



# Ensembles in Machine Learning

Bagging. RandomForest. Stacked generalization. AdaBoost. Gradient Boosting and XGBoost

Fourth Machine Learning in High Energy Physics Summer School,  
MLHEP 2018, August 6–12

Alexey Artemov<sup>1,2</sup>

<sup>1</sup> Skoltech    <sup>2</sup> National Research University Higher School of Economics

# Lecture overview

- › Bootstrap, Bagging, and Random Forests
- › Learning Theory continued
- › Stacked generalization
- › Naive boosting for regression
- › Adaptive boosting
- › Gradient boosting machine
- › Stochastic gradient boosting
- › XGBoost

# Bagging and Random Forests

# The bootstrapping procedure

- › Input: a sample  $X^\ell = \{(x_i, y_i)\}_{i=1}^\ell$
- › **Bootstrapping:** generate new samples  $X_1^m$  of  $(x_i, y_i)$  drawn from  $X^\ell$  uniformly at random with replacement (replicated  $(x_i, y_i)$  possible!)
- › **Ensemble learning idea:**

1. Generate  $N$  bootstrapped samples

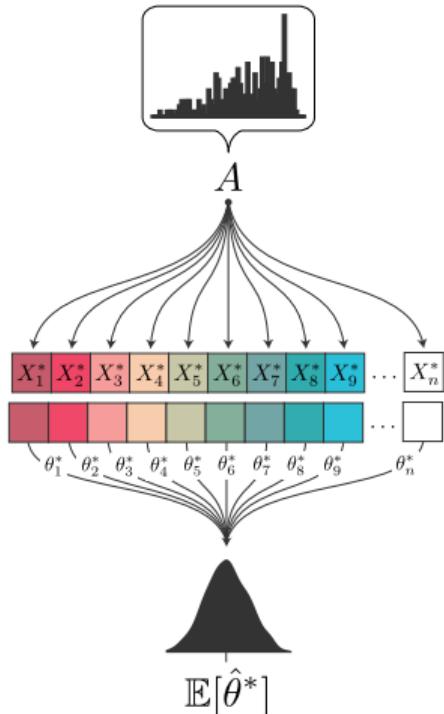
$$X_1^m, \dots, X_N^m$$

2. Learn  $N$  hypotheses  $h_1, \dots, h_N$

3. Average predictions to obtain

$$h(x) = \frac{1}{N} \sum_{i=1}^N h_i(x)$$

4. Profit!



Picture credit: <http://www.drbunsen.org/bootstrap-in-picture>

# Bagging: bootstrap aggregation (+ demo)

- › Input: a sample

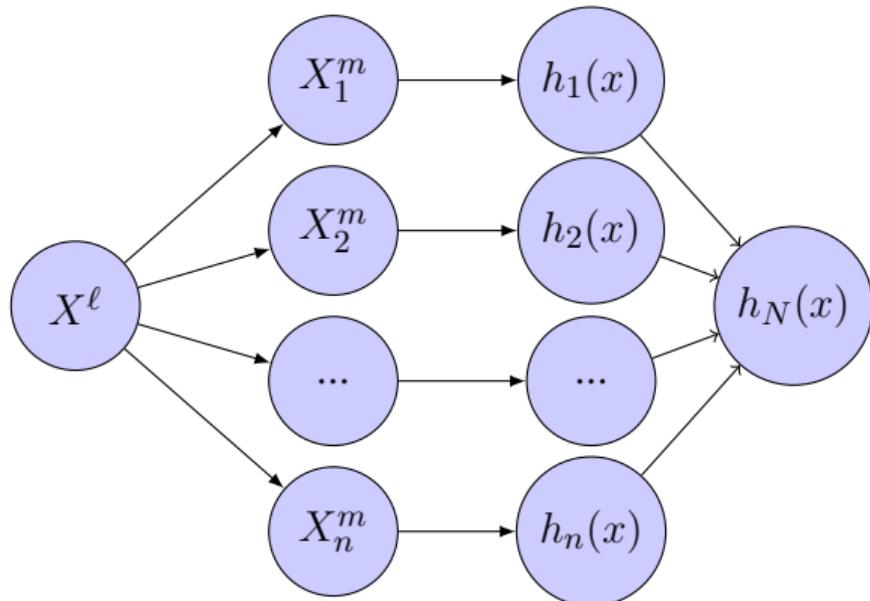
$$X^\ell = \{(x_i, y_i)\}_{i=1}^\ell$$

- › Weak learners via bootstrapping

$$\tilde{\mu}(X^\ell) = \mu(\tilde{X}^\ell)$$

- › Ensemble average

$$\begin{aligned} h_N(x) &= \frac{1}{N} \sum_{i=1}^N h_i(x) = \\ &= \frac{1}{N} \sum_{i=1}^N \tilde{\mu}(X^\ell)(x) \end{aligned}$$



# The Random Forest algorithm

- › Bagging over (weak) decision trees
- › Reduce error via averaging over instances and features
- › Input: a sample  $X^\ell = \{(\mathbf{x}_i, y_i)\}_{i=1}^\ell$ , where  $\mathbf{x}_i \in \mathbb{R}^d, y_i \in \mathbb{Y}$
- › The algorithm iterates for  $i = 1, \dots, N$ :
  1. Pick  $p$  random features out of  $d$
  2. Bootstrap a sample  $X_i^m = \{(\mathbf{x}_i, y_i)\}_{i=1}^\ell$  where  $\mathbf{x}_i \in \mathbb{R}^p, y_i \in \mathbb{Y}$
  3. Learn a decision tree  $h_i(\mathbf{x})$  using bootstrapped  $X_i^m$
  4. Stop when leafs in  $h_i$  contain less than  $n_{\min}$  instances

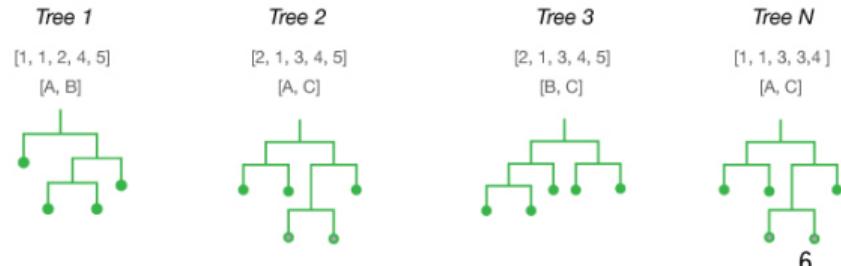
$$\mathbf{x}_i \in \{\text{A, B, C}\}$$

$$X^\ell = \{(\mathbf{x}_i, y_i)\}_{i=1}^5$$

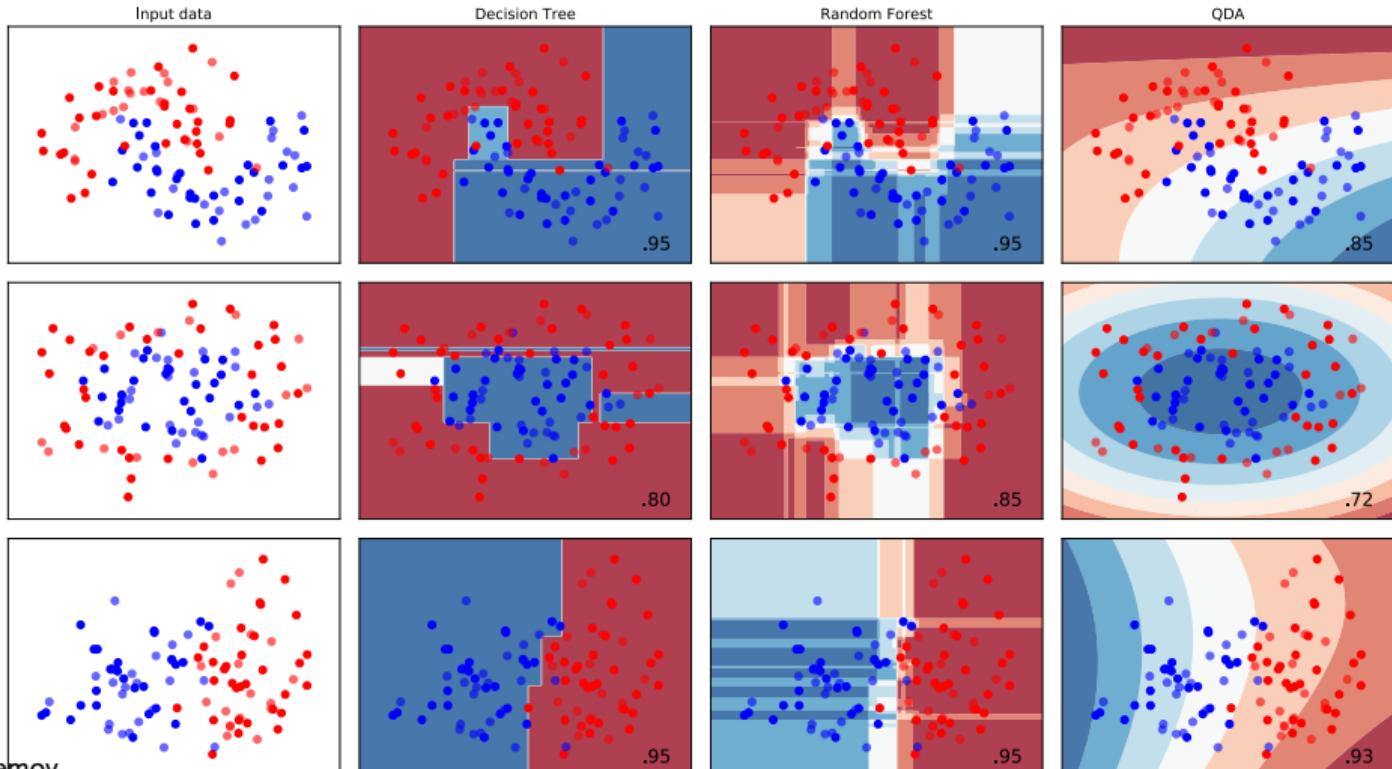
Bootstrap  $X_i^m, i \in \{1, 2, 3, 4\}$

Learn Tree $_i(\mathbf{x})$  using  $X_i^m$

Alexey Artemov



# Random Forest: synthetic examples



# Learning Theory (continued)

# Expected risk formalism

- › Input: the training set  $X^\ell = \{(x_i, y_i)\}_{i=1}^\ell$
- › Suppose  $(x_i, y_i) \in \mathbb{X} \times \mathbb{Y}$  are generated from a distribution  $p(x, y)$
- › Consider the MSE loss  $Q(y, h) = (y - h(x))^2$
- › Expected risk: average (over  $p(x, y)$ ) squared loss when using  $h$

$$R(h) = \mathbb{E}_{x,y} \left[ (y - h(x))^2 \right] = \int_{\mathbb{X}} \int_{\mathbb{Y}} p(x, y) (y - h(x))^2 dx dy.$$

- › **A statement:** the expected risk is minimized by

$$h_*(x) = \mathbb{E}[y \mid x] = \int_{\mathbb{Y}} y p(y \mid x) dy = \arg \min_h R(h)$$

# Bias-variance decomposition

- › Input: the training set  $X^\ell = \{(x_i, y_i)\}_{i=1}^\ell$
- › Learning method  $\mu : (\mathbb{X} \times \mathbb{Y})^\ell \rightarrow \mathbb{H}$
- › Can evaluate quality using average (over possible samples  $X^\ell$ )

$$\begin{aligned} Q(\mu) &= \mathbb{E}_{X^\ell} \left[ \mathbb{E}_{x,y} \left[ (y - \mu(X^\ell)(x))^2 \right] \right] = \\ &= \int_{(\mathbb{X} \times \mathbb{Y})^\ell} \int_{\mathbb{X} \times \mathbb{Y}} (y - \mu(X^\ell)(x))^2 p(x, y) \prod_{i=1}^{\ell} p(x_i, y_i) dx dy dx_i dy_i \end{aligned}$$

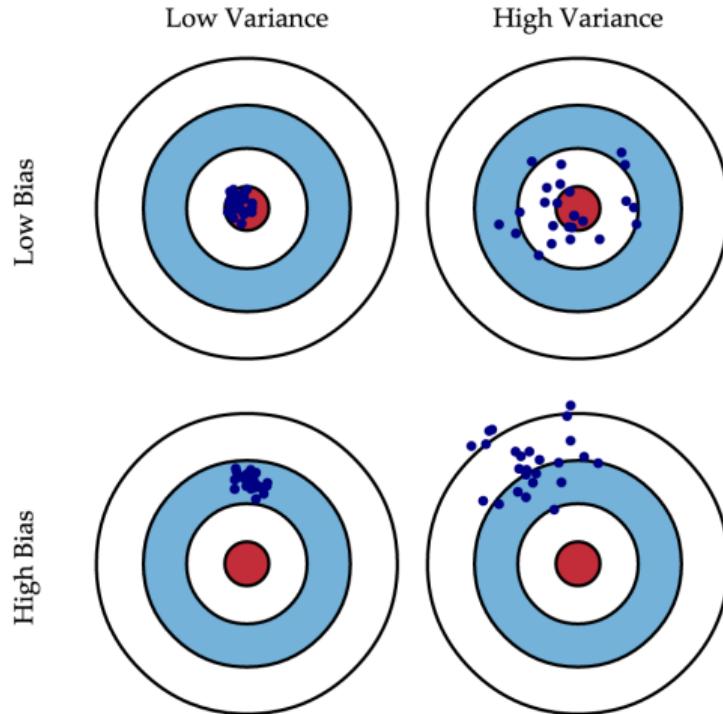
# Bias-variance decomposition

- › We arrive at the famous bias-variance(-noise) decomposition

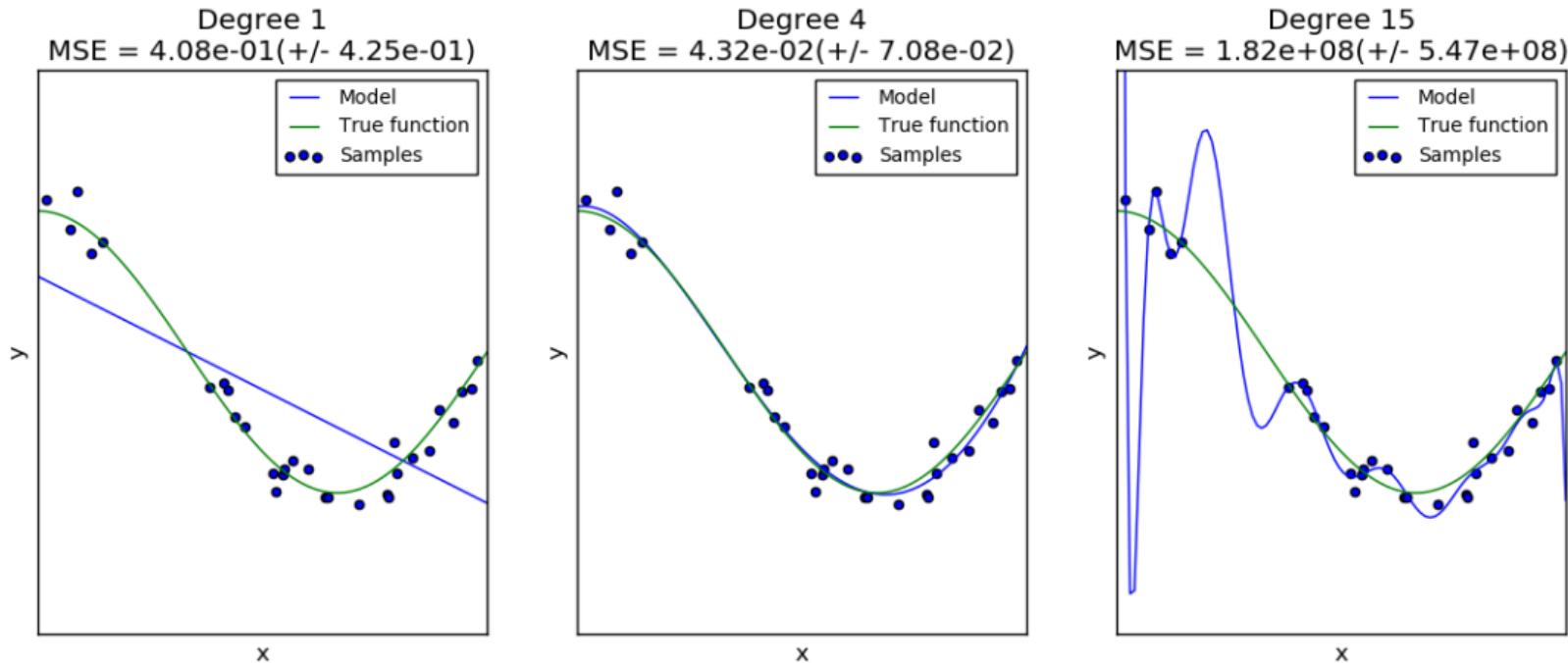
$$Q(\mu) = \underbrace{\mathbb{E}_{x,y} \left[ (y - \mathbb{E}[y | x])^2 \right]}_{\text{noise}} + \underbrace{\mathbb{E}_x \left[ (\mathbb{E}_{X^\ell} [\mu(X^\ell)] - \mathbb{E}[y | x])^2 \right]}_{\text{bias}} + \underbrace{\mathbb{E}_x \left[ \mathbb{E}_{X^\ell} \left[ (\mu(X^\ell) - \mathbb{E}_{X^\ell} [\mu(X^\ell)])^2 \right] \right]}_{\text{variance}}$$

- › **Noise term:** an error of an ideal learner (nobody can do better!)
- › **Bias term:** learner's approximation of the ideal algorithm
  - › The more complex the learning algorithm, the lower the bias
- › **Variance term:** sensitivity to sample replacement
  - › Simple algorithms have lower variance

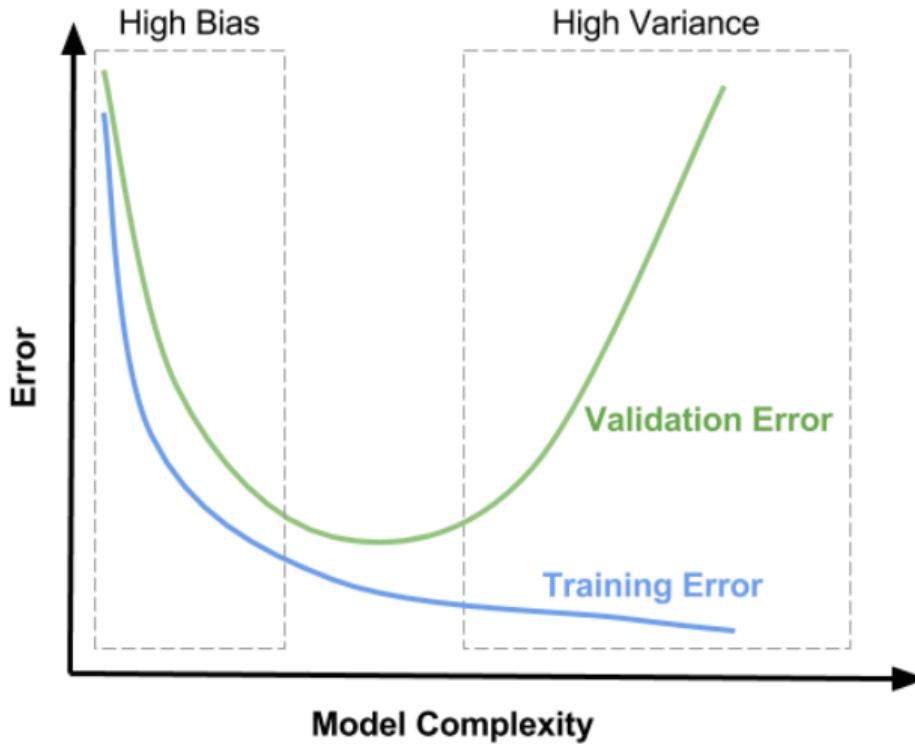
# Bias-variance decomposition



# Polynomial fits of different degrees



# Bias-variance tradeoff



# An application: statistical analysis of Bagging

- › **Bias:** not made any worse by bagging multiple hypotheses

$$\begin{aligned}\mathbb{E}_{x,y} \left[ \left( \mathbb{E}_{X^\ell} \left[ \frac{1}{N} \sum_{n=1}^N \tilde{\mu}(X^\ell)(x) \right] - \mathbb{E}[y | x] \right)^2 \right] &= \\ = \mathbb{E}_{x,y} \left[ \left( \frac{1}{N} \sum_{n=1}^N \mathbb{E}_X^\ell [\tilde{\mu}(X^\ell)(x)] - \mathbb{E}[y | x] \right)^2 \right] &= \\ = \mathbb{E}_{x,y} \left[ \left( \mathbb{E}_{X^\ell} [\tilde{\mu}(X^\ell)(x)] - \mathbb{E}[y | x] \right)^2 \right]\end{aligned}$$

