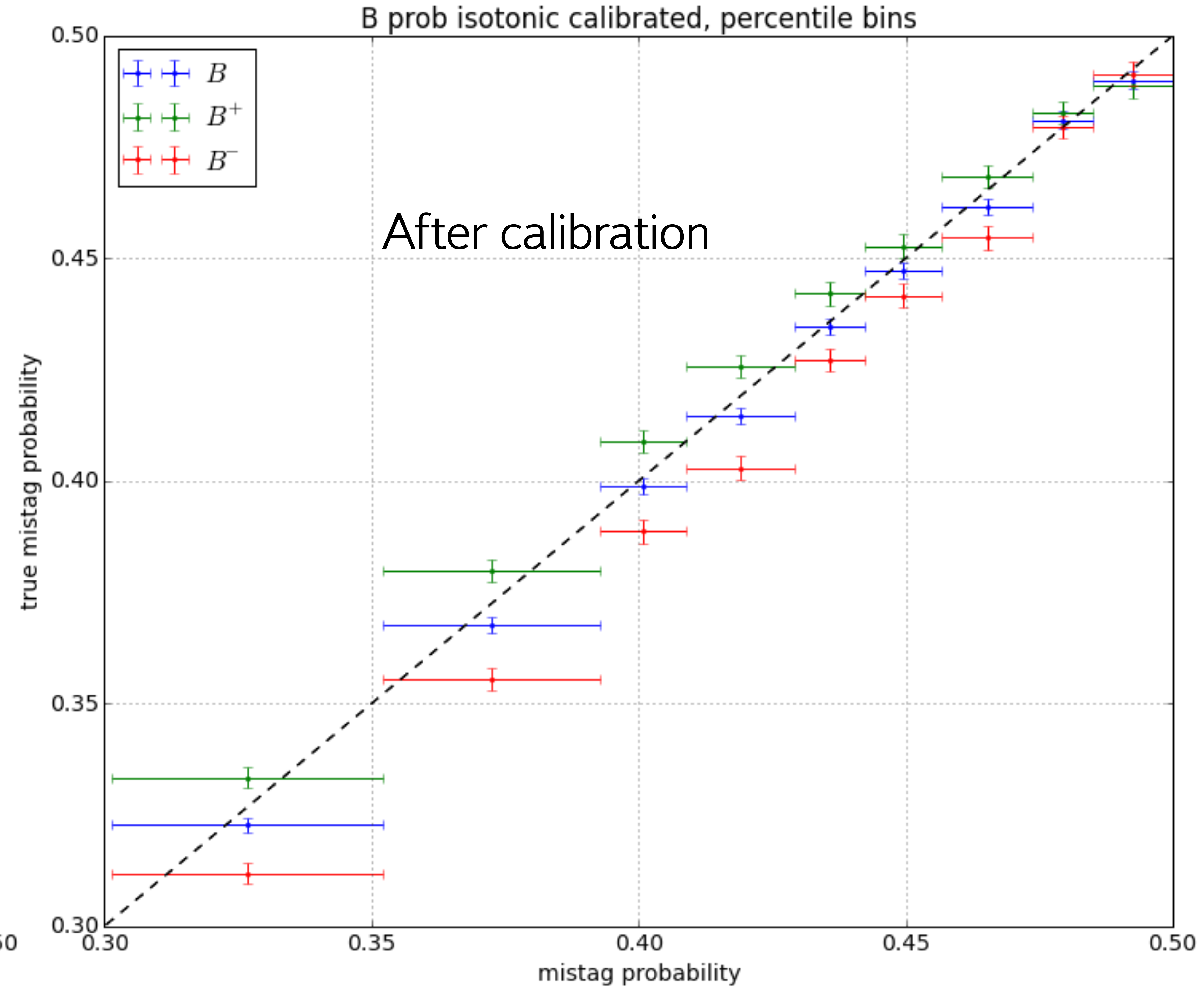
Check calibration (with B-symmetry)





Check calibration (with B-symmetry)





Conclusions

Summary & Tricks

- Measure ROC curve for events!
- Random forest to remove outliers
- sPlot technique and ML on sPlot data
- Try to combine all information for event
- Calibration aspects: isotonic regression (or its symmetric case with adding noise)
- Percentile bins (if we need to use binning)

REP

- Python-based (numpy, pandas, …), Jupyter-friendly
- Unified scikit-learn-like API to many ML packages
  (Sklearn, XGBoost, uBoost, TMVA, Theanets, …)
- Meta-algorithms pipelines («REP lego»)
- Configurable interactive reporting & visualization
  to ensure model quality (e.g. check for overfitting)
- Pluggable quality metrics
- Parallelized training of classifiers & grid search (IPython parallel)
- Available at Github

References

- LHCb Topological Trigger Reoptimization for Run 2 (repository)
- LHCb Topological Trigger Reoptimization for Run 2 (paper)
- The HLT inclusive B triggers
- sPlot
- Blogpost about sPlot (simplified explanation)
- Development of "same side" flavour tagging algorithms for measurements of flavour oscillations and CP violation
- Blogpost about classifier's output calibration to probability
- REP platform

Thanks for attention

Contacts

Likhomanenko Tatiana
researcher-developer

antares@yandex-team.ru, tatiana.likhomanenko@cern.ch