Yandex

SCHOOL OF DATA ANALYSIS

Skoltech
Skolkovo Institute of Science and Technology

# 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[1,2]

[1] Skoltech [2] National Research University Higher School of Economics

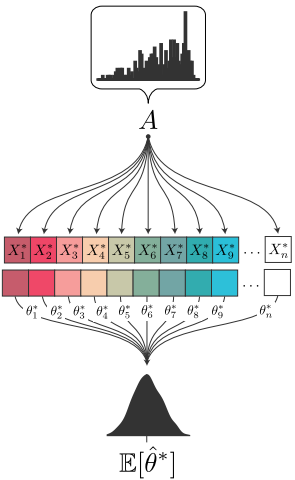
# Lecture overview

- 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 $X_1^m, \ldots, X_N^m$
  2. Learn $N$ hypotheses $h_1, \ldots, h_N$
  3. Average predictions to obtain $h(x) = \frac{1}{N} \sum_{i=1}^N h_i(x)$
  4. Profit!



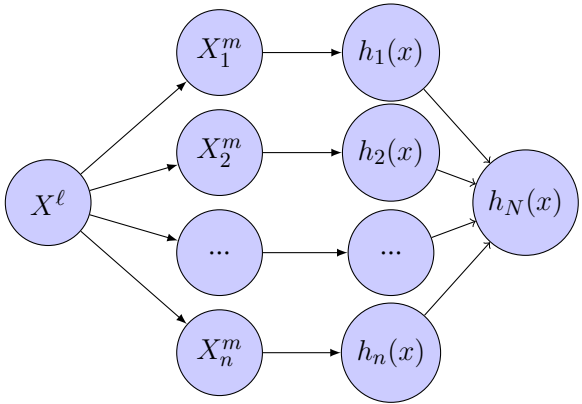

Picture credit: http://www.drbunsen.org/bootstrap-in-picture

# 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

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

$$= \frac{1}{N}\sum_{i=1}^{N} \tilde{\mu}(X^\ell)(x)$$



# The Random Forest algorithm

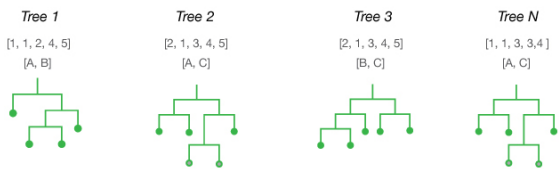
- Bagging over (weak) decision trees
- Reduce error via averaging over instances and features
- Input: a sample $X^\ell = \{(\mathbf{x}_i, y_i)\}_{i=1}^\ell$, where $\mathbf{x}_i \in \mathbb{R}^d, y_i \in \mathbb{Y}$
- The algorithm iterates for $i = 1, \ldots, N$:
  1. Pick $p$ random features out of $d$
  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 that $n_{\text{min}}$ instances

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

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

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

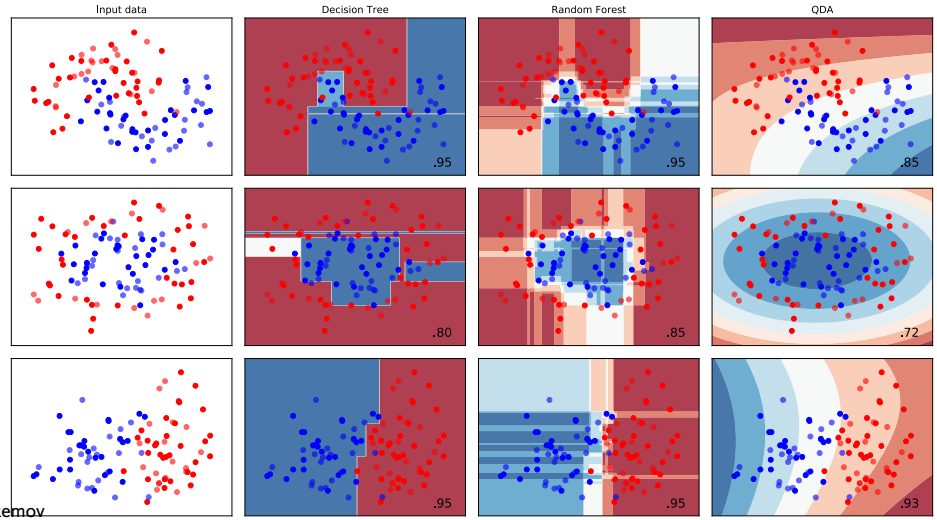
Learn Tree$_i(\mathbf{x})$ using $X_i^m$






Picture credit: http://www.thefactmachine.com/random-forests

# 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 $Q(y, h) = \big(y - h(x)\big)^2$
- Expected risk: average (over $p(x, y)$) squared loss when using $h$

$$R(h) = \mathbb{E}_{x,y}\Big[\big(y - h(x)\big)^2\Big] = \int_{\mathbb{X}}\int_{\mathbb{Y}} p(x, y)\big(y - a(x)\big)^2 dxdy.$$

- A statement: the expected risk is minimized by

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

# Bias-variance decomposition

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

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

$$= \int_{(\mathbb{X}\times\mathbb{Y})^\ell} \int_{\mathbb{X}\times\mathbb{Y}} \left(y - \mu(X^\ell)(x)\right)^2 p(x,y) \prod_{i=1}^{\ell} p(x_i, y_i) dx dy dx_i dy_i$$

# Bias-variance decomposition

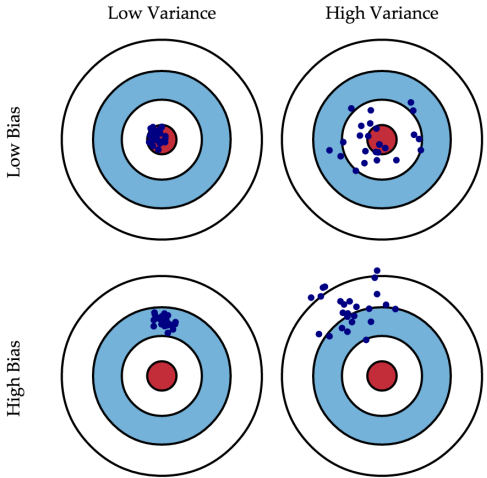
- We arrive at the famous **bias-variance(-noise) decomposition**

$$Q(\mu) = \underbrace{\mathbb{E}_{x,y}\Big[\big(y - \mathbb{E}[y \mid x]\big)^2\Big]}_{\text{noise}} +$$