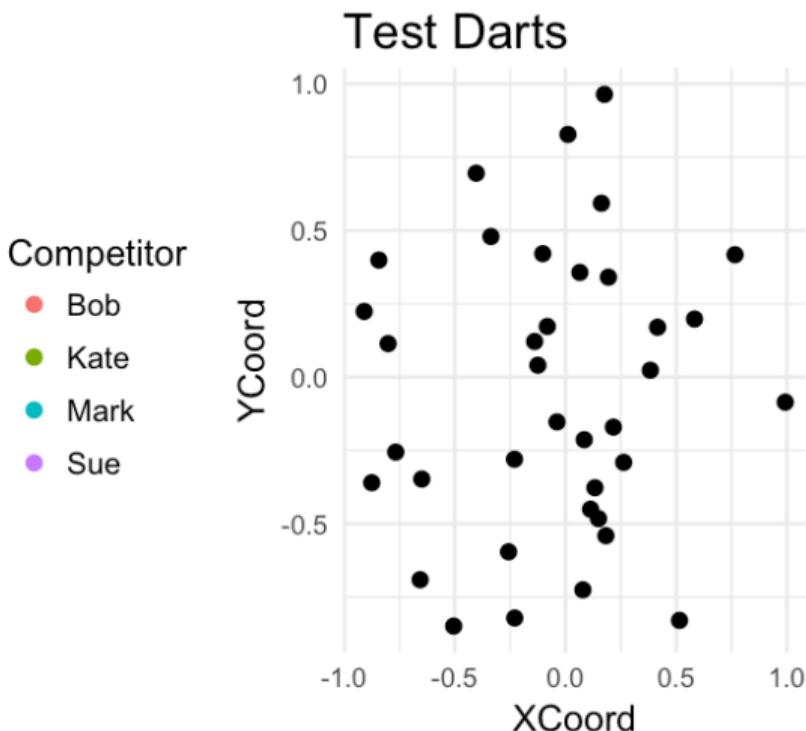
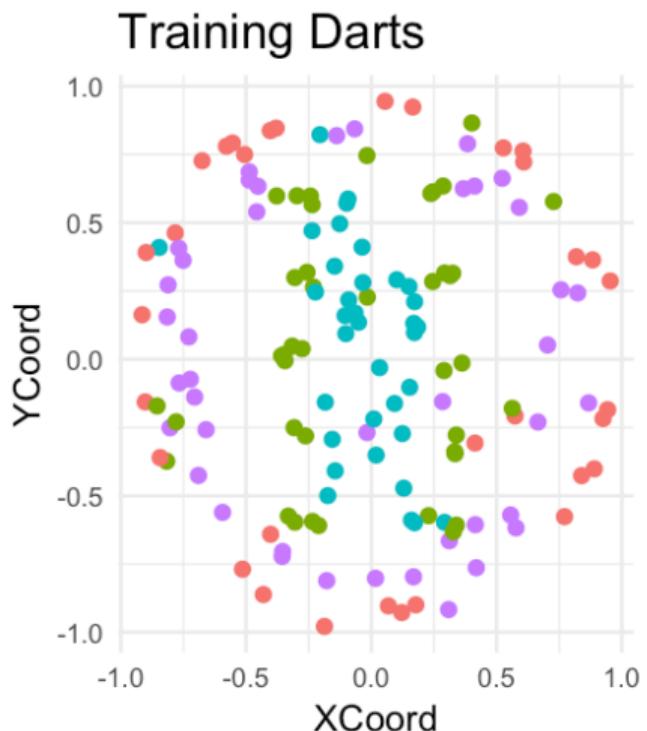
# An application: statistical analysis of Bagging

- › Variance:  $N$  times lower for uncorrelated hypotheses,  
yet not as much an improvement for highly correlated

$$\begin{aligned} \mathbb{E}_{x,y} \left[ \mathbb{E}_{X^\ell} \left[ \left( \frac{1}{N} \sum_{n=1}^N \tilde{\mu}(X^\ell)(x) - \mathbb{E}_{X^\ell} \left[ \frac{1}{N} \sum_{n=1}^N \tilde{\mu}(X^\ell)(x) \right] \right)^2 \right] \right] &= \\ &= \frac{1}{N} \mathbb{E}_{x,y} \left[ \mathbb{E}_{X^\ell} \left[ \left( \tilde{\mu}(X^\ell)(x) - \mathbb{E}_{X^\ell} [\tilde{\mu}(X^\ell)(x)] \right)^2 \right] \right] + \\ &\quad + \frac{N(N-1)}{N^2} \mathbb{E}_{x,y} \left[ \mathbb{E}_{X^\ell} \left[ \left( \tilde{\mu}(X^\ell)(x) - \mathbb{E}_{X^\ell} [\tilde{\mu}(X^\ell)(x)] \right) \times \right. \right. \\ &\quad \left. \left. \times \left( \tilde{\mu}(X^\ell)(x) - \mathbb{E}_{X^\ell} [\tilde{\mu}(X^\ell)(x)] \right) \right] \right] \end{aligned}$$

# Stacked generalization

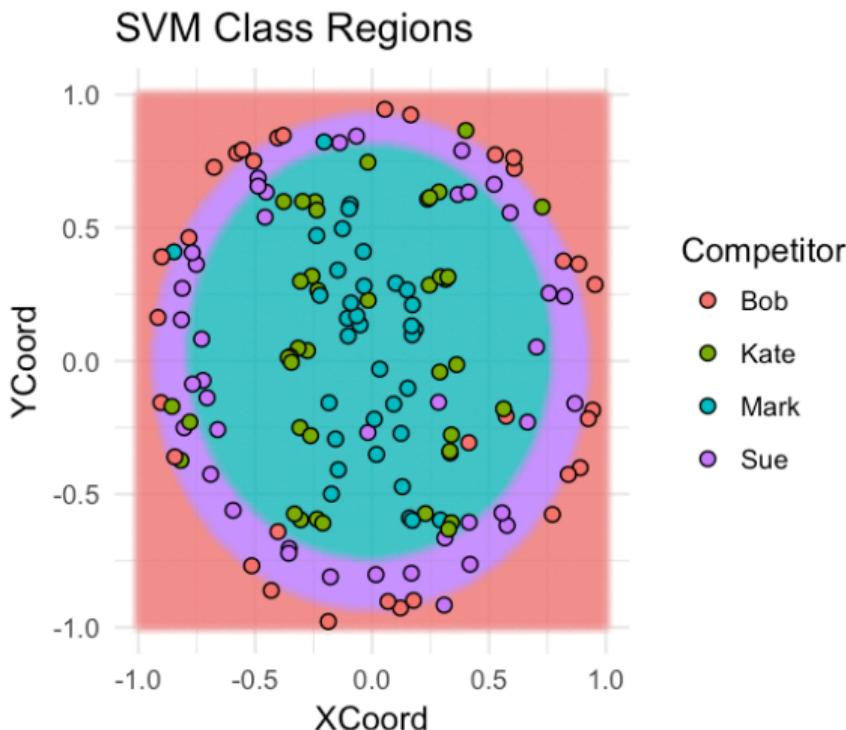
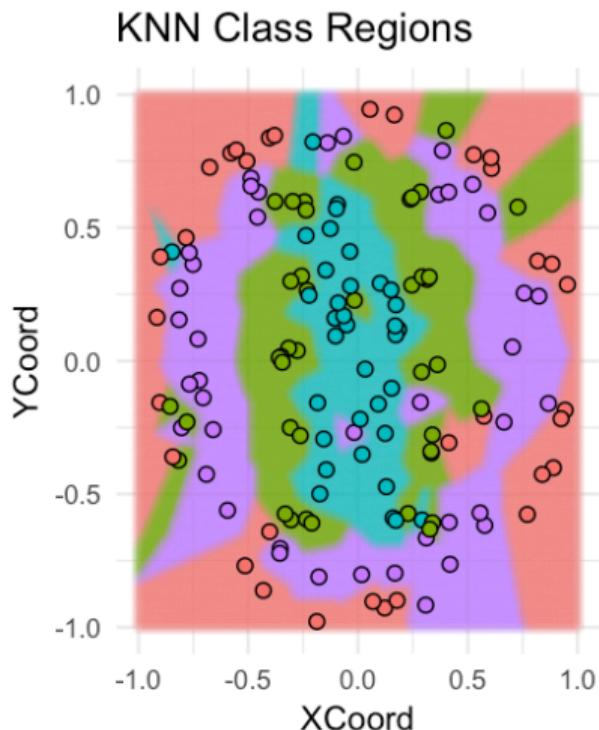
# Stacking motivation: the game of Darts



# Base model training

- › Select  $k$  nearest neighbours as base model 1
- › Fit base model 1 in the most fancy way possible
  - (grid search for optimal  $k$  using  $K$ -fold cross-validation, etc.)
- ›  $k$ -NN accuracy on Test Darts: 70%
  
- › Select Support Vector Machine as base model 2
- › Fit base model 2 in the most fancy way possible
  - (different penalizations, grid search for optimal kernel width using  $K$ -fold cross-validation, etc.)
- › SVM accuracy on Test Darts: 78%

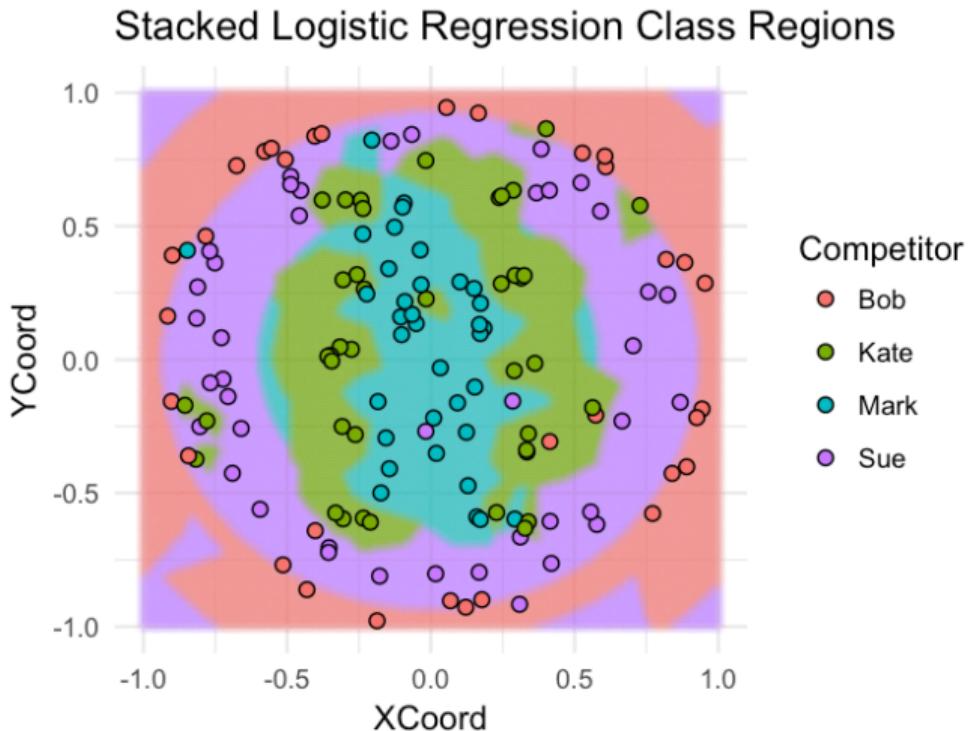
# Results for base models



# Stacking base models

1. Partition `train` into 5 folds
2. Create `train_meta/test_meta`: same `row/fold` IDs as in `train/test`, empty `M1/M2`
3. For each  $\text{Fold}_i \in \{\text{Fold}_1, \dots, \text{Fold}_5\}$ 
  - 3.1 Combine the other 4 folds for training  $\rightarrow \text{Fold}_{-i}$
  - 3.2 Fit each base model to  $\text{Fold}_{-i}$ , predict on  $\text{Fold}_i$ , save predictions to `M1/M2` in `train_meta`
4. Fit each base model to `train`, predict on `test`, save predictions to `M1/M2` in `test_meta`
5. Fit stacking model `S` to `train_meta`, using `M1/M2` as features
6. Use the stacked model `S` to make final predictions on `test_meta`

# Results for base models



# An intermediate conclusion

- › Bootstrapping: a general statistical technique for computing sample functionals (and their variance)
- › Bagging: meta-learner over arbitrary weak algorithms via bootstrap aggregation
- › The Random Forest algorithm: bagging over decision trees
- › Stacked generalization aka stacking: blend output of weak learners (weak signals) with raw features

# Naive boosting for regression

# Boosting for regression

- › Consider a regression problem  $\frac{1}{2} \sum_{i=1}^{\ell} (h(x_i) - y_i)^2 \rightarrow \min_h$
- › Search for solution in the form of weak learner composition  $a_N(x) = \sum_{n=1}^N h_n(x)$  with weak learners  $h_n \in \mathbb{H}$
- › The **boosting approach**: add weak learners greedily
  1. Start with a “trivial” weak learner  $h_0(x) = \frac{1}{\ell} \sum_{i=1}^{\ell} y_i$
  2. At step  $N$ , compute the residuals

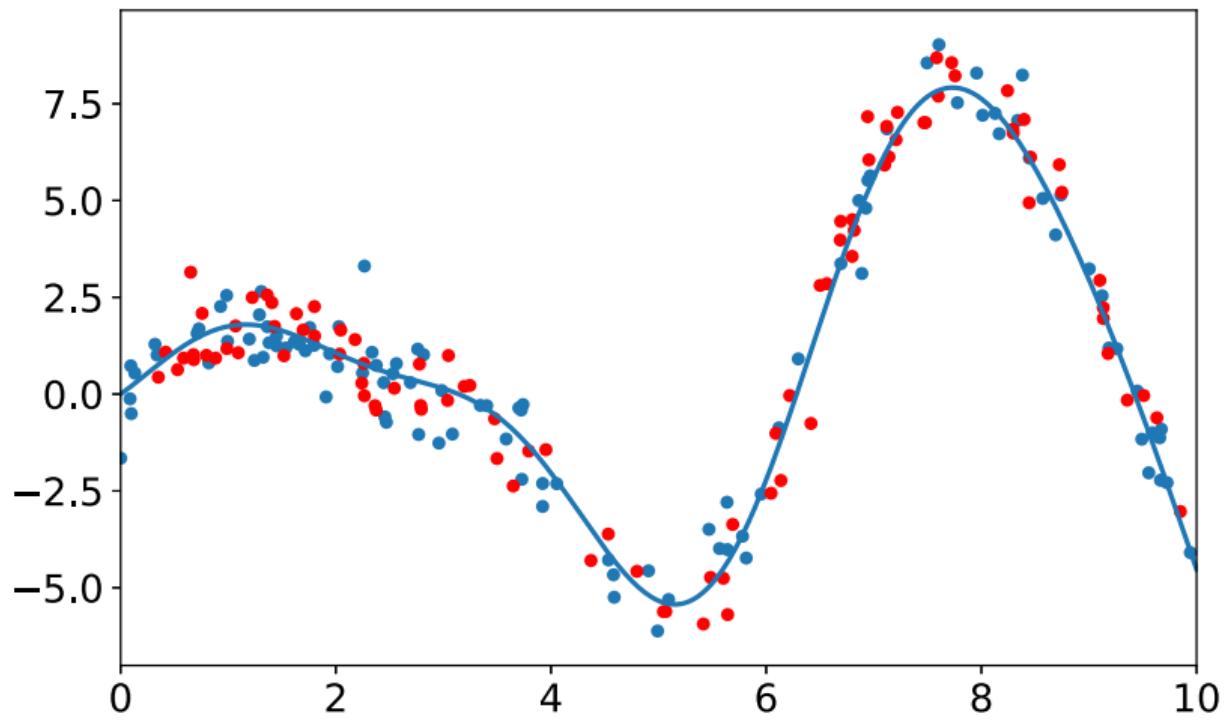
$$s_i^{(N)} = y_i - \sum_{n=1}^{N-1} h_n(x_i) = y_i - a_{N-1}(x_i), \quad i = 1, \dots, \ell$$

3. Learn the next weak algorithm using

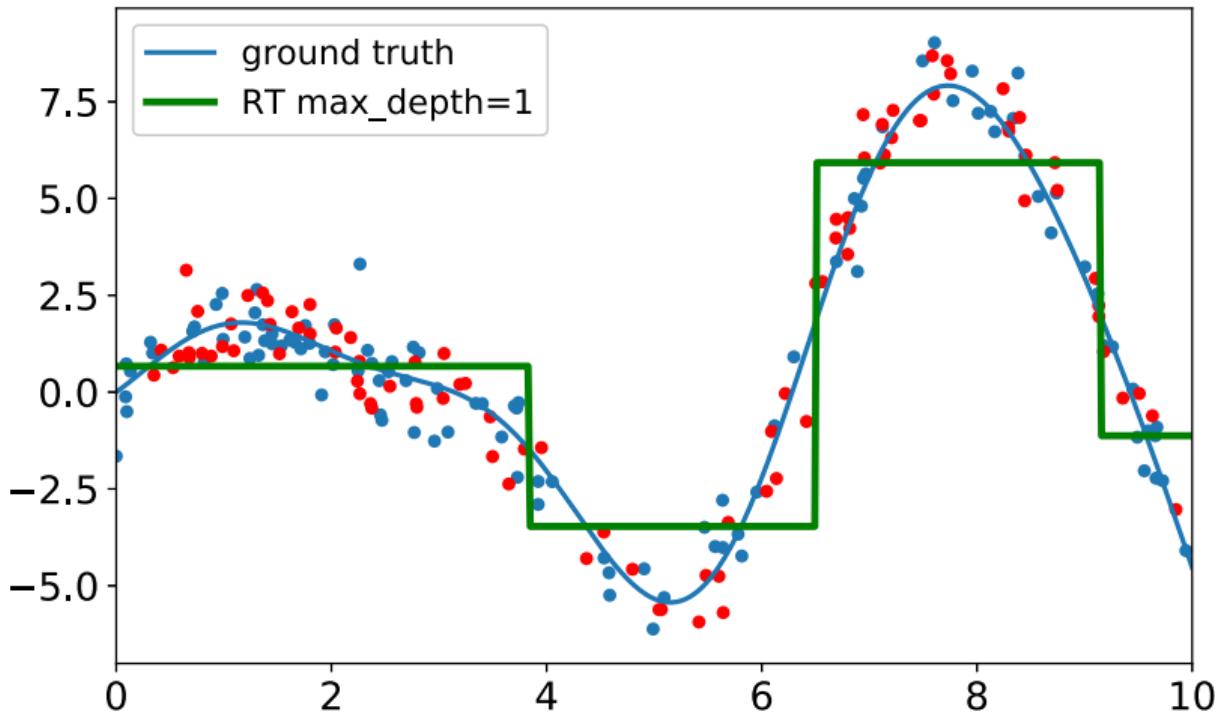
$$a_N(x) := \arg \min_{h \in \mathbb{H}} \frac{1}{2} \sum_{i=1}^{\ell} (h(x_i) - s_i^{(N)})^2$$

Alexey Artemov (this implementation may be found in, e.g., `scikit-learn`)

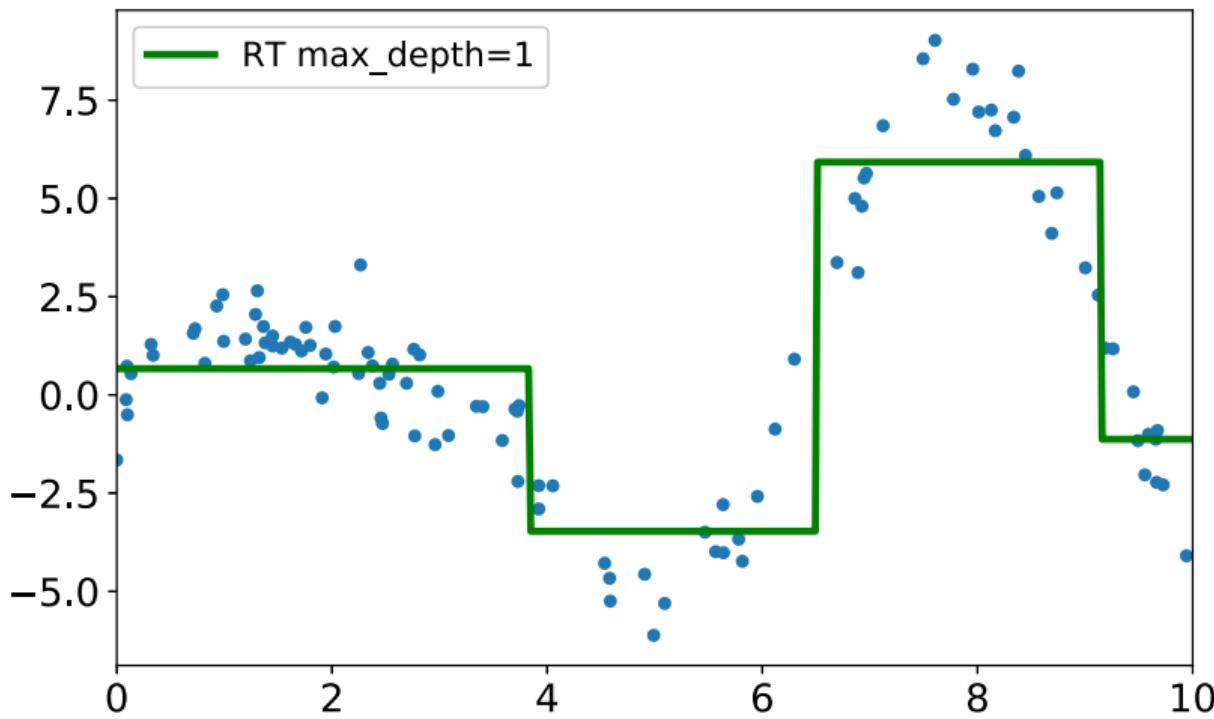
# Boosting: an example regression problem



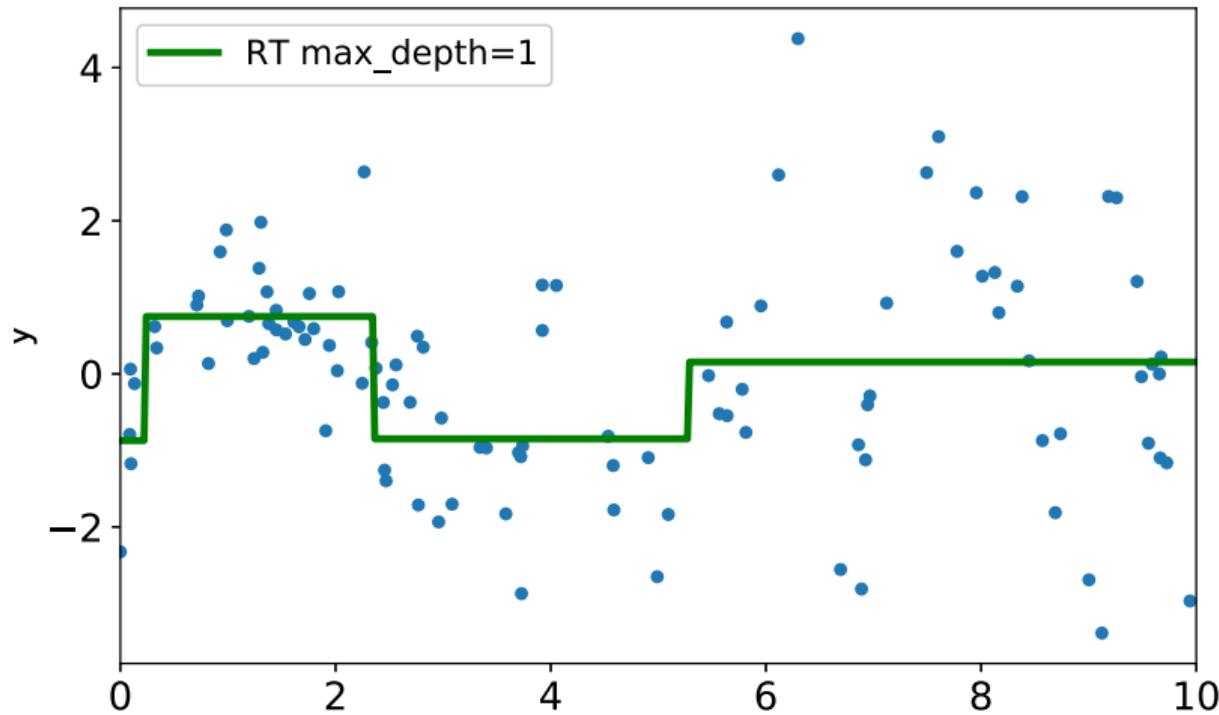
# Boosting: an example regression problem



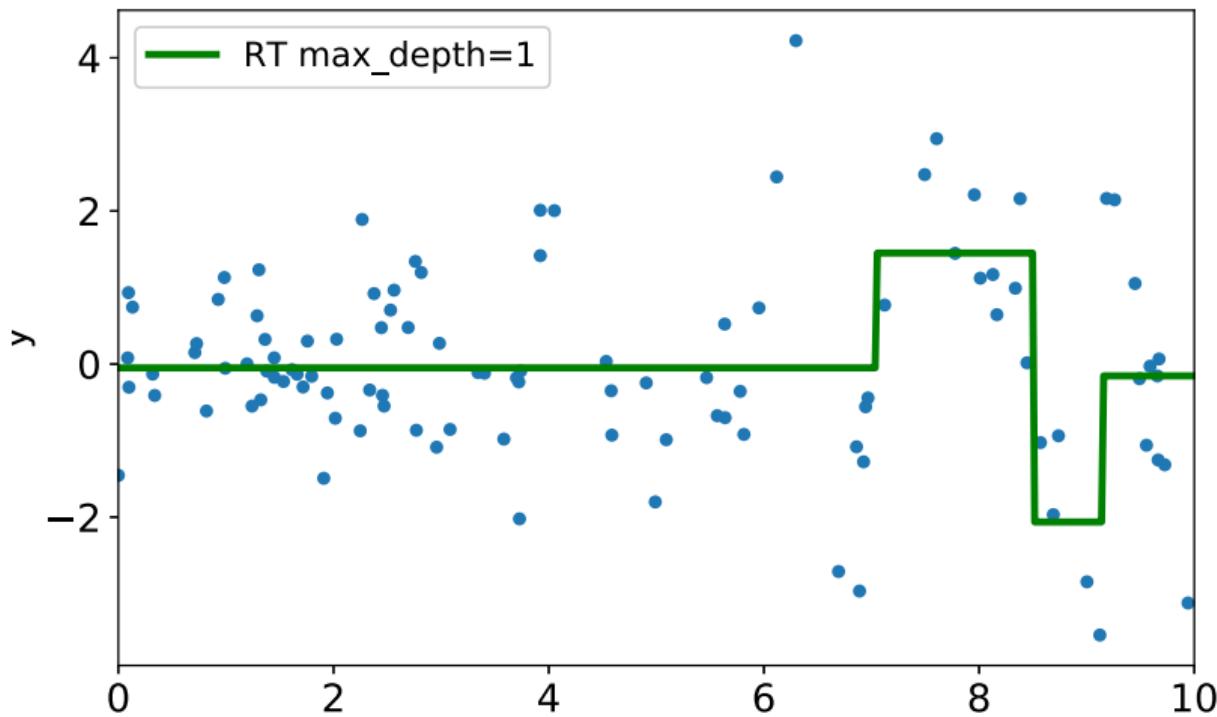
# Boosting: an example regression problem



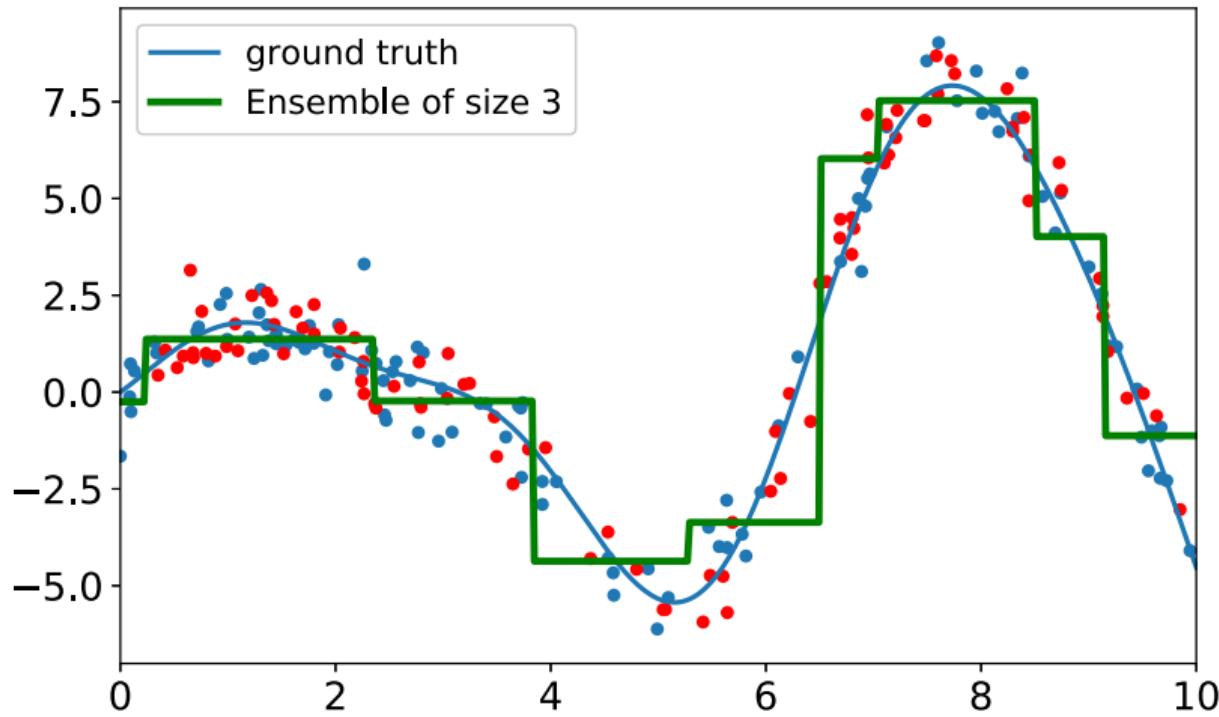
# Boosting: an example regression problem



# Boosting: an example regression problem



# Boosting: an example regression problem

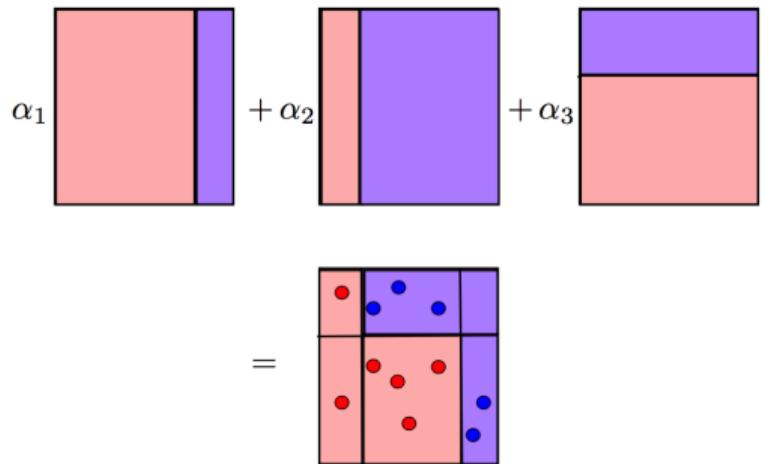
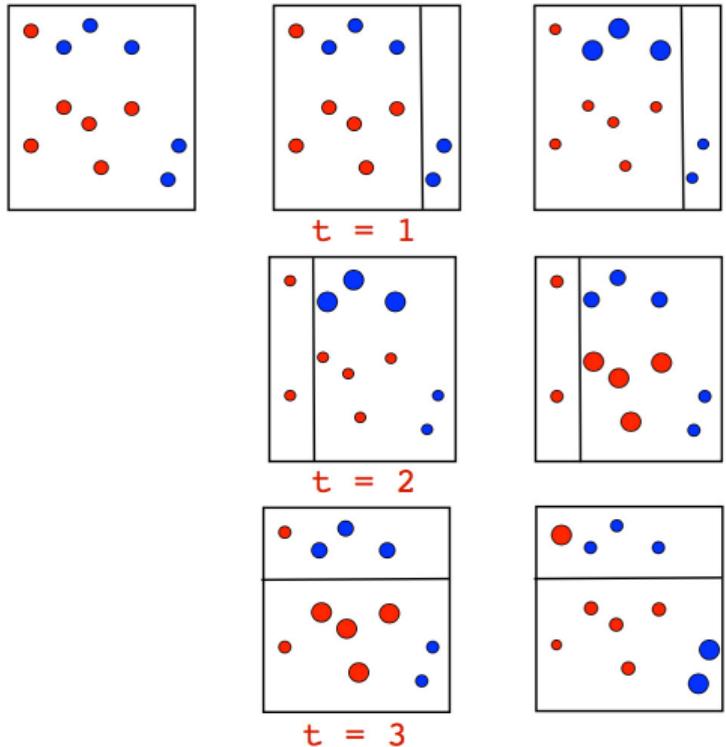


# Reweighting and AdaBoost

# Adaptive boosting for classification

- › Training set  $X^\ell = \{(\mathbf{x}_i, y_i)\}_{i=1}^\ell, y_i \in \{-1, +1\}$
- › Search for solution in the form of a **weighted** sum  $a_N(\mathbf{x}) = \sum_{n=1}^N \gamma_n h_n(\mathbf{x})$  with weak learners  $h_n \in \mathbb{H}$
- › At step  $N$ , extend  $a$  with  $h_N$ :  $a_N(\mathbf{x}_i) = a_{N-1}(\mathbf{x}_i) + \gamma_N h_N(\mathbf{x}_i)$ .  
How do we choose  $h_N$  and its weight  $\gamma_N$ ?
- › Measure fit quality using **exponential loss**  $Q(a_N, X^\ell) = \sum_{i=1}^\ell \exp\{-y_i a_N(\mathbf{x}_i)\}$
- › Derivation yields  $h_N = \arg \min_h \sum_{i=1}^\ell (h(\mathbf{x}_i) - w_i y_i)^2$   
with **weights**  $w_i = \exp\{-y_i a_{N-1}(\mathbf{x}_i)\}$
- › Weak learner weight: minimize  $\gamma_N = \arg \min_\gamma Q(a_{N-1}(\mathbf{x}_i) + \gamma h_N(\mathbf{x}_i), X^\ell)$

# Adaptive boosting for classification



# Adaptive boosting for classification

[Video: AdaBoost in Action]

<https://www.youtube.com/watch?v=k4G2VCu0MMg>

# Gradient boosting

# Gradient boosting: motivation

- With  $a_{N-1}(\mathbf{x})$  already built, how to find the next  $\gamma_N$  and  $h_N$  if

$$\sum_{i=1}^{\ell} L(y_i, a_{N-1}(\mathbf{x}_i) + \gamma h(\mathbf{x}_i)) \rightarrow \min_{\gamma, h}$$

- Recall: functions decrease in the direction of negative gradient
- View  $L(y, z)$  as a function of  $z$  ( $= a_N(\mathbf{x}_i)$ ), execute gradient descent on  $z$
- Search for such  $s_1, \dots, s_\ell$  that

$$\sum_{i=1}^{\ell} L(y_i, a_{N-1}(\mathbf{x}_i) + s_i) \rightarrow \min_{s_1, \dots, s_\ell}$$

- Choose  $s_i = -\left. \frac{\partial L(y_i, z)}{\partial z} \right|_{z=a_{N-1}(\mathbf{x}_i)}$ , approximate  $s_i$ 's by  $h_N(\mathbf{x}_i)$

# The Gradient Boosting Machine [Friedman, 2001]

- › Input:
  - › Training set  $X^\ell = \{(\mathbf{x}_i, y_i)\}_{i=1}^\ell$
  - › Number of boosting iterations  $N$
  - › Loss function  $Q(y, z)$  with its gradient  $\frac{\partial Q}{\partial z}$
  - › A family  $\mathbb{H} = \{h(\mathbf{x})\}$  of weak learners and their associated learning procedures
  - › Additional hyperparameters of weak learners (tree depth, etc.)
- › Initialize GBM  $h_0(\mathbf{x})$  using some simple rule (zero, most popular class, etc.)
- › Execute boosting iterations  $t = 1, \dots, N$  (see next slide)
- › Compose the final GBM learner:  $a_N(\mathbf{x}) = \sum_{t=0}^N \gamma_t h_t(\mathbf{x})$

# The Gradient Boosting Machine [Friedman, 2001]

At every iteration:

1. Compute **pseudo-residuals**:  $s_i = - \left. \frac{\partial Q(y_i, z)}{\partial z} \right|_{z=a_{N-1}(\mathbf{x}_i)}, i = 1, \dots, \ell$
2. Learn  $h_N(\mathbf{x}_i)$  by regressing onto  $s_1, \dots, s_\ell$ :

$$h_N(x) = \arg \min_{h \in \mathbb{H}} \sum_{i=1}^{\ell} (h(\mathbf{x}_i) - s_i)^2$$