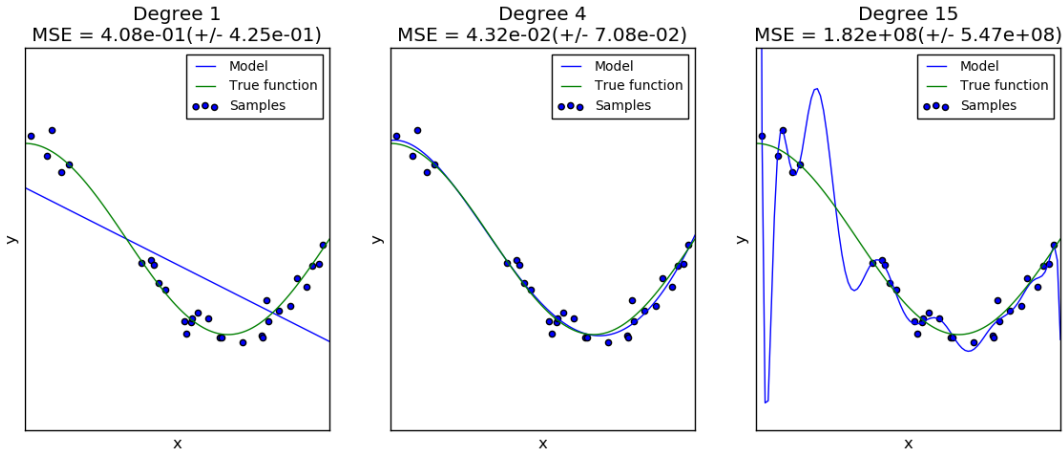
$$+ \underbrace{\mathbb{E}_x\Big[\big(\mathbb{E}_{X^\ell}\big[\mu(X^\ell)\big] - \mathbb{E}[y \mid x]\big)^2\Big]}_{\text{bias}} + \underbrace{\mathbb{E}_x\Big[\mathbb{E}_{X^\ell}\Big[\big(\mu(X^\ell) - \mathbb{E}_{X^\ell}\big[\mu(X^\ell)\big]\big)^2\Big]\Big]}_{\text{variance}}$$

- 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

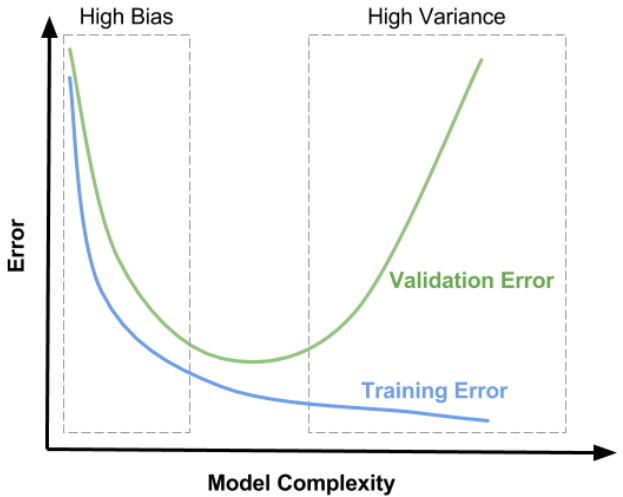
# Bias-variance decomposition



# Polynomial fits of different degrees



# Bias-variance tradeoff






Picture credit: http://www.learnopencv.com/bias-variance-tradeoff-in-machine-learning

# An application: statistical analysis of Bagging

› Bias: not made any worse by bagging multiple hypotheses

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

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

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

# An application: statistical analysis of Bagging

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

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

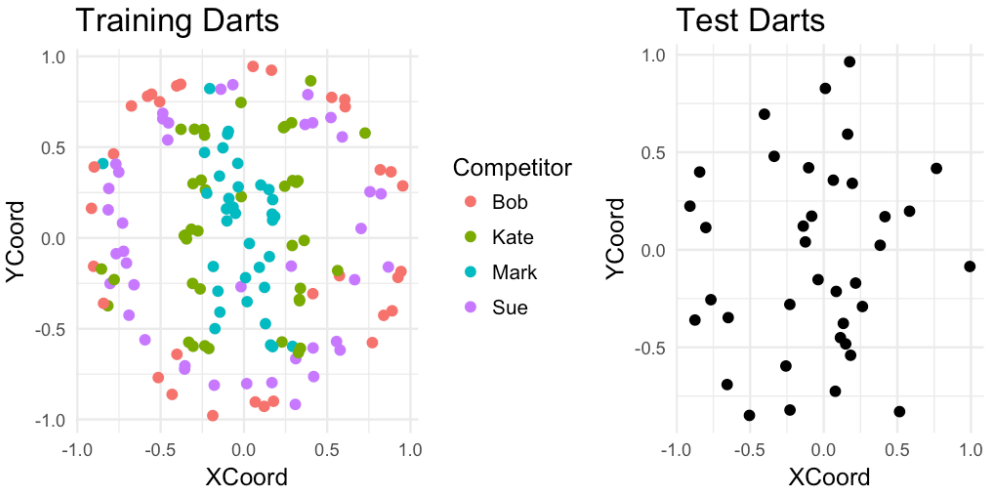
$$= \frac{1}{N}\mathbb{E}_{x,y}\left[\mathbb{E}_{X^\ell}\left[\left(\tilde{\mu}(X^\ell)(x) - \mathbb{E}_{X^\ell}\left[\tilde{\mu}(X)(x)\right]\right)^2\right]\right] +$$

$$+ \frac{N(N-1)}{N^2}\mathbb{E}_{x,y}\left[\mathbb{E}_{X^\ell}\left[\left(\tilde{\mu}(X^\ell)(x) - \mathbb{E}_{X^\ell}\left[\tilde{\mu}(X^\ell)(x)\right]\right)\times\right.\right.$$

$$\left.\left.\times\left(\tilde{\mu}(X^\ell)(x) - \mathbb{E}_{X^\ell}\left[\tilde{\mu}(X^\ell)(x)\right]\right)\right]\right]$$

# Stacked generalization

# Stacking motivation: the game of Darts





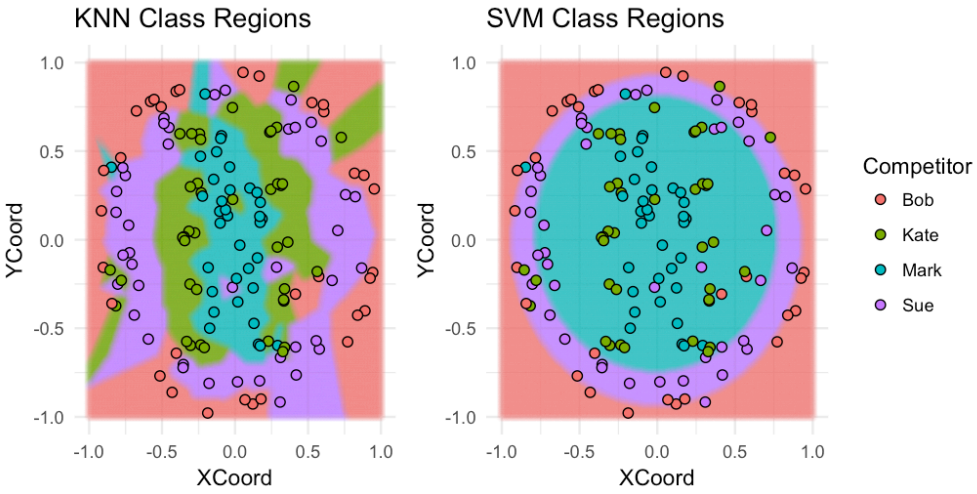

Picture credit: http://blog.kaggle.com/2016/12/27/a-kagglers-guide-to-model-stacking-in-practice/

# 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






Picture credit: http://blog.kaggle.com/2016/12/27/a-kagglers-guide-to-model-stacking-in-practice/

# 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 $\mathrm{Fold}_i \in \{\mathrm{Fold}_1, \ldots, \mathrm{Fold}_5\}$
   - 3.1 Combine the other 4 folds for training $\rightarrow \mathrm{Fold}_{-i}$
   - 3.2 Fit each base model to $\mathrm{Fold}_{-i}$, predict on $\mathrm{Fold}_i$, save prdections 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




Picture credit: http://blog.kaggle.com/2016/12/27/a-kagglers-guide-to-model-stacking-in-practice

# An intermediate conclusion

- Bootstrapping: a general statistical technique for computing sample functionals (and their variance)
- Bagging: meta-learner over arbitrary weak algorithms via **b**ootstrap **agg**regation
- 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 $\frac{1}{2}\sum_{i=1}^{\ell}(h(x_i) - y_i)^2 \to \min\limits_{h}$
- 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

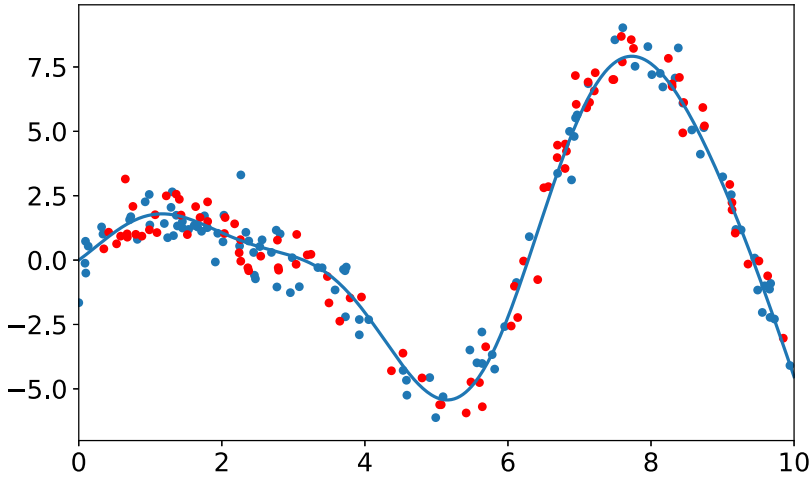
$$s_i^{(N)} = y_i - \sum_{n=1}^{N-1} h_n(x_i) = y_i - a_{N-1}(x_i), \qquad i = 1, \ldots, \ell$$

  3. Learn the next weak algorithm using

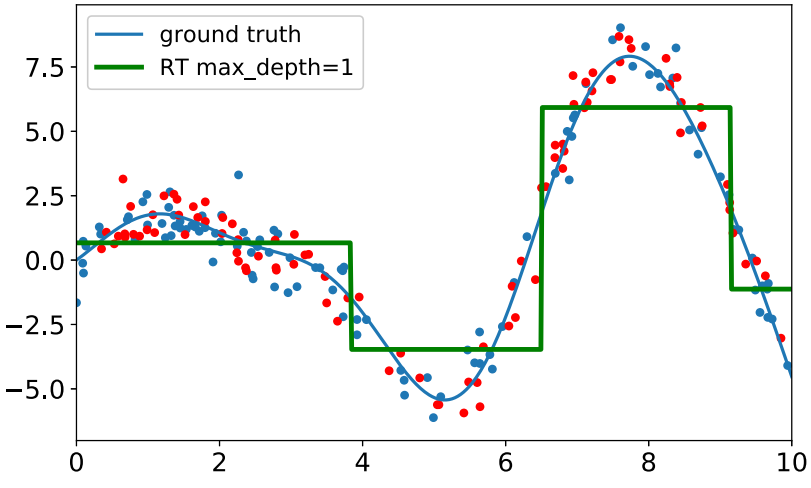
$$a_N(x) := \arg\min_{h \in \mathbb{H}} \frac{1}{2}\sum_{i=1}^{\ell}(h(x_i) - s_i^{(N)})^2$$

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

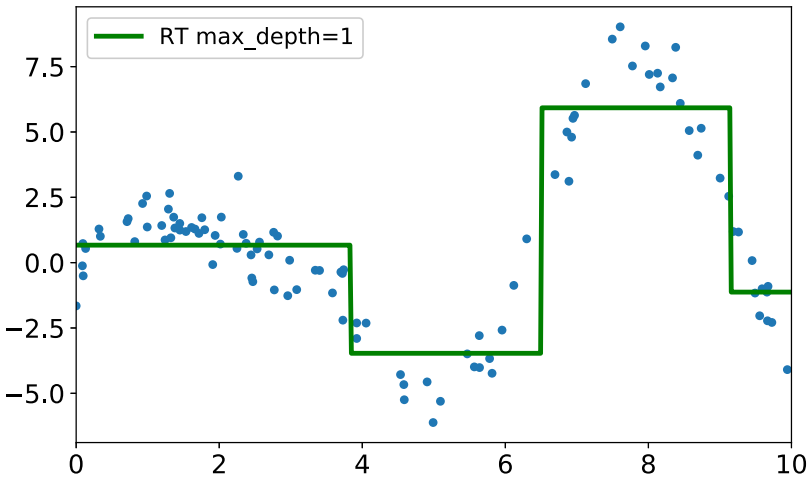
# Boosting: an example regression problem



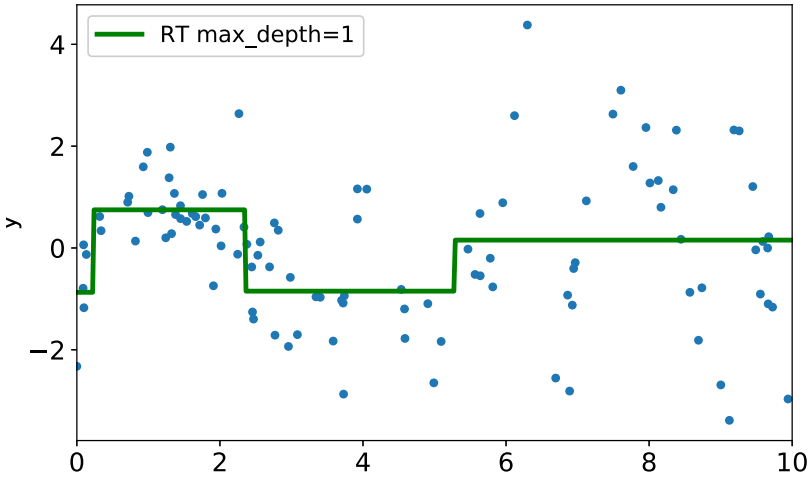
# Boosting: an example regression problem