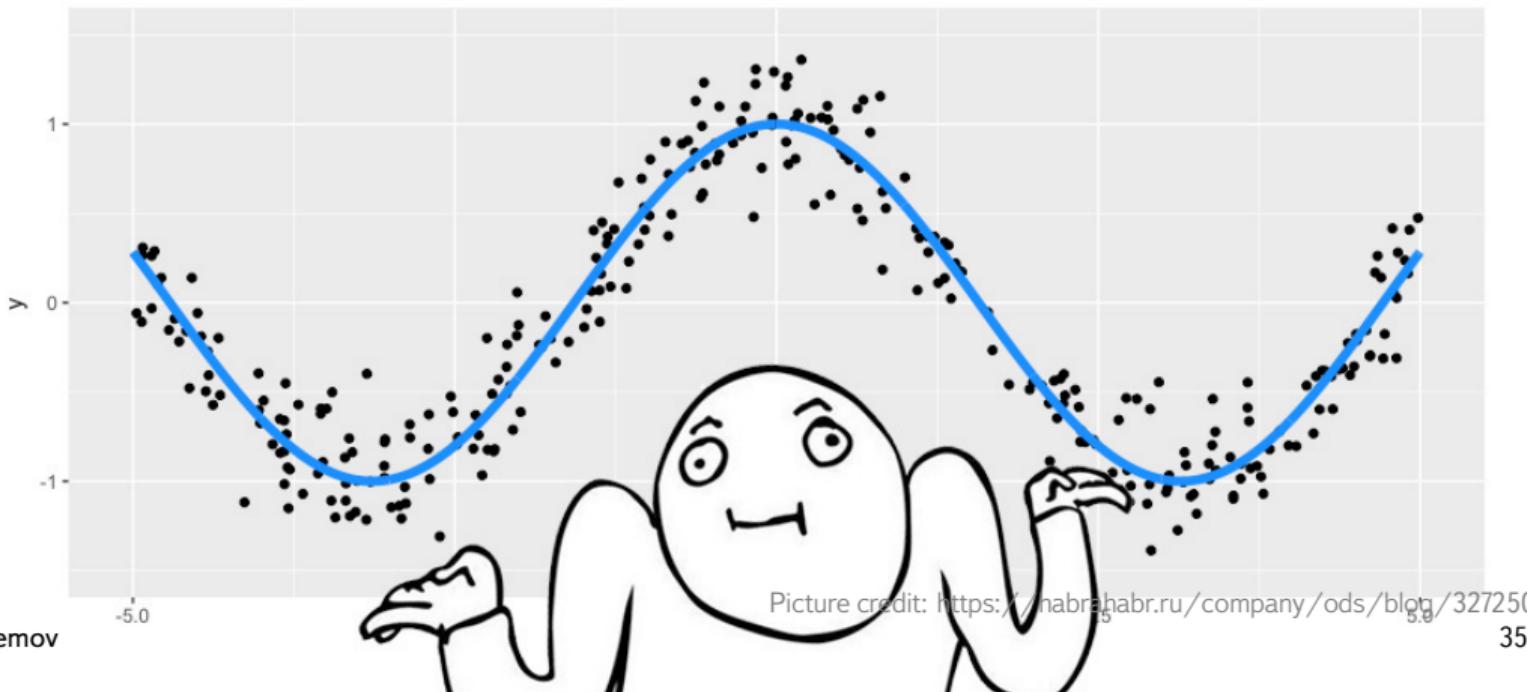
3. Find the optimal  $\gamma_N$  using plain gradient descent:

$$\gamma_N = \arg \min_{\gamma \in \mathbb{R}} \sum_{i=1}^{\ell} Q(y_i, a_{N-1}(\mathbf{x}_i) + \gamma h_N(\mathbf{x}_i))$$

4. Update the GBM by  $a_N(\mathbf{x}_i) \leftarrow a_{N-1}(\mathbf{x}_i) + \gamma_N h_N(\mathbf{x}_i)$

# GBM: an example regression problem

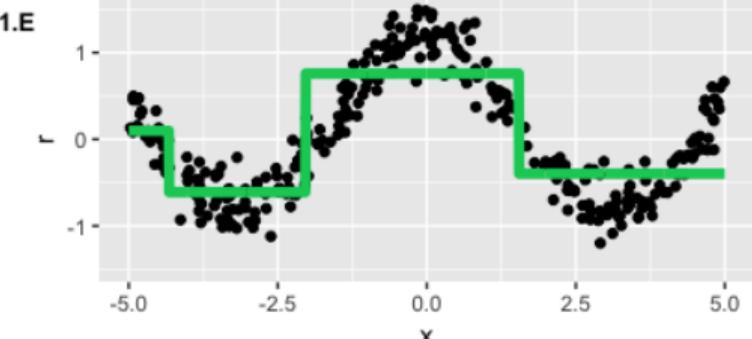
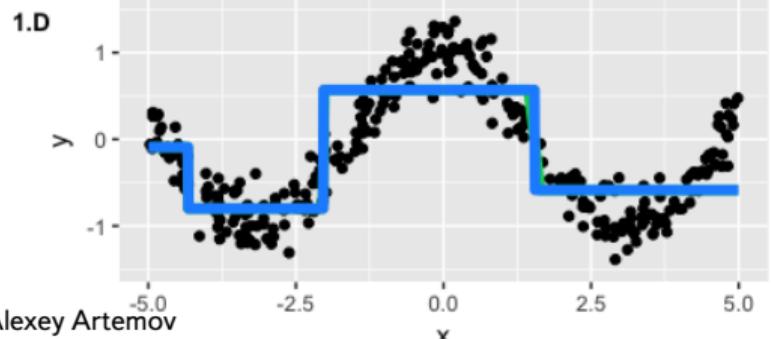
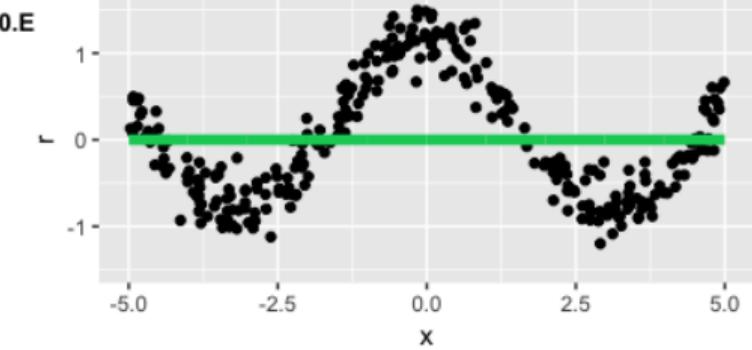
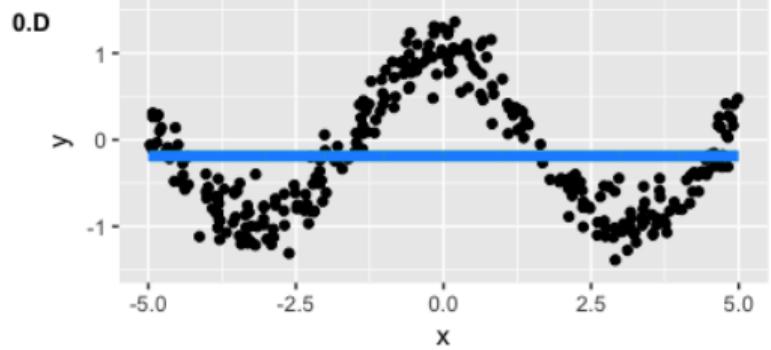
- Consider a training set for a  $X^{300} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{300}$   
where  $x_i \in [-5, 5]$ ,  $y_i = \cos(x_i) + \varepsilon_i$ ,  $\varepsilon_i \sim \mathcal{N}(0, 1/5)$



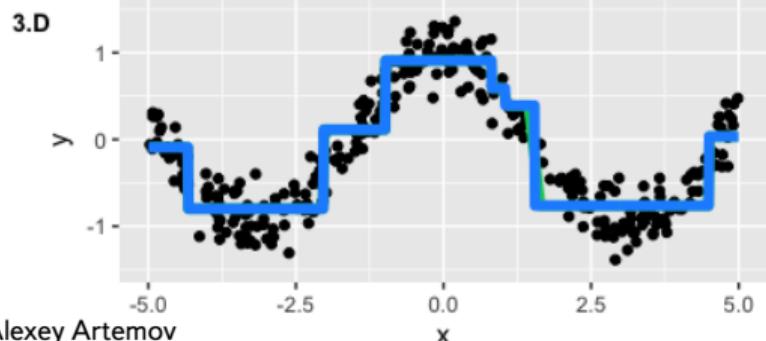
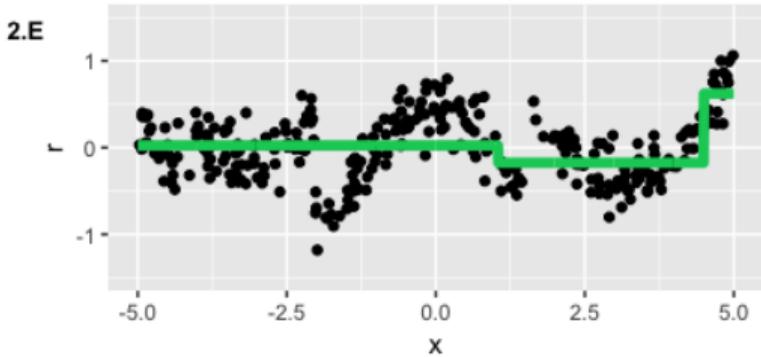
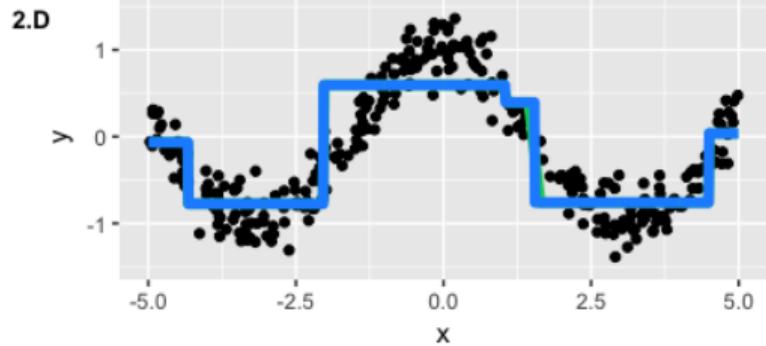
# GBM: an example regression problem

- › Pick  $N = 3$  boosting iterations
- › Quadratic loss  $Q(y, z) = (y - z)^2$
- › Gradient of the quadratic loss  $\frac{\partial Q(y_i, z)}{\partial z} = (y - z)$  is just residuals
- › Pick decision trees as weak learners  $h_i(\mathbf{x})$
- › Set 2 as the maximum depth for decision trees

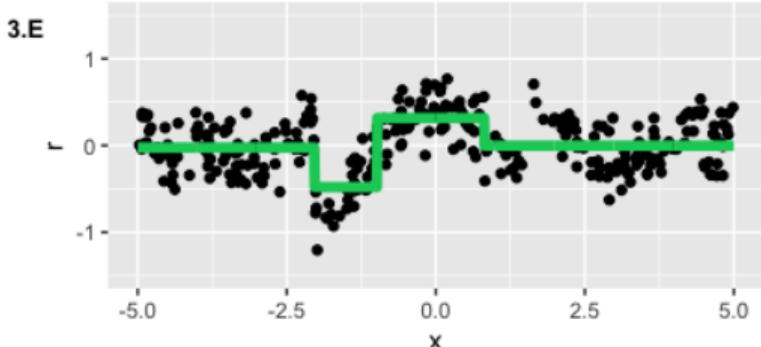
# GBM: an example regression problem



# GBM: an example regression problem

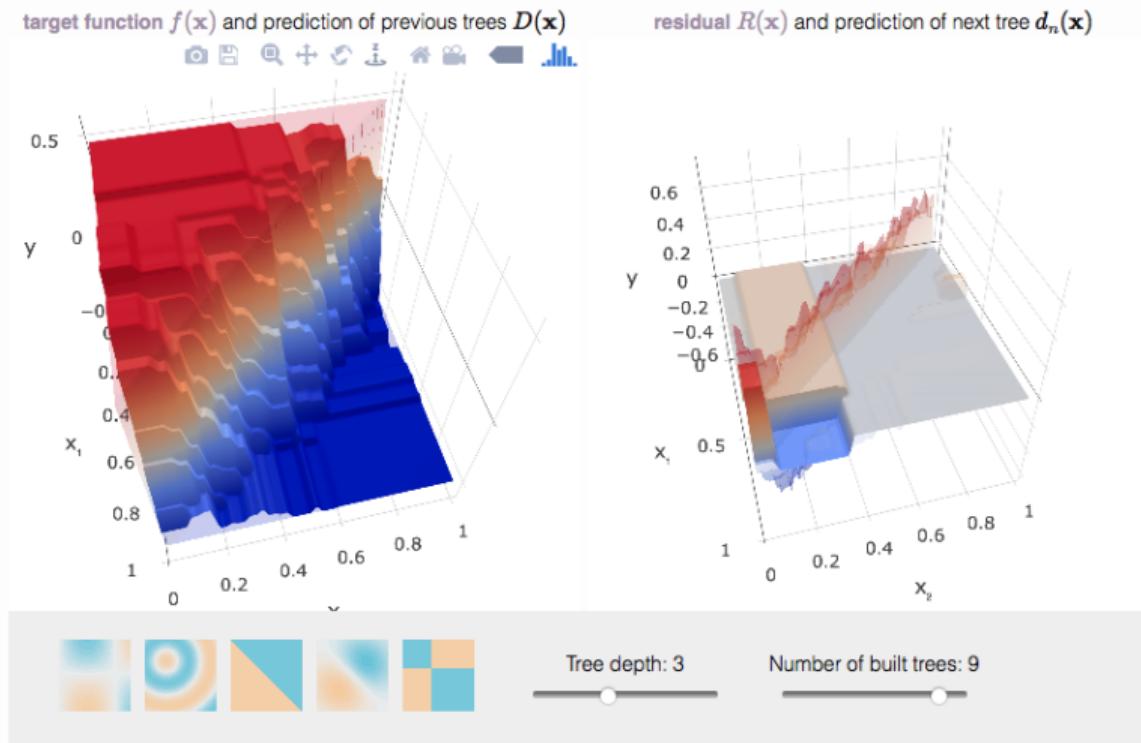


Alexey Artemov



# GBM: an interactive demo

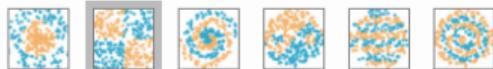
[http://arogozhnikov.github.io/2016/06/24/gradient\\_boosting\\_explained.html](http://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html)



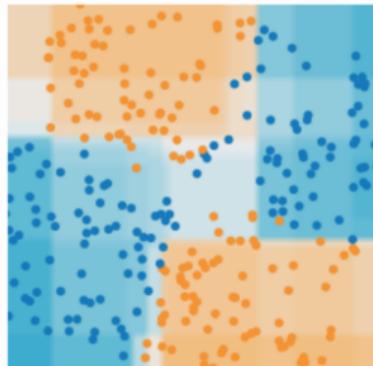
# GBM: an interactive demo

[http://arogozhnikov.github.io/2016/07/05/gradient\\_boosting\\_playground.html](http://arogozhnikov.github.io/2016/07/05/gradient_boosting_playground.html)

Dataset to classify:



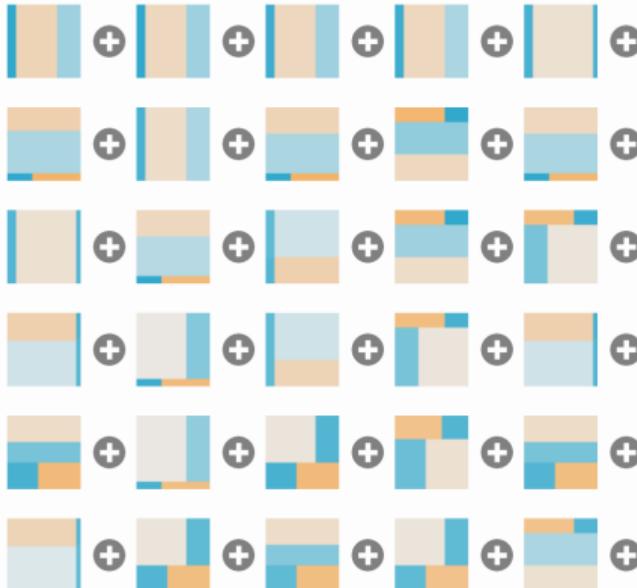
Prediction:



↑  
predictions of GB (all 50 trees)

train loss: 0.266      test loss: 0.312

Decision functions of first 30 trees



# GBM: regularization via shrinkage

- › For **too simple weak learners**, the negative gradient is approximated badly  $\implies$  random walk in space of samples
- › For **too complex weak learners**, a few boosting steps may be enough for overfitting
- › **Shrinkage:** make shorter steps using a learning rate  $\eta \in (0, 1]$

$$a_N(\mathbf{x}_i) \leftarrow a_{N-1}(\mathbf{x}) + \eta \gamma_N h_N(\mathbf{x})$$

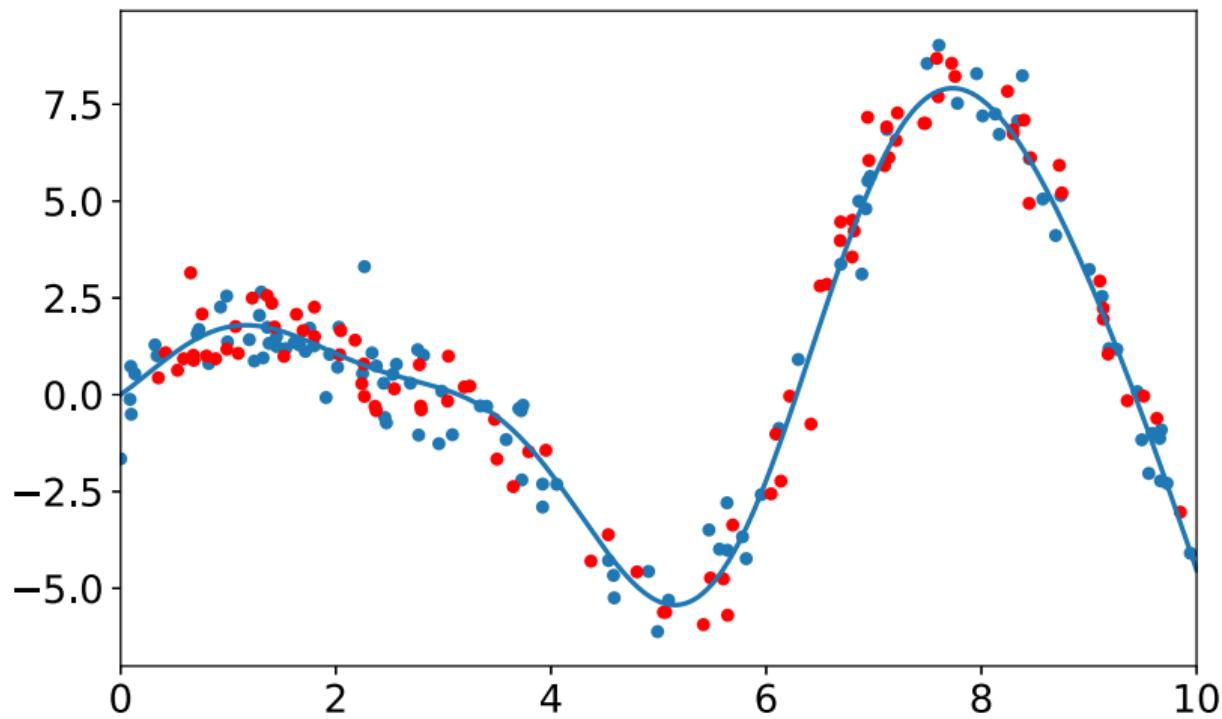
(effectively distrust gradient direction estimated via weak learners)