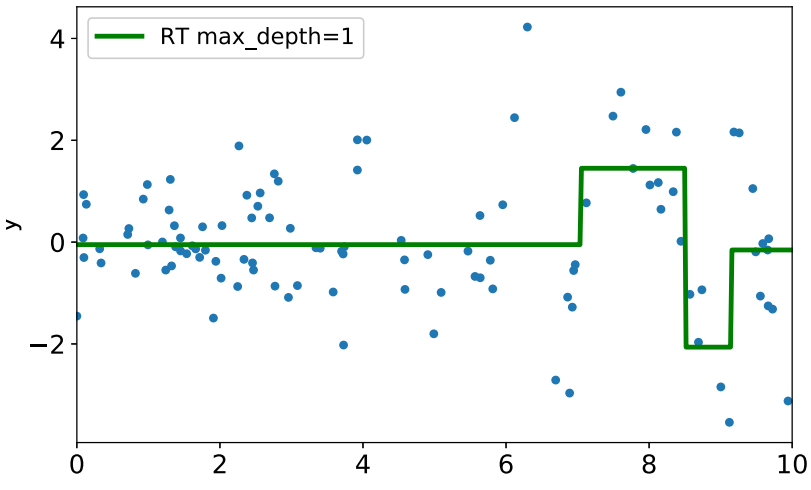
# Boosting: an example regression problem



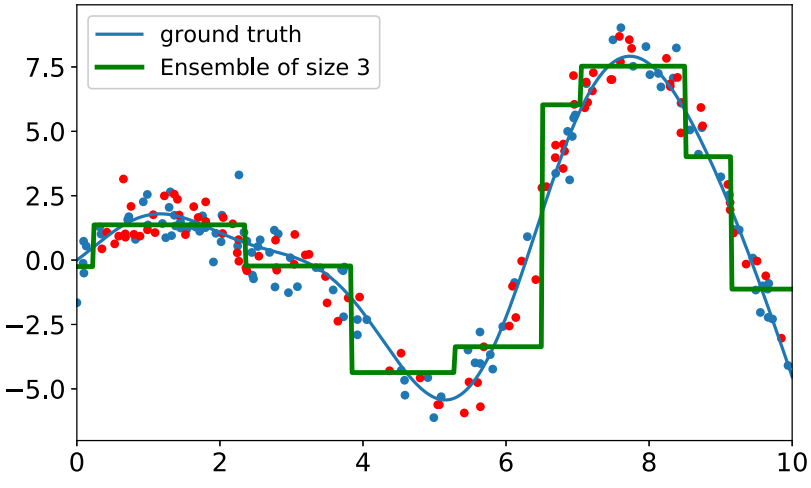
# Boosting: an example regression problem



# Boosting: an example regression problem



# Boosting: an example regression problem

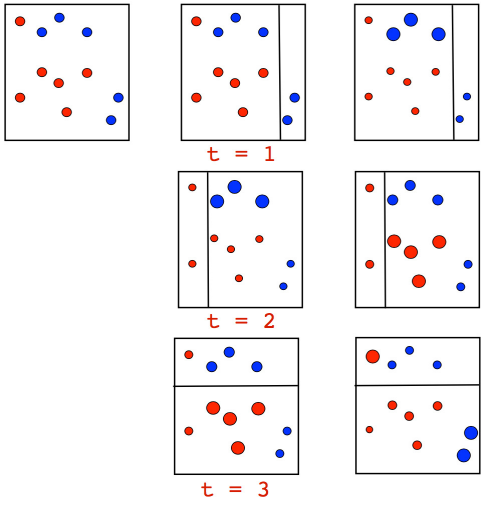


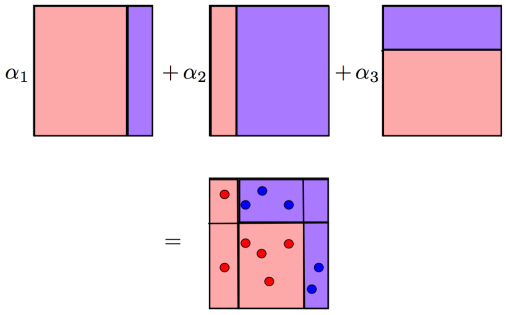
# Reweighting and AdaBoost

# 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






Picture credit: https://habrahabr.ru/company/ods/blog/327250

# Adaptive boosting for classification

[Video: AdaBoost in Action]
https://www.youtube.com/watch?v=k4G2VCuOMMg

# Gradient boosting

# Gradient boosting: motivation

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

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

- 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 $s_1, \ldots, s_\ell$ that

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

- 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 $t = 1, \ldots, N$ (see next slide)
- Compose the final GBM learner: $a_N(\mathbf{x}) = \sum_{t=0}^N \gamma_i h_i(\mathbf{x})$

# The Gradient Boosting Machine [Friedman, 2001]

At every iteration:

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

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

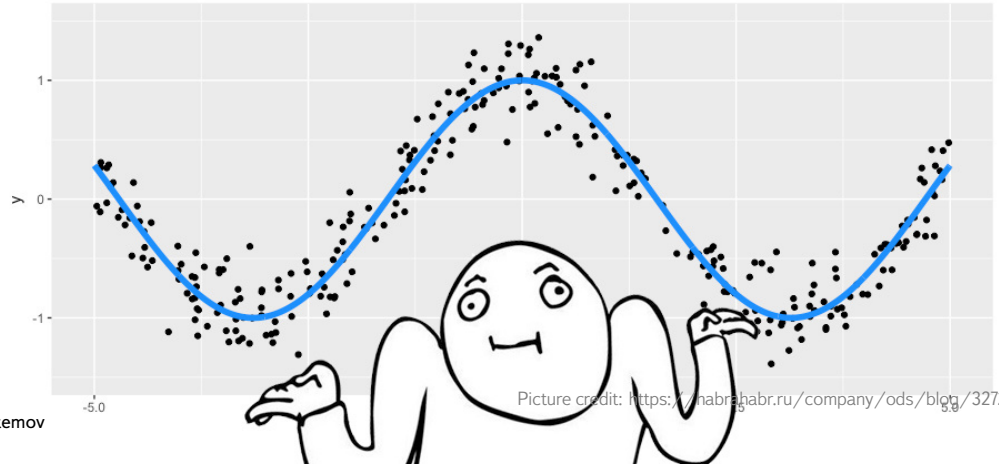
3. Find the optimal $\gamma_N$ using plain gradient descent:

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

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

# GBM: an example regression problem

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

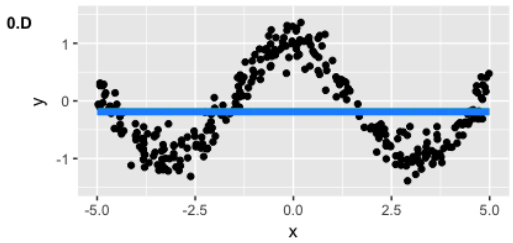


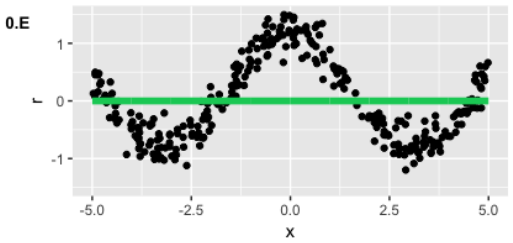

Picture credit: https://habrahabr.ru/company/ods/blog/327250

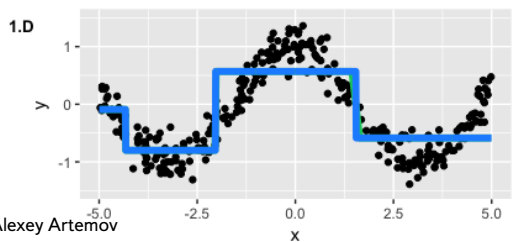
# GBM: an example regression problem

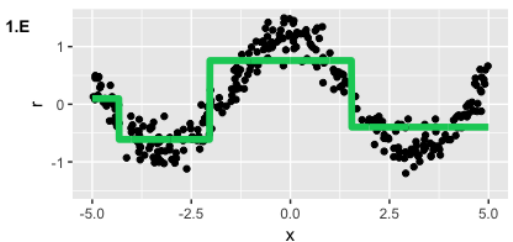
- Pick $N = 3$ boosting iterations
- Quadratic loss $Q(y, z) = (y - z)^2$
- Gradient of the quadratic loss $\frac{\partial Q(y_i, z)}{\partial z} = (y - z)$ is just redisuals
- Pick decision trees as weak learners $h_i(\mathbf{x})$
- Set 2 as the maximum depth for decision trees

# GBM: an example regression problem

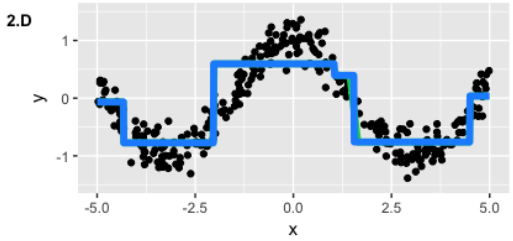


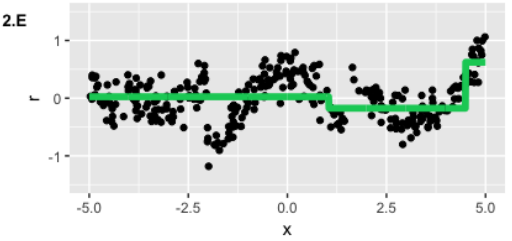



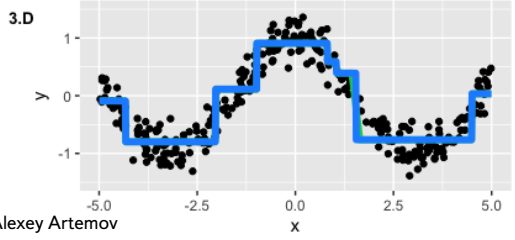



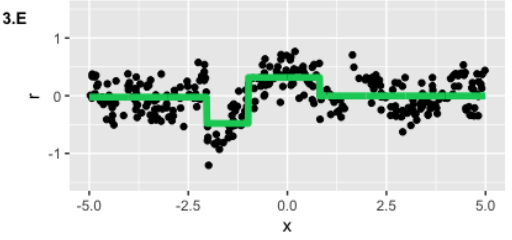



# GBM: an example regression problem



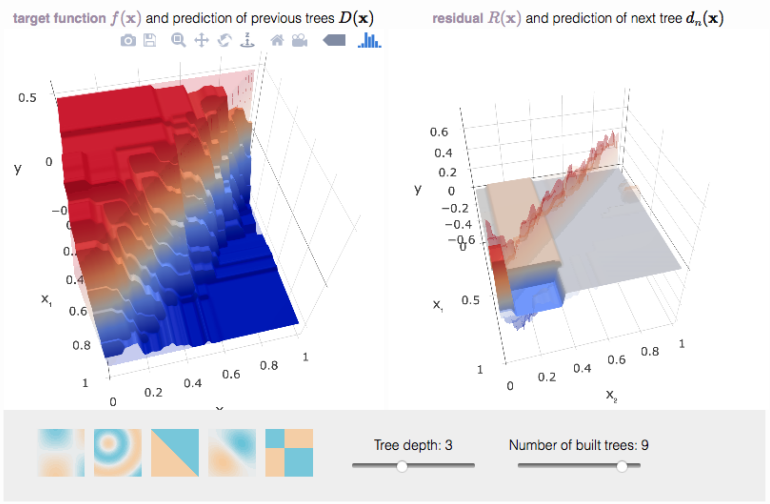



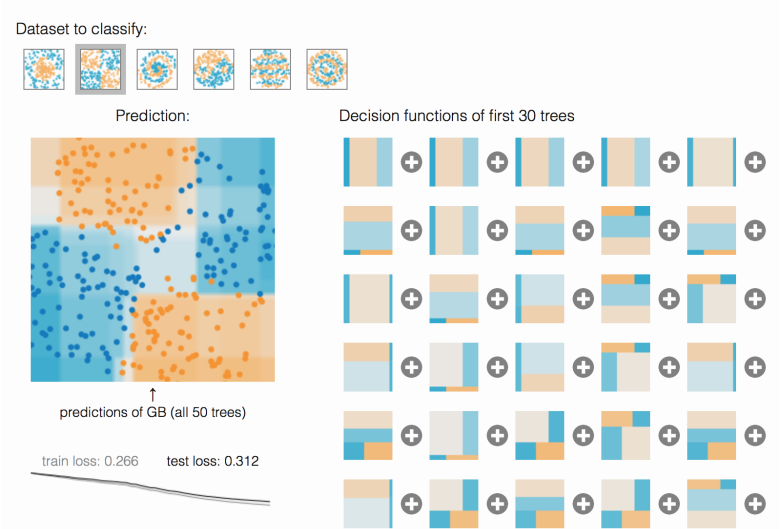
# GBM: an interactive demo

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



# GBM: regulization via shrinkage

- For too simple weak learners, the negative gradient is approximated badly $\Longrightarrow$ 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 $\eta \in (0, 1]$

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

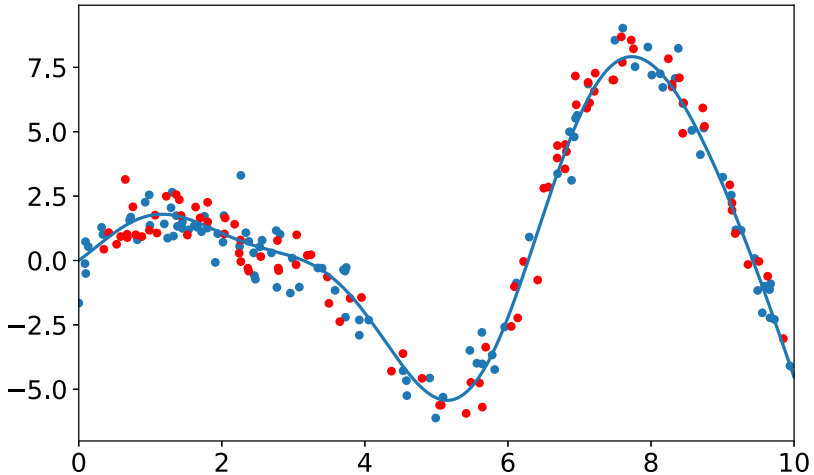
(effectively distrust gradient direction estimated via weak learners)