# GBM: shrinkage animated

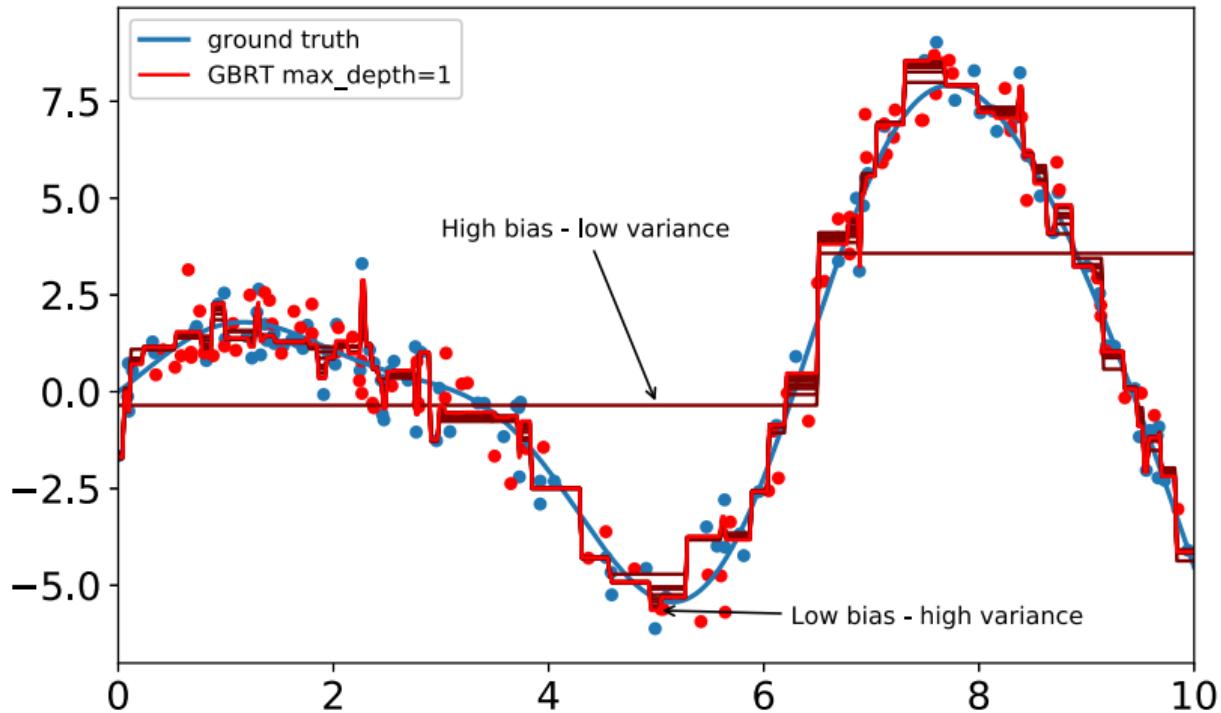
[Figure: High shrinkage](#)

[Figure: Low shrinkage](#)

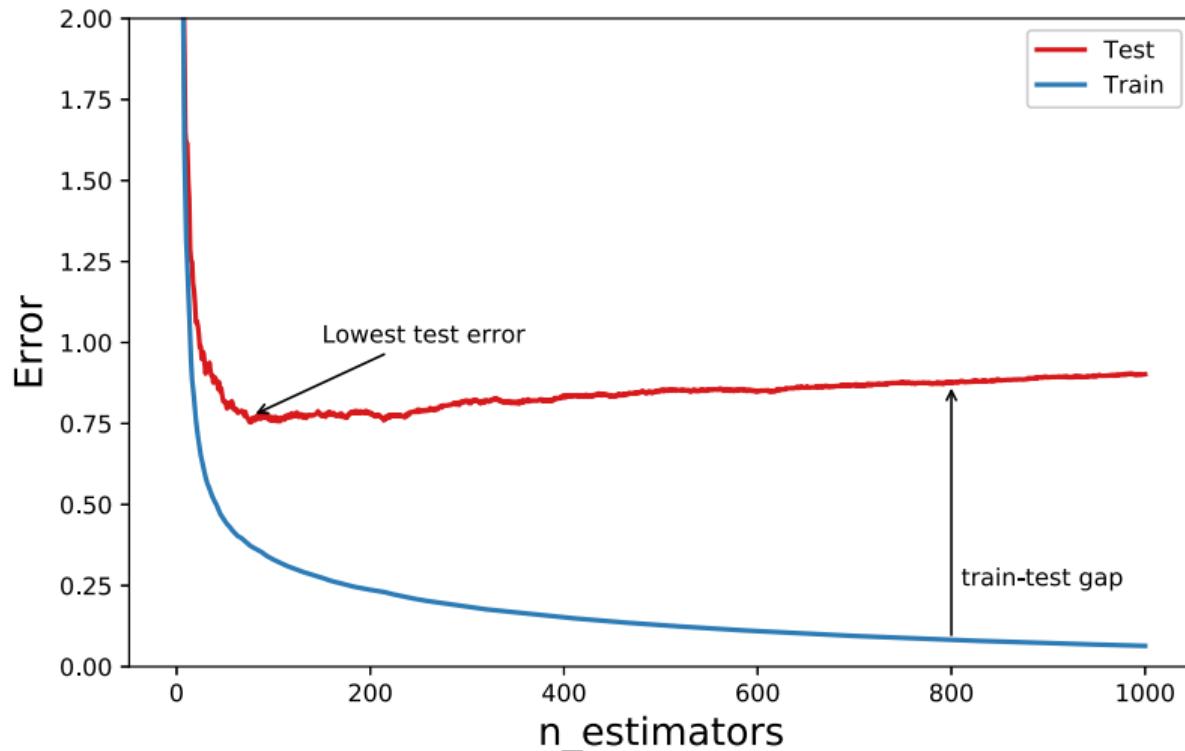
# GBM: regularization approaches



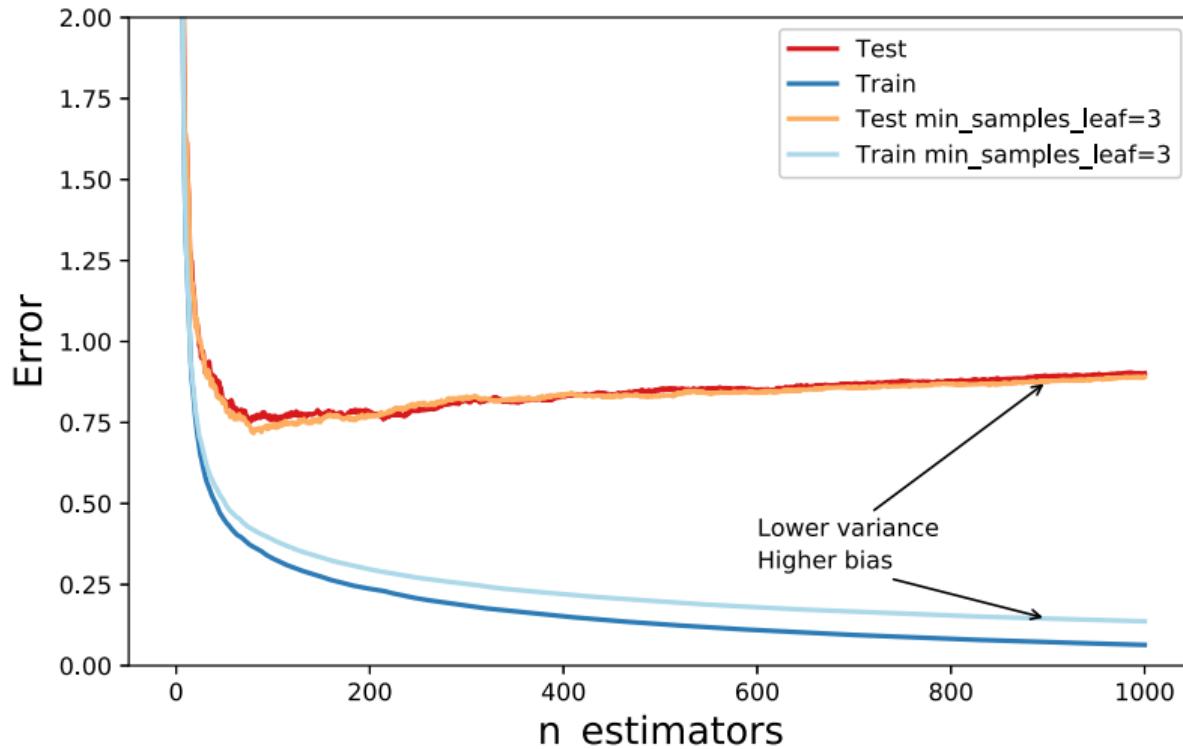
# GBM: regularization approaches



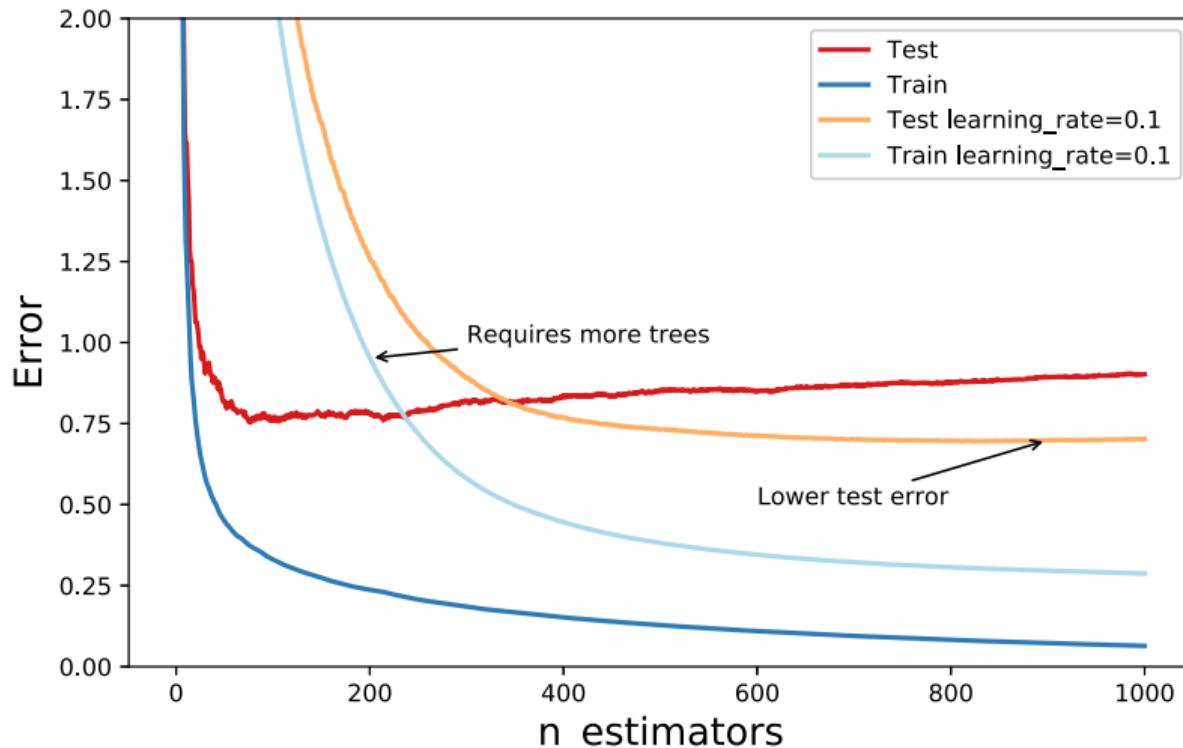
# GBM: regularization approaches



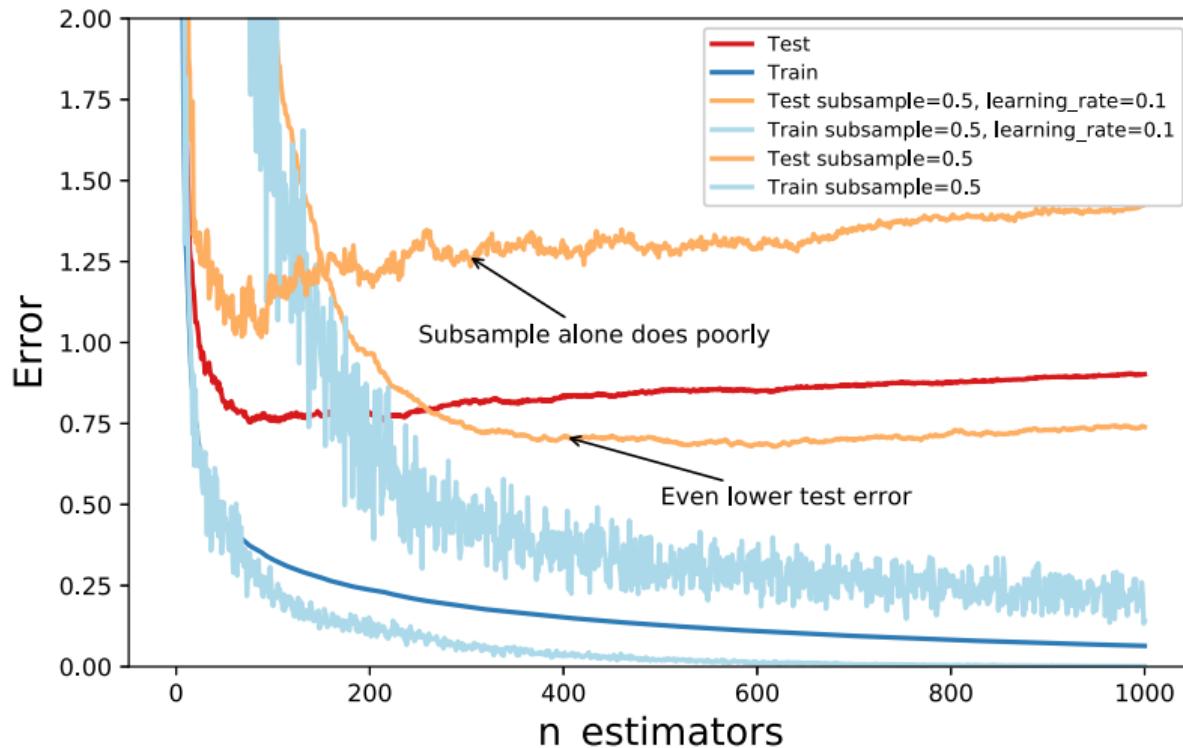
# GBM: regularization approaches



# GBM: regularization approaches



# GBM: regularization approaches



# XGBoost algorithm

# Extreme Gradient Boosting

1. Approximate the descent direction constructed using second order derivatives

$$\sum_{i=1}^{\ell} \left( -s_i h(\mathbf{x}_i) + \frac{1}{2} t_i h^2(\mathbf{x}_i) \right) \rightarrow \min_h, \quad t_i = \left. \frac{\partial^2}{\partial z^2} L(y_i, z) \right|_{a_{N-1}(\mathbf{x}_i)}$$

2. Penalize large leaf counts  $J$  and large leaf coefficient norm  $\|b\|_2^2 = \sum_{j=1}^J b_j^2$

$$\sum_{i=1}^{\ell} \left( -s_i h(x_i) + \frac{1}{2} t_i h^2(x_i) \right) + \gamma J + \frac{\lambda}{2} \sum_{j=1}^J b_j^2 \rightarrow \min_h$$

where  $b(\mathbf{x}) = \sum_{j=1}^J b_j [\mathbf{x} \in R_j]$

# Extreme Gradient Boosting

3. Choose split  $[x_j < t]$  at node  $R$  to maximize

$$Q = H(R) - H(R_\ell) - H(R_r) \rightarrow \max,$$

where the impurity criterion

$$H(R) = -\frac{1}{2} \left( \sum_{(t_i, s_i) \in R} s_j \right)^2 \Bigg/ \left( \sum_{(t_i, s_i) \in R} t_j + \lambda \right) + \gamma$$

4. The stopping rule: declare the node a leaf if even the best split gives negative  $Q$

# An intermediate conclusion

- › **Boosting:** a general meta-algorithm aimed at composing a strong hypothesis from multiple weak hypotheses
- › Boosting can be applied for arbitrary losses, arbitrary problems (regression, classification) and over arbitrary weak learners
- › The **Gradient Boosting Machine:** a general approach to boosting adding weak learners that approximate gradient of the loss function
- › **AdaBoost:** gradient boosting with an exponential loss function resulting in reweighting training instances when adding weak learners
- › **XGBoost:** gradient boosting with second order optimization, penalized loss and particular choice of impurity criterion

# Boosting to uniformity

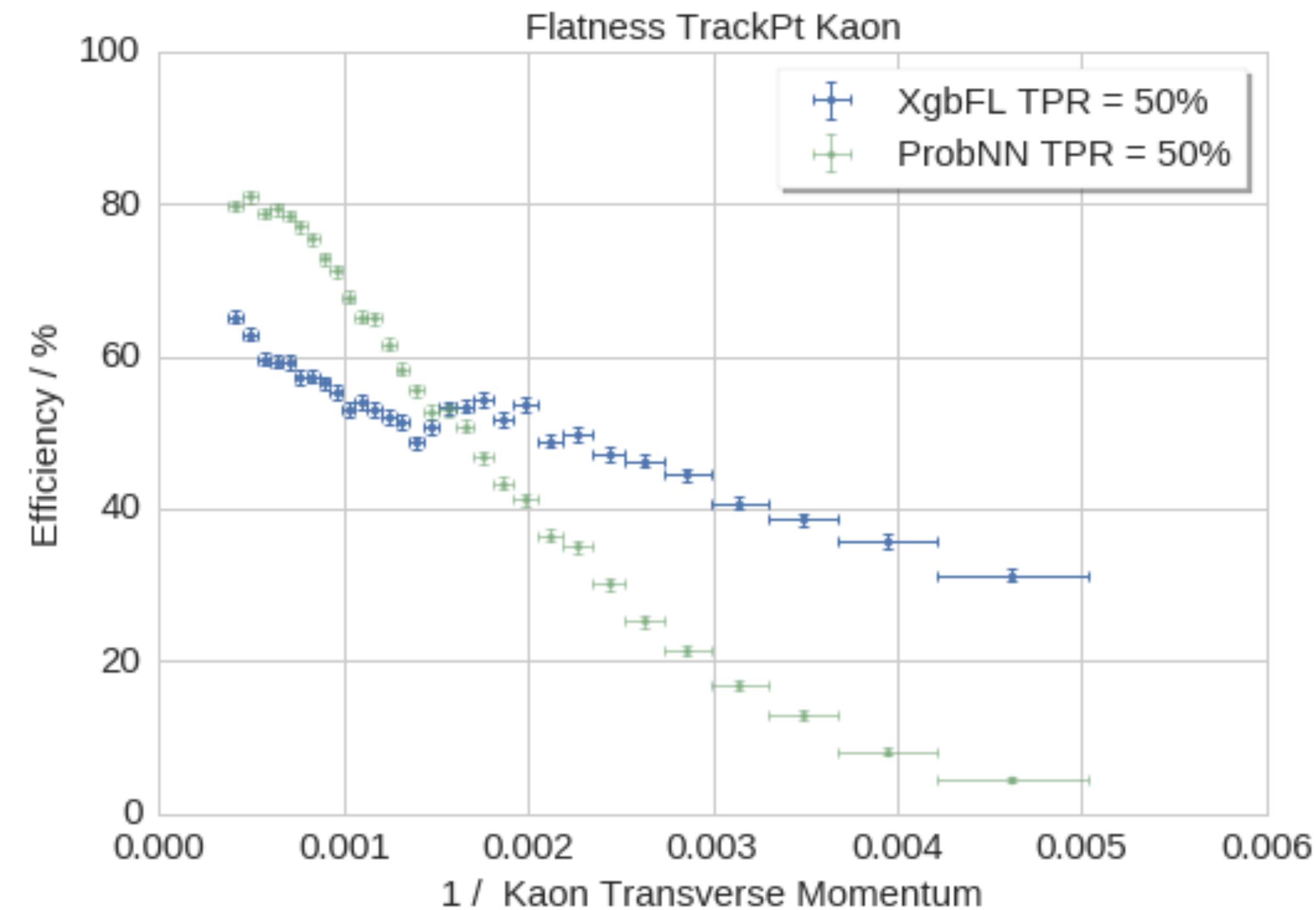


# Uniformity

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

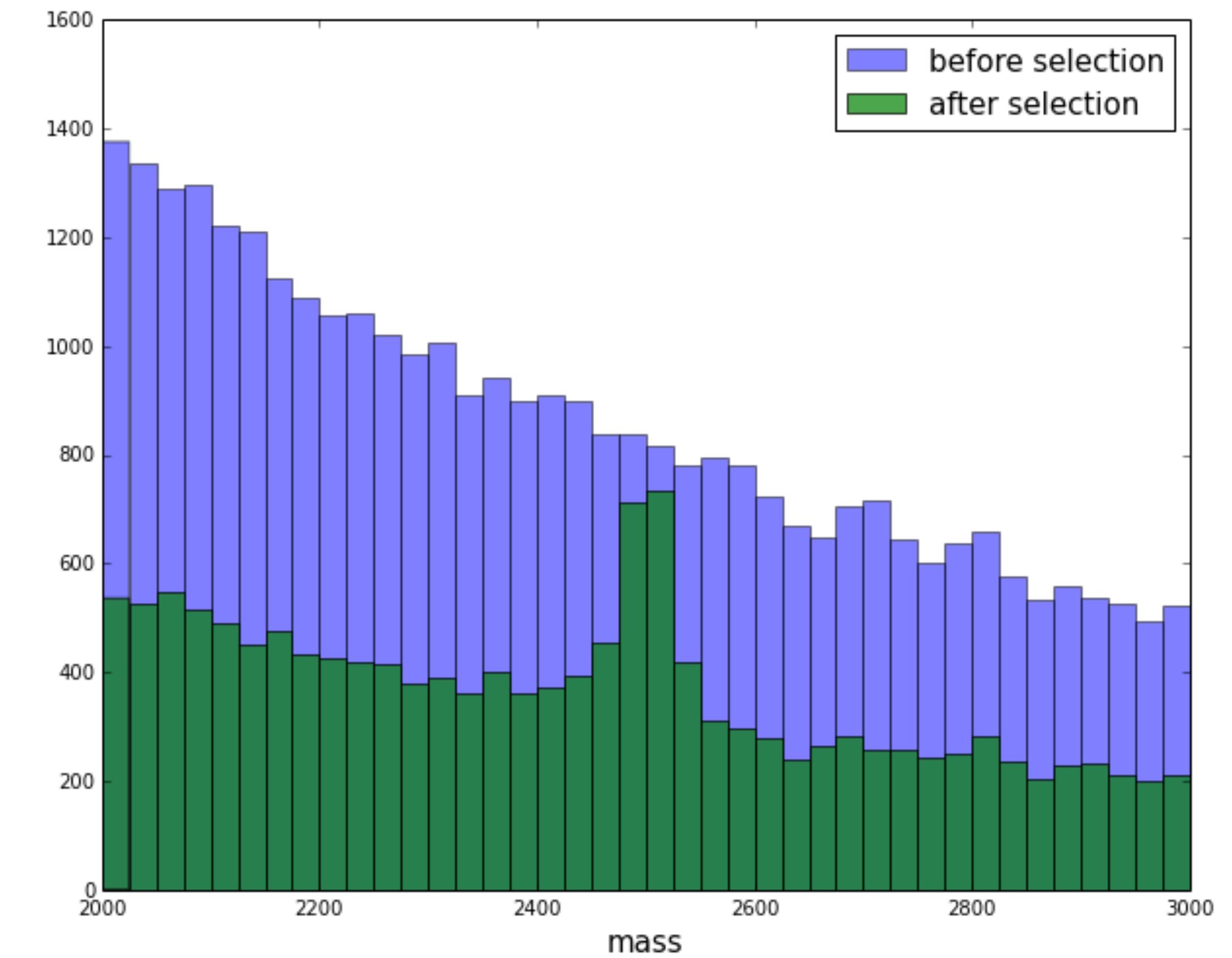


# Non-flatness along the mass

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

Goal:  $FPR = \text{const}$  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:  $\mathcal{L} = \sum_i e^{-y_i D(x_i)}, \quad y_i = \pm 1$
- › LogLoss:  $\mathcal{L} = \sum_i \log(1 + e^{-y_i D(x_i)}), \quad y_i = \pm 1$

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

# uBoostBDT

- › Aims to get  $\text{FPR}_{\text{region}} = \text{const}$
- › Fix target efficiency, for example  $\text{FPR}_{\text{target}} = 30\%$ , and find corresponding threshold
- › train a tree, its decision function is  $d(x)$
- › increase weight for misclassified events:  $w_i \leftarrow w_i \exp(-\alpha y_i d(x_i))$
- › increase weight of background events **in the regions with high FPR**
$$w_i \leftarrow w_i \exp(\beta(\text{FPR}_{\text{region}} - \text{FPR}_{\text{target}}))$$
- › This way we achieve  $\text{FPR}_{\text{region}} = 30\%$  in all regions only for some threshold on training dataset

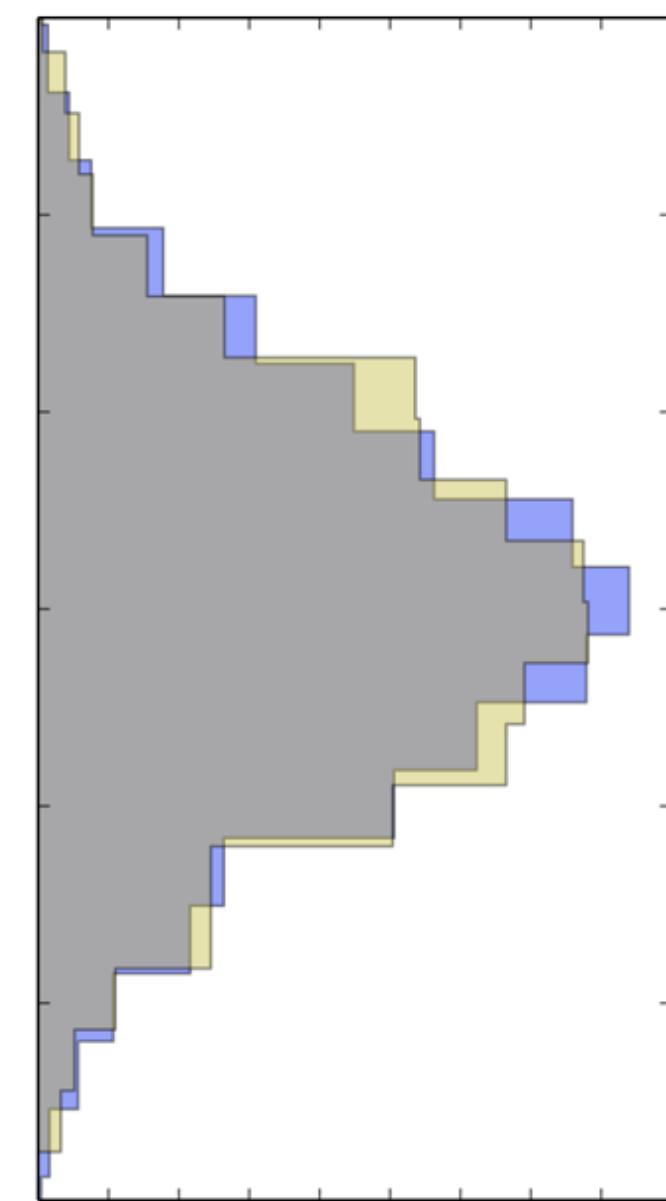
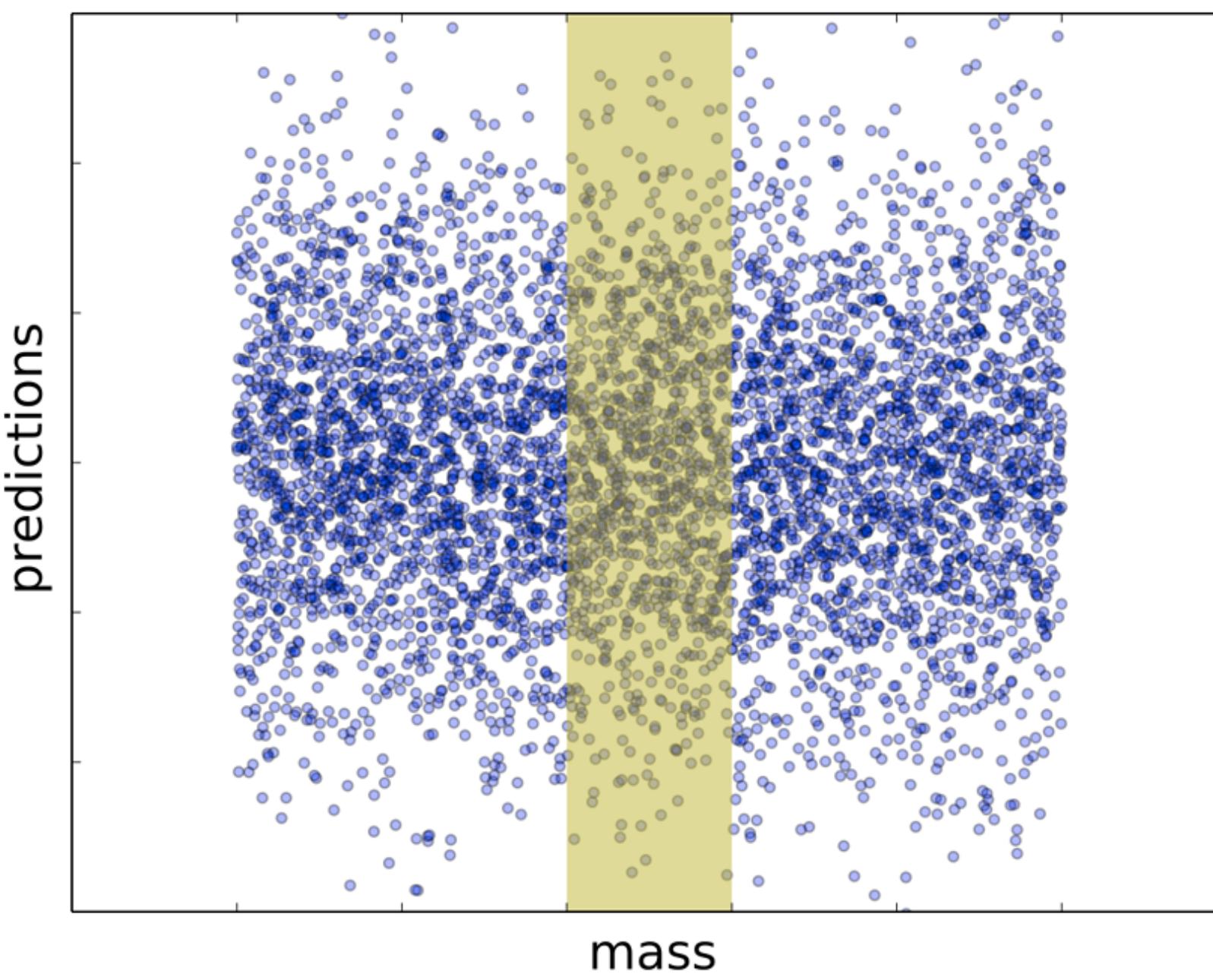
# uBoost

uBoost is an ensemble of uBoostBDTs, each uBoostBDT uses own  $FPR_{target}$  (all possible FPRs with step of 1%)

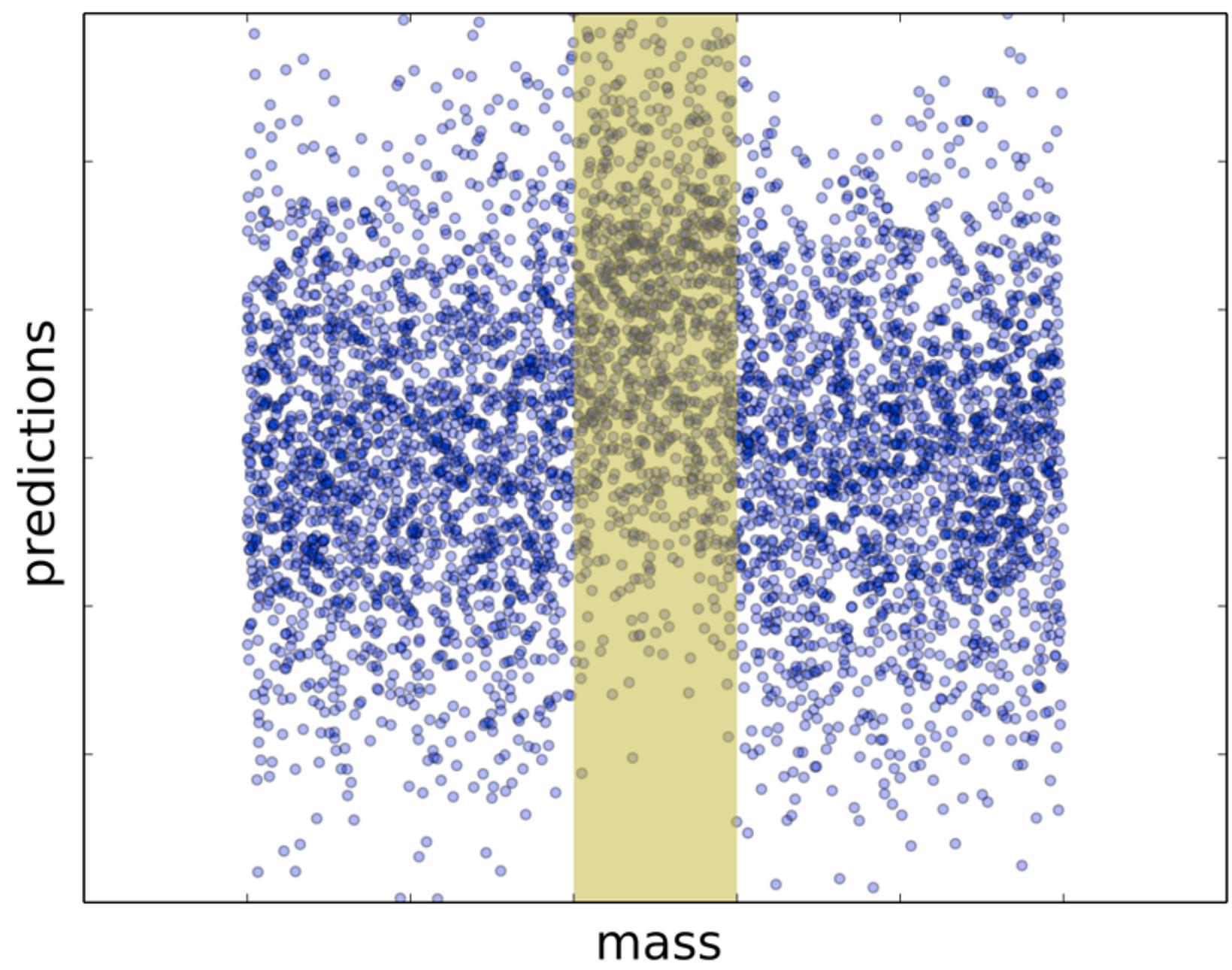
uBoostBDT returns 0 or 1 (passed or not the threshold corresponding to  $FPR_{target}$ ), simple averaging is used to obtain predictions.