# GBM: shrinkage animated

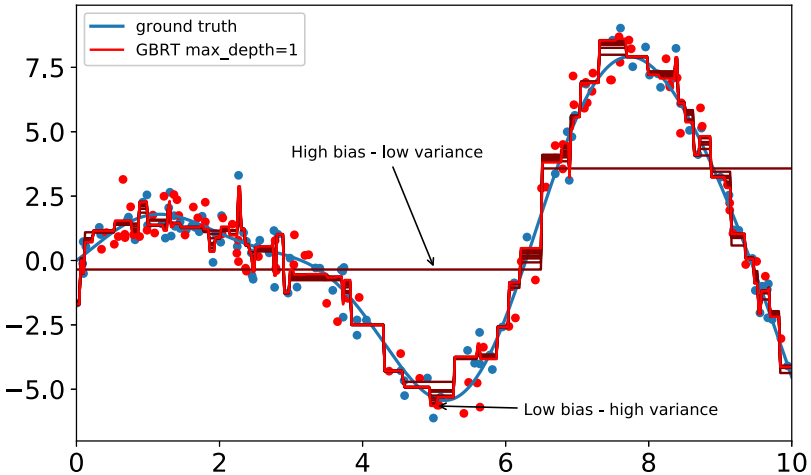
Figure: High shrinkage

Figure: Low shrinkage

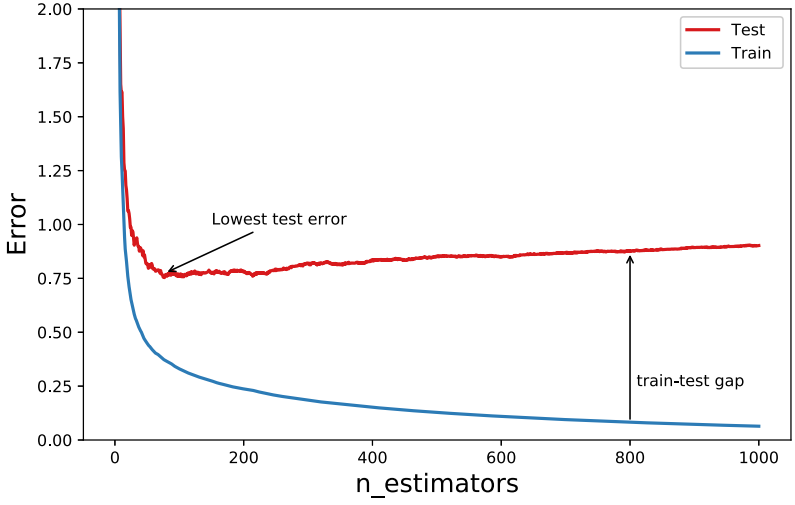
# GBM: regularization approaches



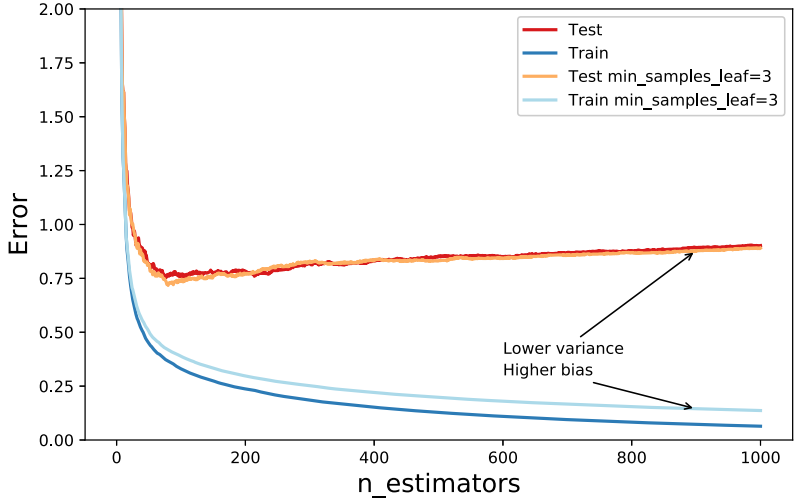
# GBM: regularization approaches



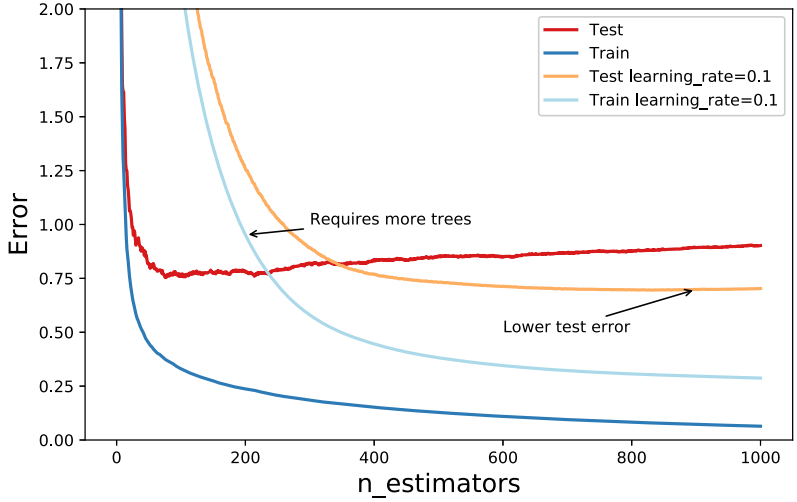
# GBM: regularization approaches



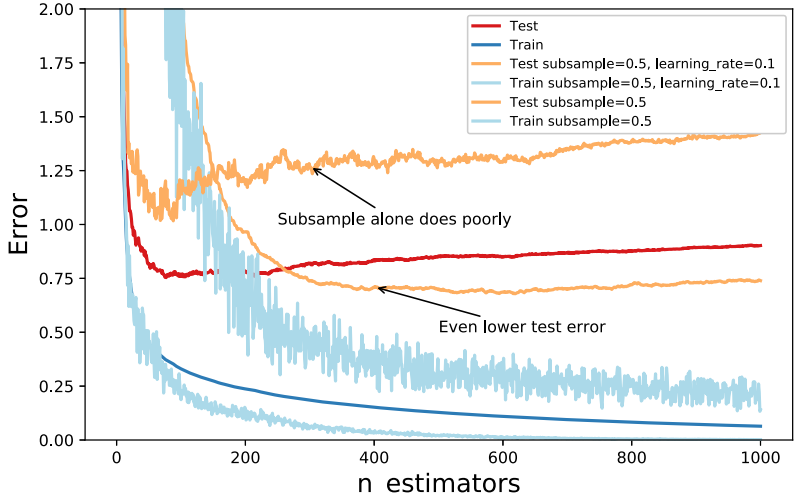
# GBM: regularization approaches



# GBM: regularization approaches



# GBM: regularization approaches



# XGBoost algorithm

# Extreme Gradient Boosting

1. Approximate the descent direction constructed using second order derivatives

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

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

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

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