- › drives to uniform selection
- › very complex training
- › many trees
- › estimation of threshold in uBoostBDT may be biased

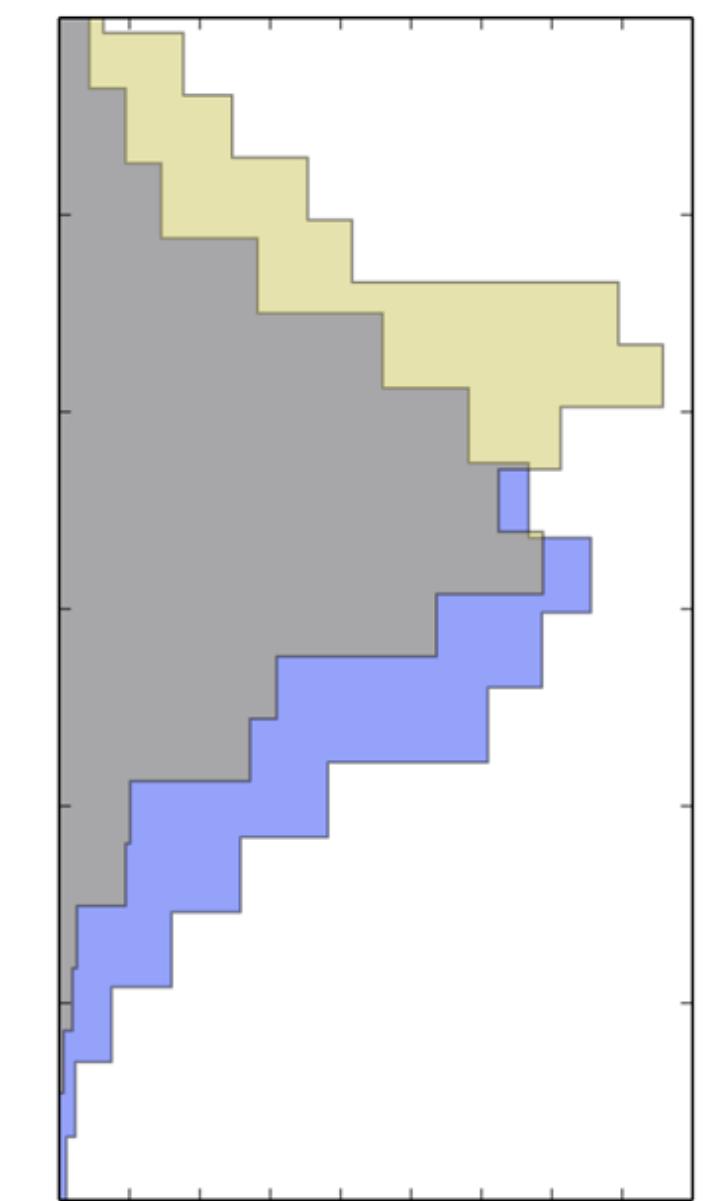
# Non-uniformity measure



Uniform predictions



mass



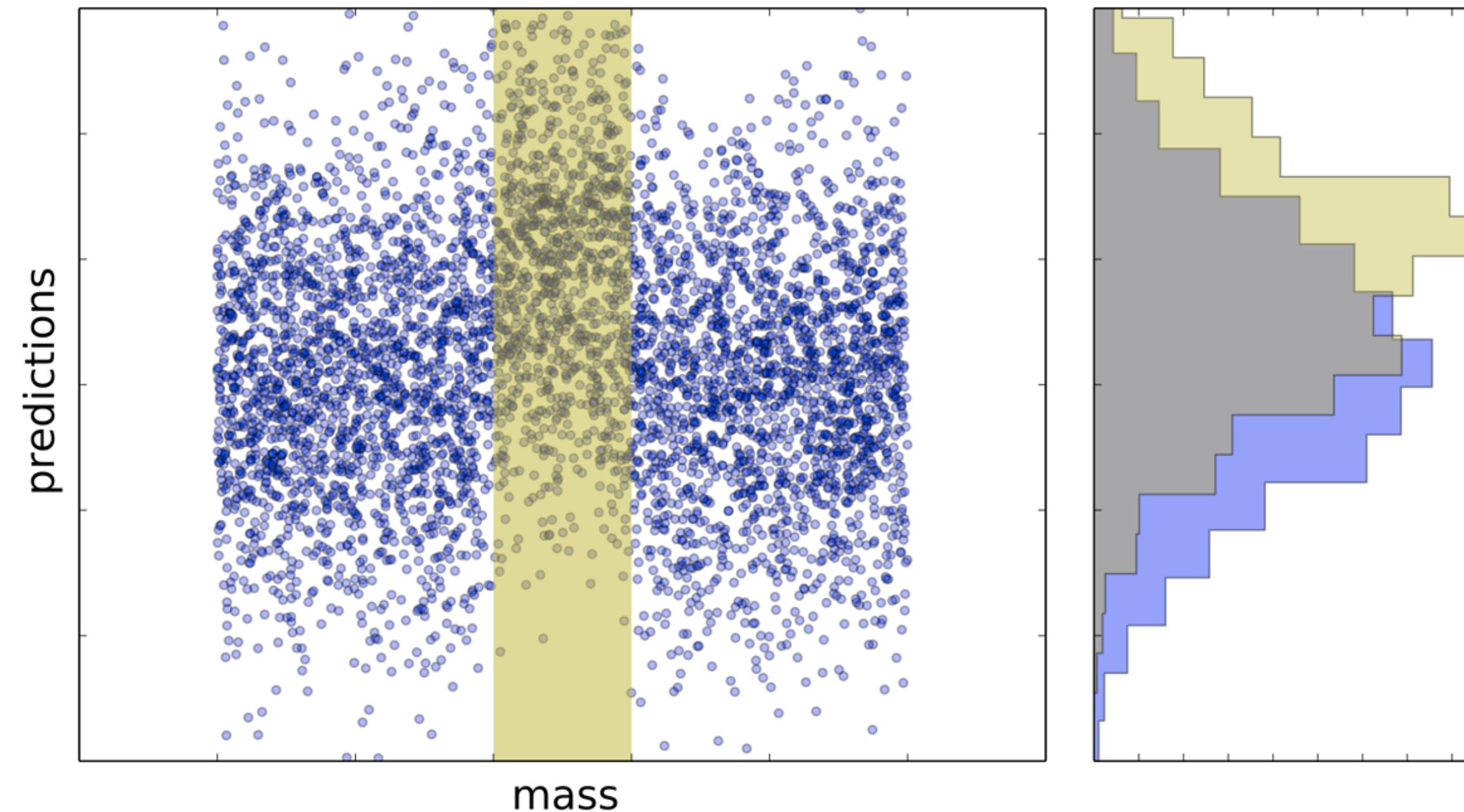
Non-uniform predictions  
(peak in highlighted  
region)

- › 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)

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



# 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:

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

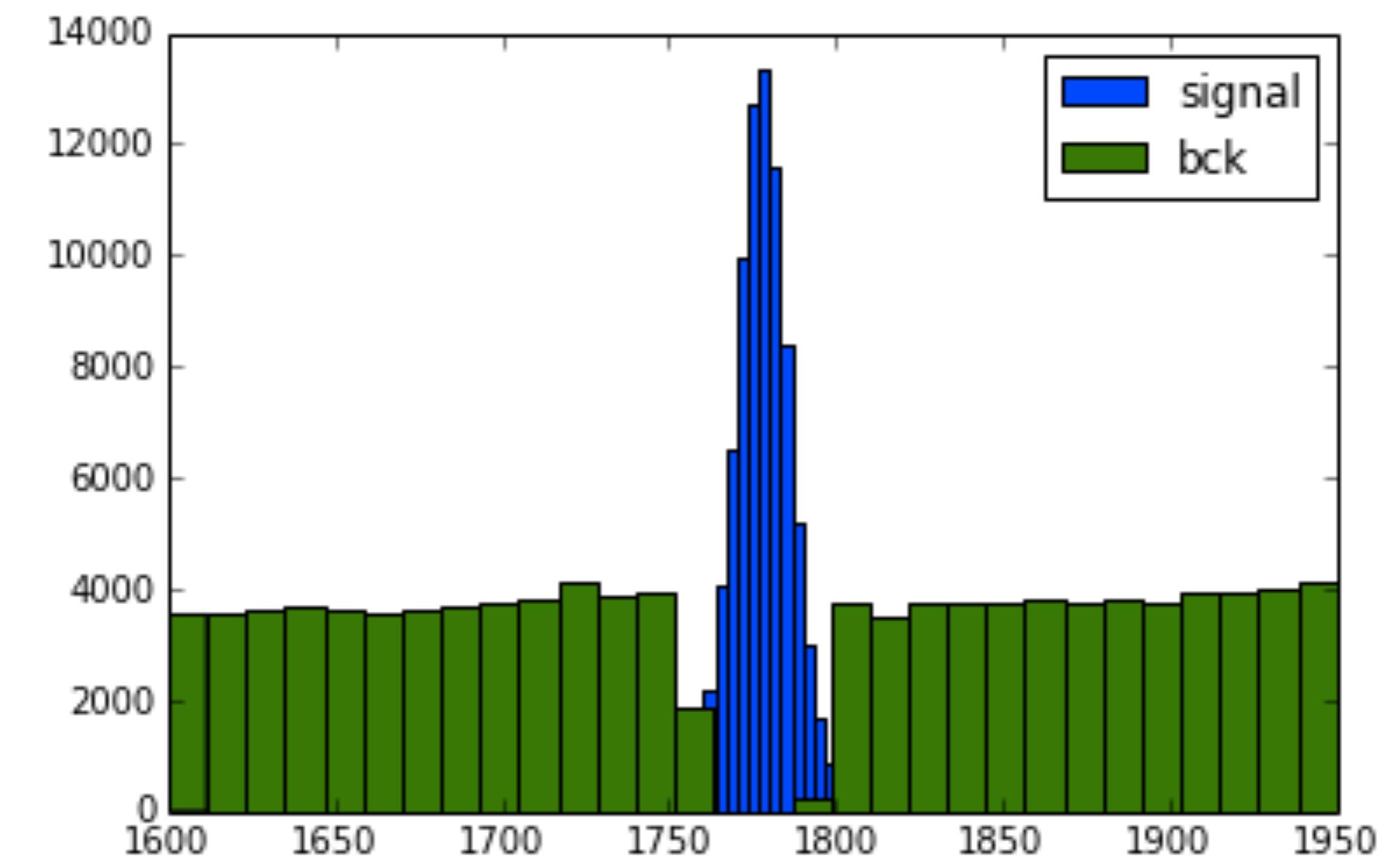
- › Flatness loss approximates non-differentiable CvM measure:

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

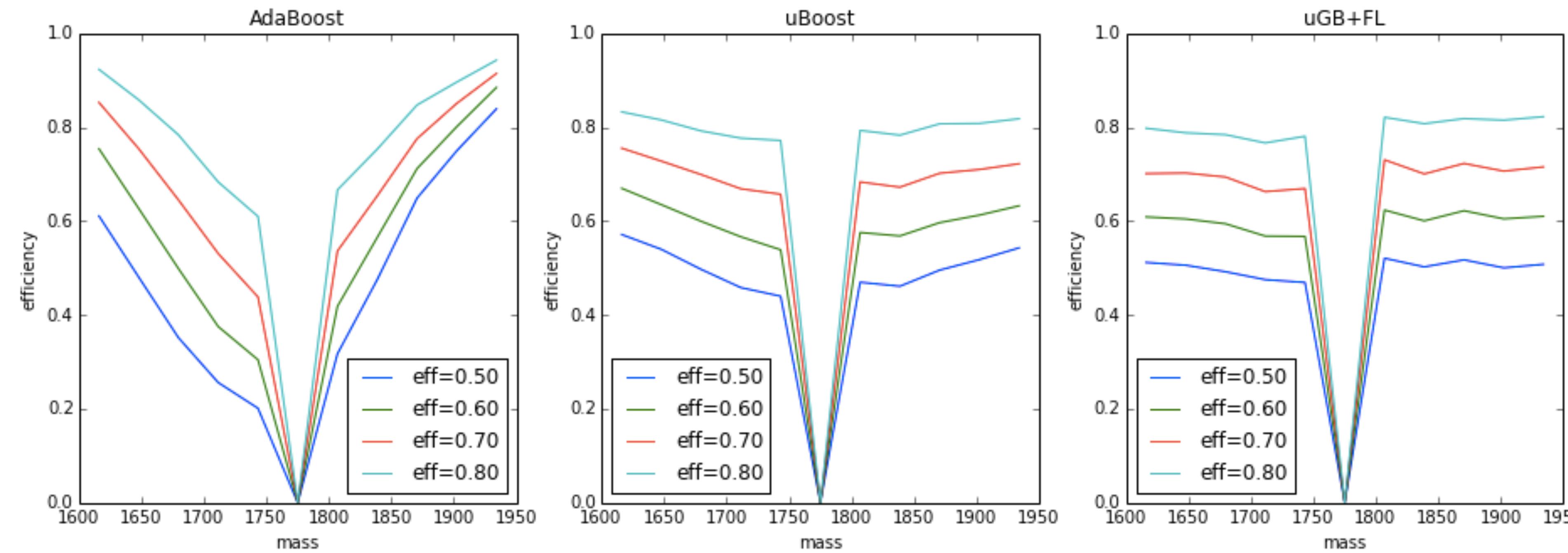
$$\frac{\partial}{\partial D(x_i)} \mathcal{L}_{\text{FL}} \sim 2(F_{\text{region}}(s) - F_{\text{global}}(s)) \Big|_{s=D(x_i)}$$

# 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)

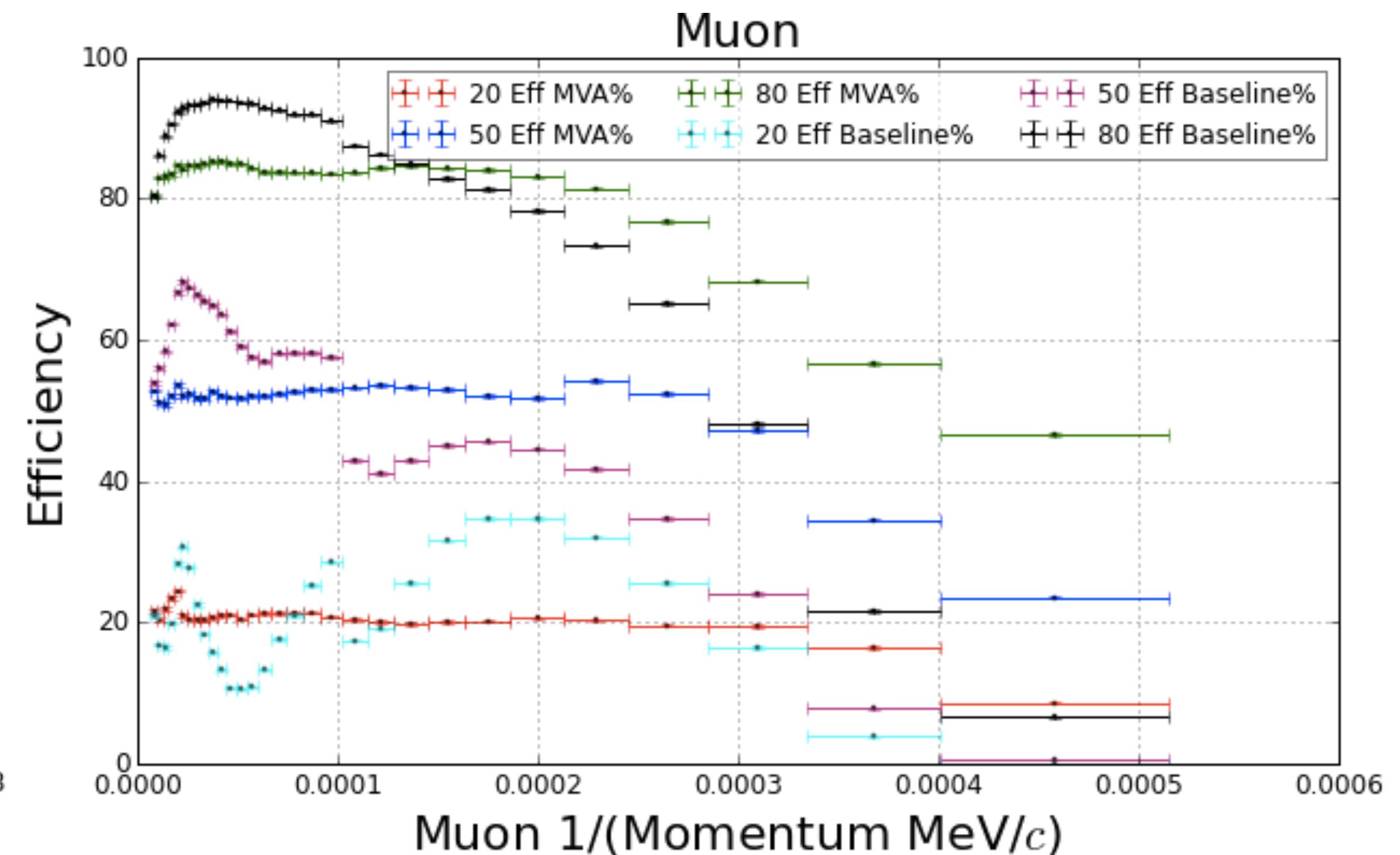
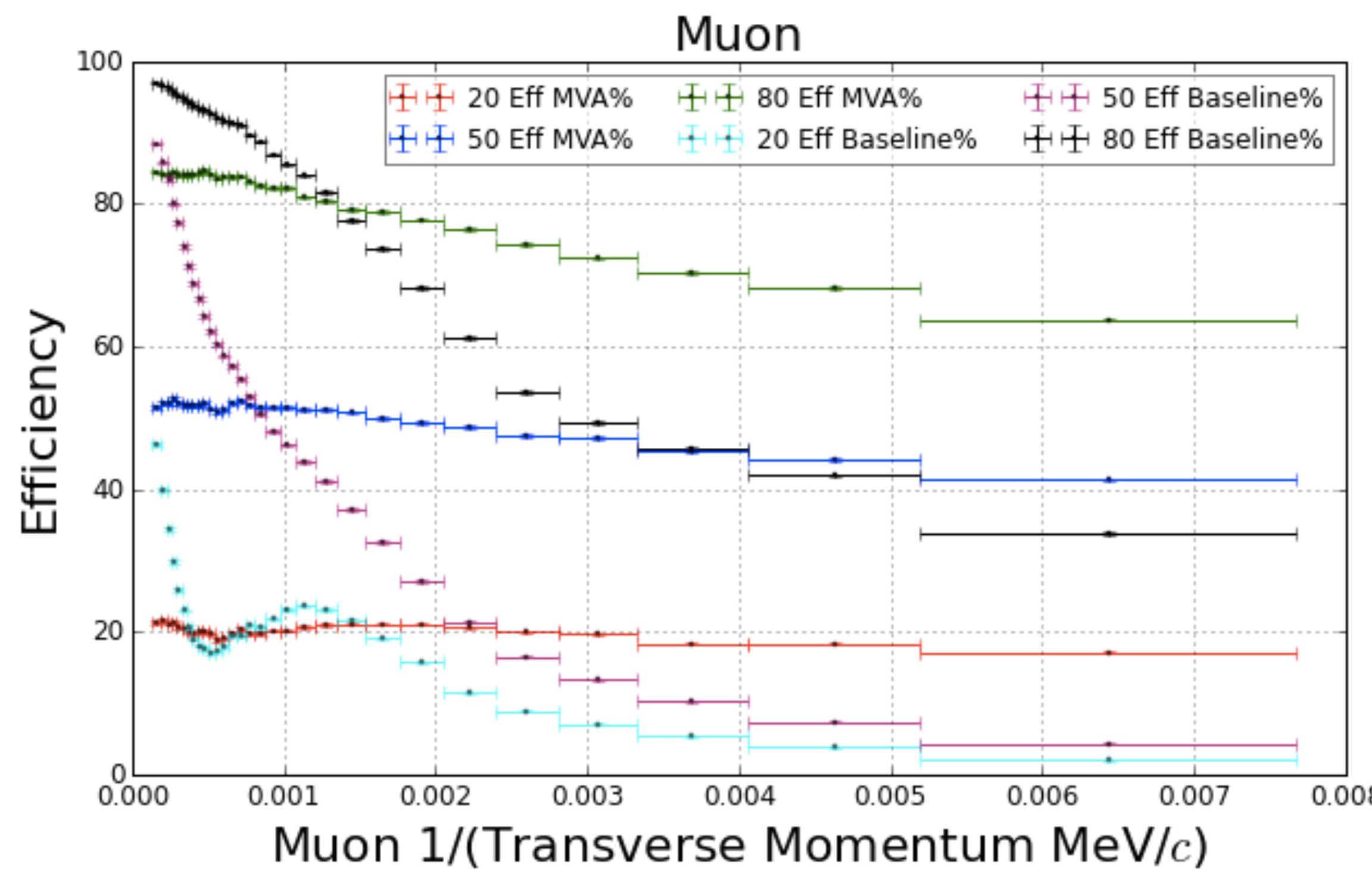


# Rare decay analysis DEMO



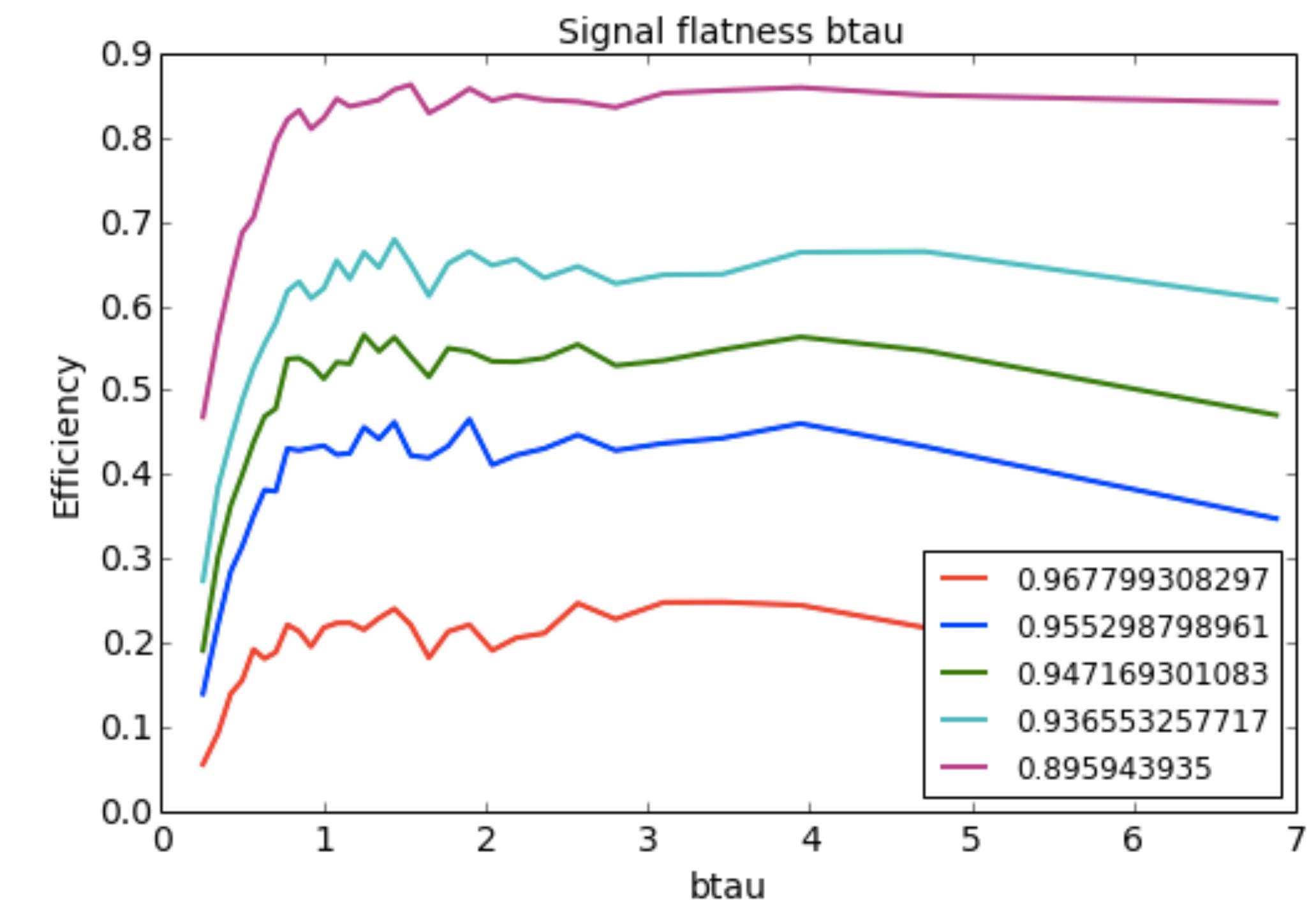
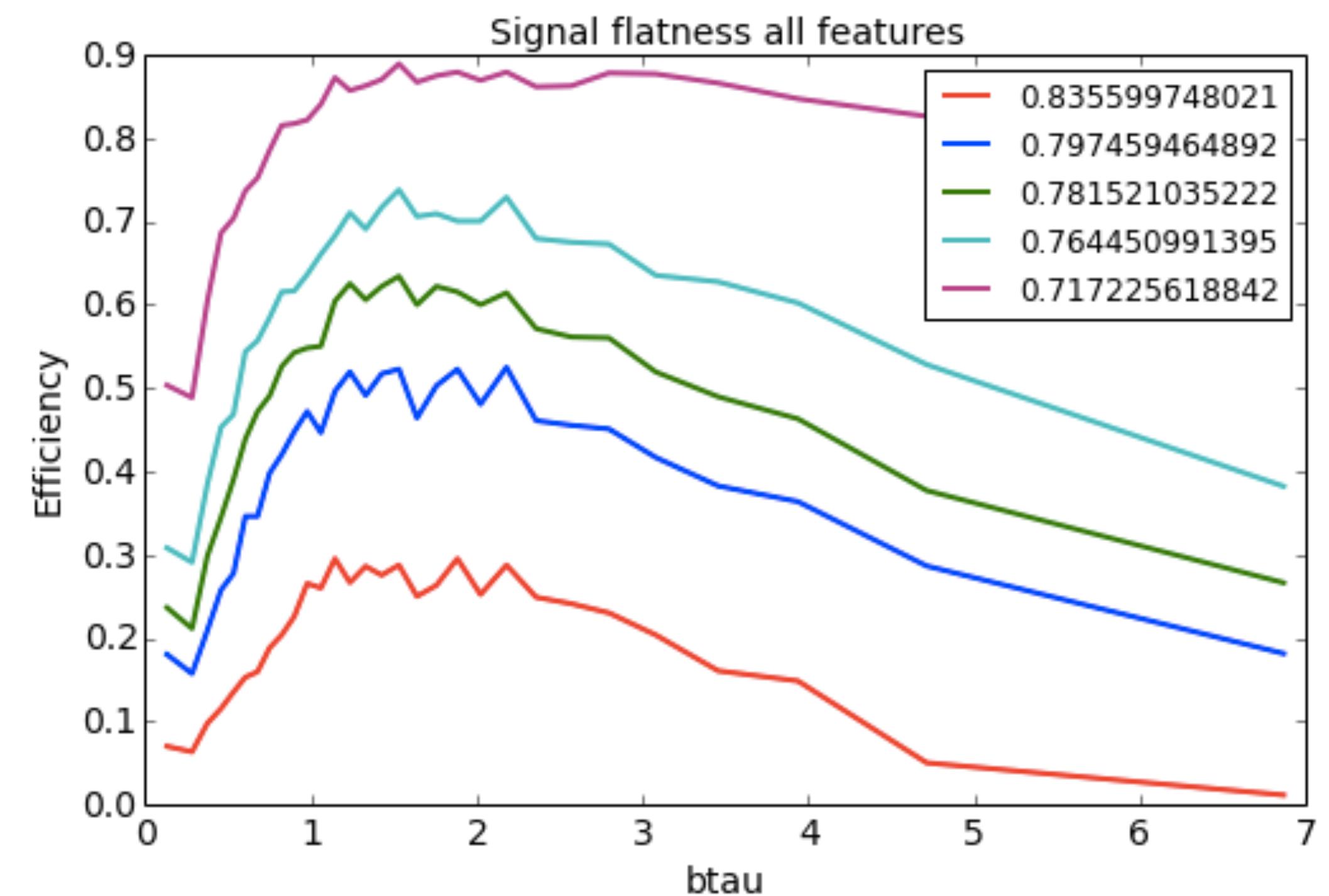
all models use the same set of features for discrimination,  
but AdaBoost got serious dependence on the mass

# PID DEMO



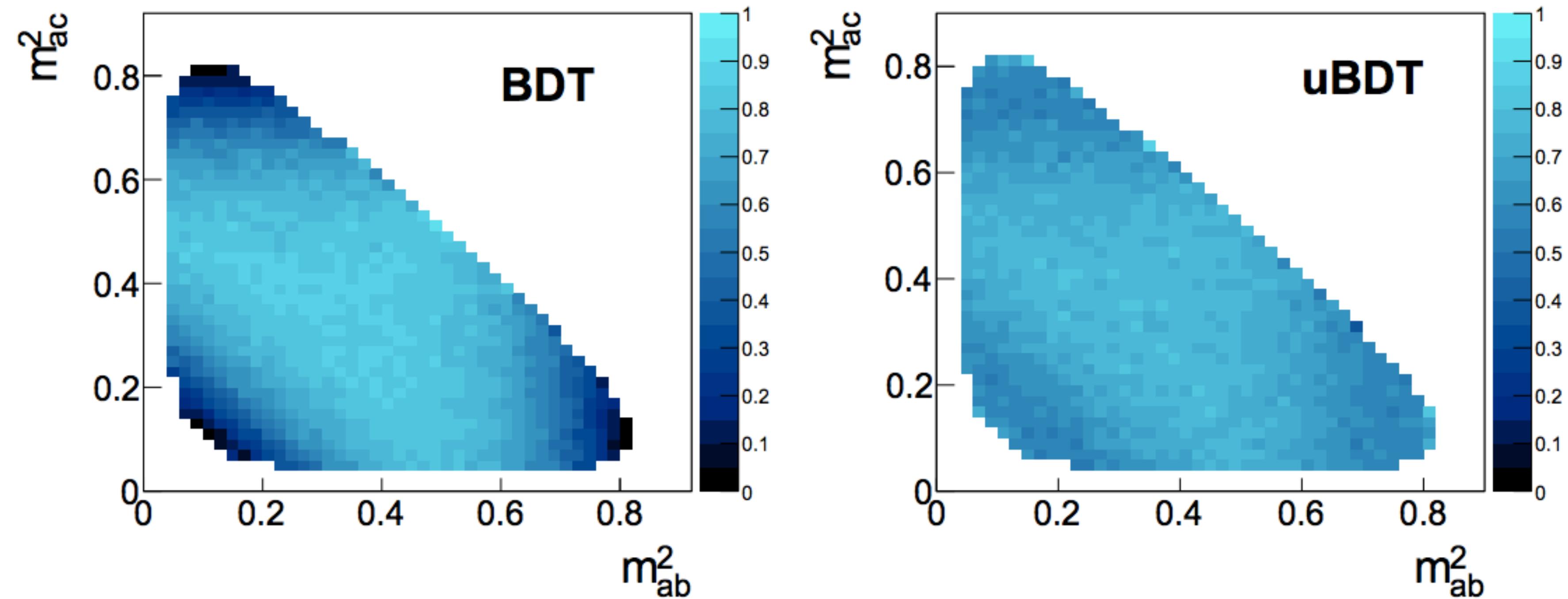
Features strongly depend on the momentum and transverse momentum. Both algorithms use the same set of features. Used MVA is a specific BDT implementation with flatness loss.

# 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

```
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

# Categorical features

# Categorical features



Image: [www.petsworld.in](http://www.petsworld.in)

# One-hot encoding

[proton, pion, kaon]  $\rightarrow$  [[1, 0, 0], [0, 1, 0], [0, 0, 1]]

# One-hot encoding

[proton, pion, kaon]  $\rightarrow$  [[1, 0, 0], [0, 1, 0], [0, 0, 1]]

- › Doesn't scale well with the number of categories

# CTR (aka click-through ratio)

For each pair  
(target\_class, categorical\_feature\_value):

$$\text{ctr}_i = \frac{\text{countInClass} + \text{prior}}{\text{totalCount} + 1}$$

- › countInClass – number of objects in the  $i$ -th class with the current categorical feature value
- › prior – algorithm parameter
- › totalCount – total number of objects with the current categorical feature value

# CTR example

fruit	target	ctr
apple	0	0.625
orange	0	0.25
apple	1	0.625
apple	1	0.625

prior = 0.5

# Classes counter

For each pair

(target\_class, categorical\_feature\_value):

$$\text{count}_i = \frac{\text{curCount} + \text{prior}}{\text{totalCount} + 1}$$

- › curCount – number of objects with the current categorical feature value
- › prior – algorithm parameter
- › totalCount – total number of objects

# Counters example

fruit	target	ctr	counter
apple	0	0.625	0.7
orange	0	0.25	0.3
apple	1	0.625	0.7
apple	1	0.625	0.7

prior = 0.5

# Overfitting

# Gradients bias in gradient boosting

- › Each subsequent tree is fit to the gradient between the current predictions on train and the true labels

# Gradients bias in gradient boosting

- › Each subsequent tree is fit to the gradient between the current predictions on train and the true labels
- › The gradient is estimated using the model fitted on the very dataset used for training

# Gradients bias in gradient boosting

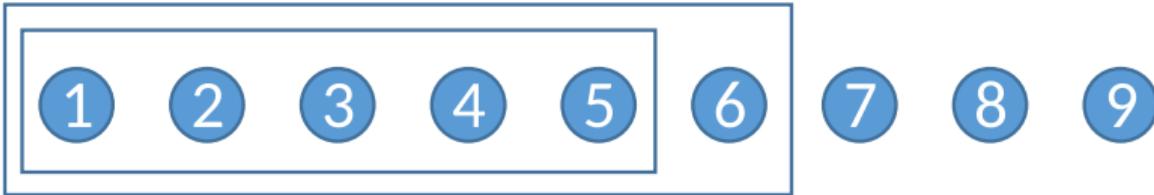
- › Each subsequent tree is fit to the gradient between the current predictions on train and the true labels
- › The gradient is estimated using the model fitted on the very dataset used for training
- › The gradients are likely to be overfitted

# Dynamic boosting



- › Order data randomly

# Dynamic boosting



- › Order data randomly
- › For each element maintain prediction based on the previous model elements

# GB matters

Gradient boosting is the workhorse of Yandex Internet business

# Meet CatBoost



- › Gradient boosting on decision trees
- › Categorical features handling (even more advanced than discussed!)
- › A novel boosting scheme (submitted to NIPS)
- › Released into open source by Yandex on Tuesday
- › Used in the LHCb PID

Bonus track

# Minimum of the expected risk: the proof

› Transform the loss

$$\begin{aligned} Q(y, h(x)) &= (y - h(x))^2 = (y - \mathbb{E}(y | x) + \mathbb{E}(y | x) - h(x))^2 = \\ &= (y - \mathbb{E}(y | x))^2 + 2(y - \mathbb{E}(y | x))(\mathbb{E}(y | x) - h(x)) + \\ &\quad + (\mathbb{E}(y | x) - h(x))^2. \end{aligned}$$

› Write the expected risk

$$\begin{aligned} R(h) &= \mathbb{E}_{x,y} Q(y, h(x)) = \\ &= \mathbb{E}_{x,y} (y - \mathbb{E}(y | x))^2 + \mathbb{E}_{x,y} (\mathbb{E}(y | x) - h(x))^2 + \\ &\quad + 2\mathbb{E}_{x,y} (y - \mathbb{E}(y | x))(\mathbb{E}(y | x) - h(x)). \end{aligned}$$

# Minimum of the expected risk: the proof

- Consider  $\mathbb{E}_{x,y}(y - \mathbb{E}(y | x))(\mathbb{E}(y | x) - h(x))$  which is essentially some

$$\mathbb{E}_x \mathbb{E}_y [f(x, y) | x] = \int_{\mathbb{X}} \left( \int_{\mathbb{Y}} f(x, y) p(y | x) dy \right) p(x) dx$$

meaning that (as  $(\mathbb{E}(y | x) - h(x))$  is independent of  $y$ ):

$$\begin{aligned} & \mathbb{E}_x \mathbb{E}_y \left[ (y - \mathbb{E}(y | x)) (\mathbb{E}(y | x) - h(x)) | x \right] = \\ &= \mathbb{E}_x \left( (\mathbb{E}(y | x) - h(x)) \mathbb{E}_y \left[ (y - \mathbb{E}(y | x)) | x \right] \right) = \\ &= \mathbb{E}_x \left( (\mathbb{E}(y | x) - h(x)) (\mathbb{E}(y | x) - \mathbb{E}(y | x)) \right) = \\ &= 0 \end{aligned}$$

# Minimum of the expected risk: the proof

- › We obtain that the expected risk has the form

$$R(h) = \mathbb{E}_{x,y}(y - \mathbb{E}(y | x))^2 + \mathbb{E}_{x,y}(\mathbb{E}(y | x) - h(x))^2.$$

- › Both summands are nonnegative meaning that the sum is minimized when

$$h_*(x) = \mathbb{E}(y | x) = \int_{\mathbb{Y}} y p(y | x) dy.$$

# Bias-variance decomposition

- › Input: the training set  $X^\ell = \{(x_i, y_i)\}_{i=1}^\ell$
- › Learning method  $\mu : (\mathbb{X} \times \mathbb{Y})^\ell \rightarrow \mathbb{H}$
- › Can evaluate quality using average (over possible samples  $X^\ell$ )

$$\begin{aligned} Q(\mu) &= \mathbb{E}_{X^\ell} \left[ \mathbb{E}_{x,y} \left[ (y - \mu(X^\ell)(x))^2 \right] \right] = \\ &= \int_{(\mathbb{X} \times \mathbb{Y})^\ell} \int_{\mathbb{X} \times \mathbb{Y}} (y - \mu(X^\ell)(x))^2 p(x, y) \prod_{i=1}^{\ell} p(x_i, y_i) dx dy dx_i dy_i \end{aligned}$$

- › For a fixed sample  $X^\ell$ , we know the expected risk

$$\mathbb{E}_{x,y} \left[ (y - \mu(X^\ell))^2 \right] = \mathbb{E}_{x,y} \left[ (y - \mathbb{E}[y | x])^2 \right] + \mathbb{E}_{x,y} \left[ (\mathbb{E}[y | x] - \mu(X^\ell))^2 \right]$$

# Bias-variance decomposition

- › Plugging the expression for fixed  $X^\ell$  into  $Q(\mu)$ , we obtain

$$Q(\mu) = \mathbb{E}_{X^\ell} \left[ \underbrace{\mathbb{E}_{x,y} \left[ (y - \mathbb{E}[y | x])^2 \right]}_{\text{independent of } X^\ell} + \mathbb{E}_{x,y} \left[ (\mathbb{E}[y | x] - \mu(X^\ell))^2 \right] \right]$$

- › Transforming the second summand

$$\begin{aligned} & \mathbb{E}_{x,y} \left[ \mathbb{E}_{X^\ell} \left[ (\mathbb{E}[y | x] - \mu(X^\ell))^2 \right] \right] = \\ &= \mathbb{E}_{x,y} \left[ \mathbb{E}_{X^\ell} \left[ (\mathbb{E}[y | x] - \mathbb{E}_{X^\ell} [\mu(X^\ell)] + \mathbb{E}_{X^\ell} [\mu(X^\ell)] - \mu(X^\ell))^2 \right] \right] = \\ &= \mathbb{E}_{x,y} \left[ \mathbb{E}_{X^\ell} \left[ \underbrace{(\mathbb{E}[y | x] - \mathbb{E}_{X^\ell} [\mu(X^\ell)])^2}_{\text{independent of } X^\ell} \right] \right] + \mathbb{E}_{x,y} \left[ \mathbb{E}_{X^\ell} \left[ (\mathbb{E}_{X^\ell} [\mu(X^\ell)] - \mu(X^\ell)) \right] \right. \\ &\quad \left. + 2\mathbb{E}_{x,y} \left[ \mathbb{E}_{X^\ell} \left[ (\mathbb{E}[y | x] - \mathbb{E}_{X^\ell} [\mu(X^\ell)]) (\mathbb{E}_{X^\ell} [\mu(X^\ell)] - \mu(X^\ell)) \right] \right] \right] \end{aligned}$$

# Bias-variance decomposition

› We prove that the last term is zero:

$$\begin{aligned}\mathbb{E}_{X^\ell} \left[ (\mathbb{E}[y | x] - \mathbb{E}_{X^\ell} [\mu(X^\ell)]) (\mathbb{E}_{X^\ell} [\mu(X^\ell)] - \mu(X^\ell)) \right] &= \\ = (\mathbb{E}[y | x] - \mathbb{E}_{X^\ell} [\mu(X^\ell)]) \mathbb{E}_X \left[ \mathbb{E}_{X^\ell} [\mu(X^\ell)] - \mu(X^\ell) \right] &= \\ = (\mathbb{E}[y | x] - \mathbb{E}_{X^\ell} [\mu(X^\ell)]) \left[ \mathbb{E}_{X^\ell} [\mu(X^\ell)] - \mathbb{E}_{X^\ell} [\mu(X^\ell)] \right] &= \\ &= 0.\end{aligned}$$