# Extreme Gradient Boosting

3. Choose split $[\mathbf{x}_j < t]$ at node $R$ to maximize

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

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

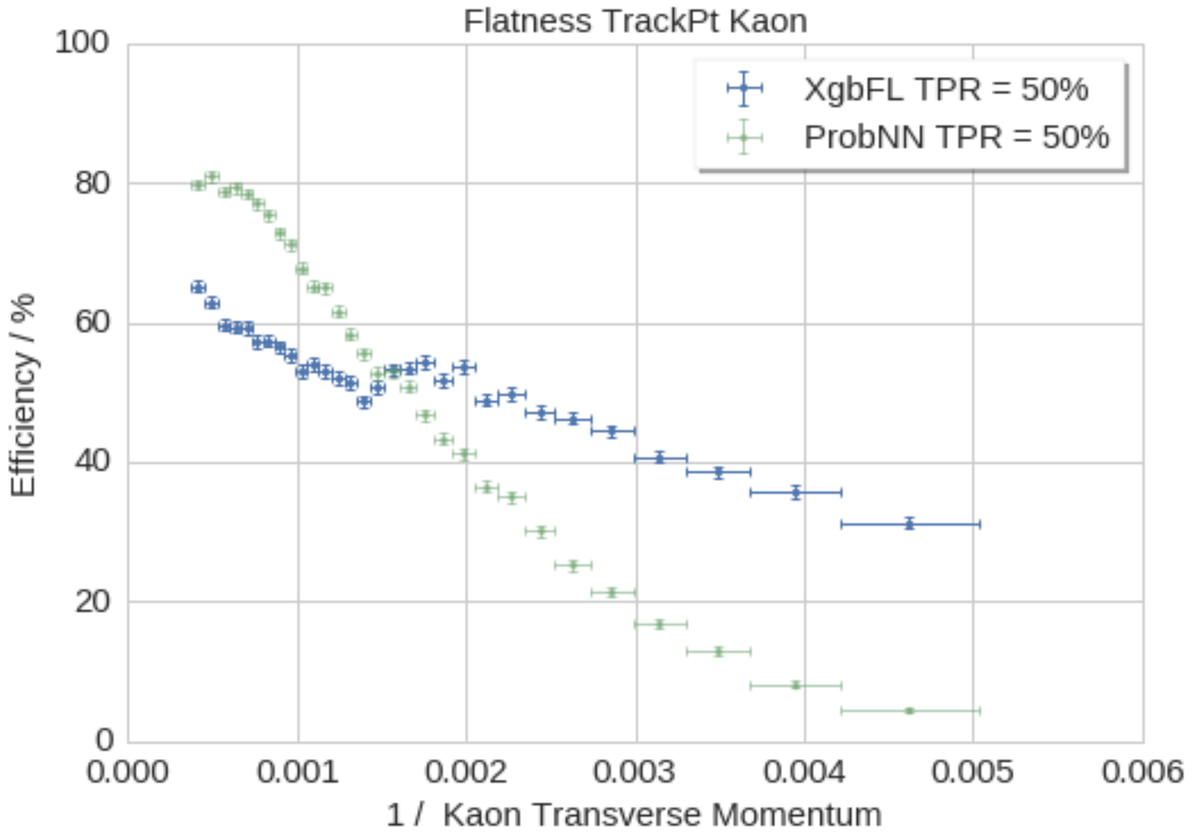
# Boosting to uniformity

# 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

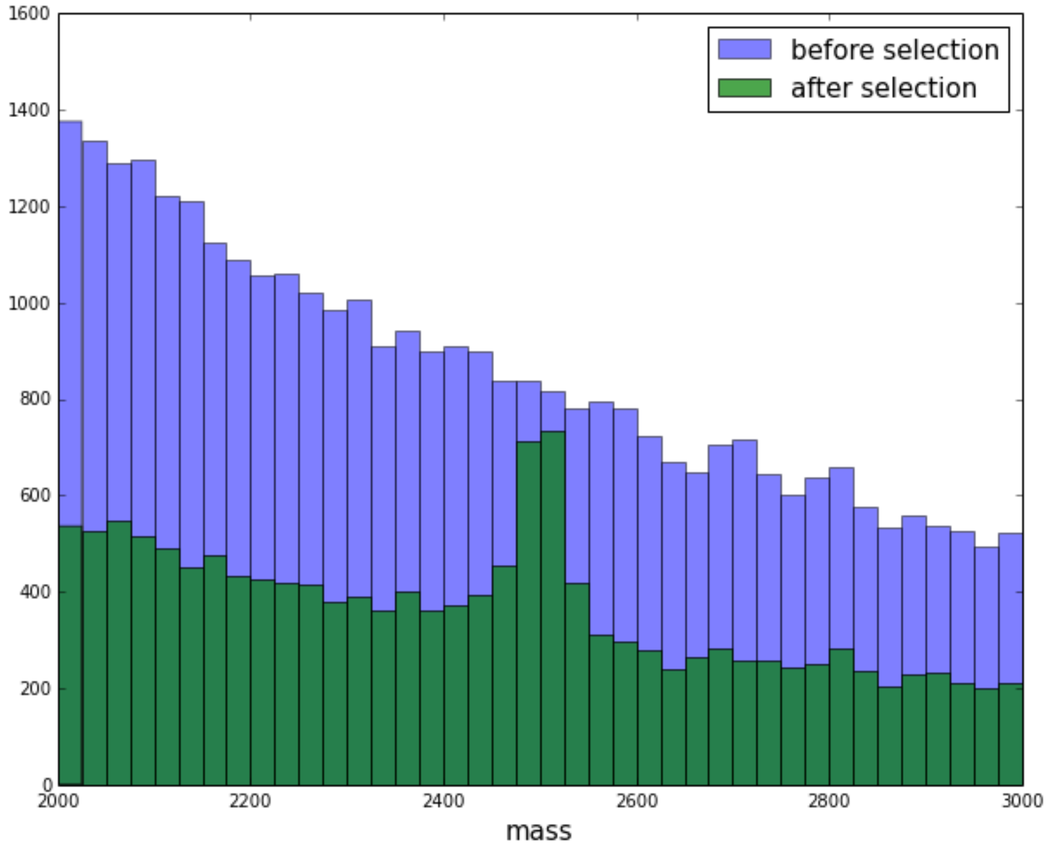


# Non-flatness along the mass

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

Goal: $FPR = const$ 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

# uBoostBDT

- Aims to get $\text{FPR}_{\text{region}}=const$
- 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

$$w_i \leftarrow w_i \exp\left(\beta(\mathrm{FPR}_{\mathrm{region}} - \mathrm{FPR}_{\mathrm{target}})\right)$$

- This way we achieve $\text{FPR}_{\text{region}}=30\%$ in all regions only for some threshold on training dataset

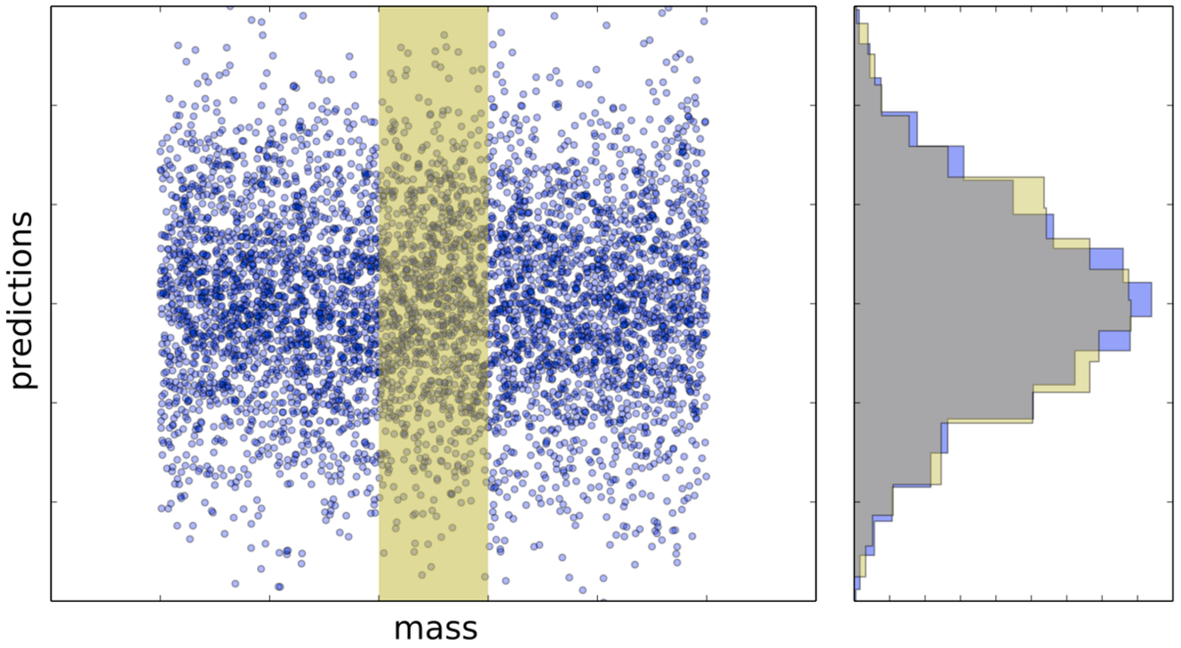
# uBoost

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

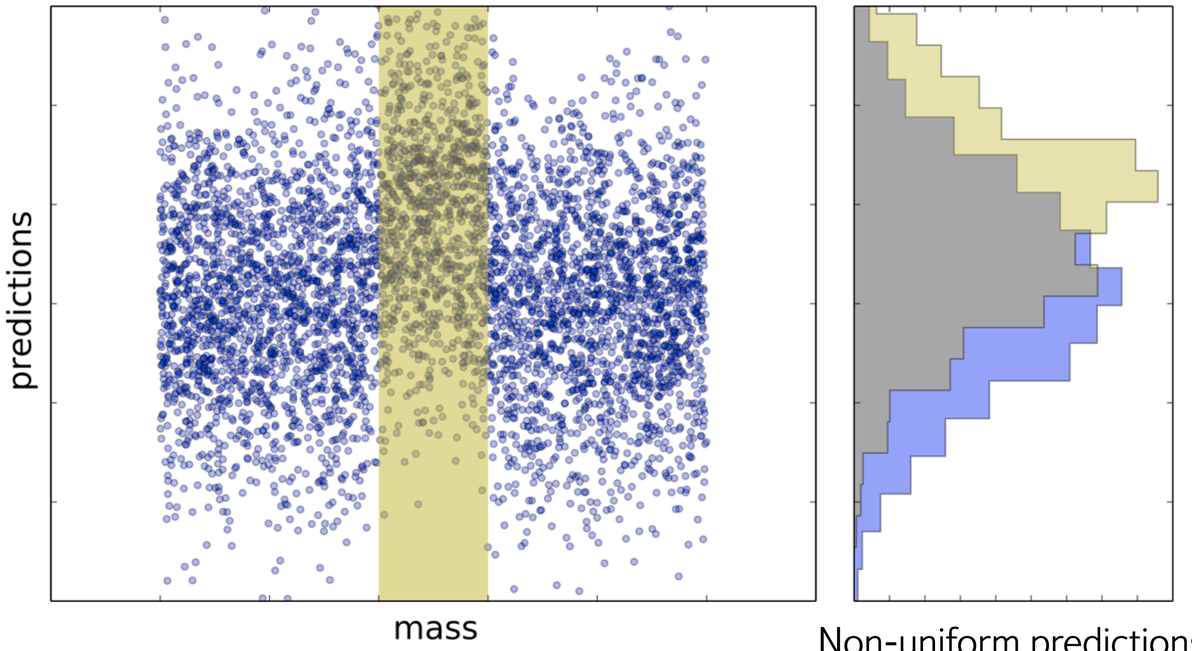
uBoostBDT returns 0 or 1 (passed or not the threshold corresponding to $FPR_{target}$),
simple averaging is used to obtain predictions.

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

# Non-uniformity measure




Uniform predictions



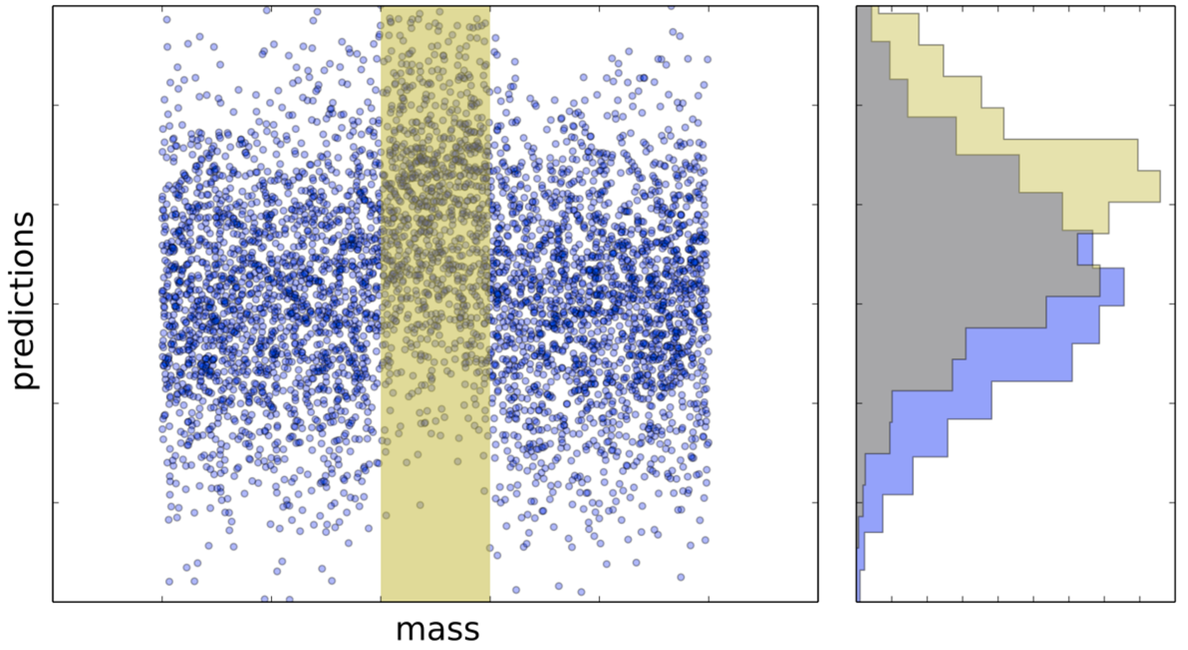

Non-uniform predictions (peak in highlighted region)

- 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)

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



# 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:

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

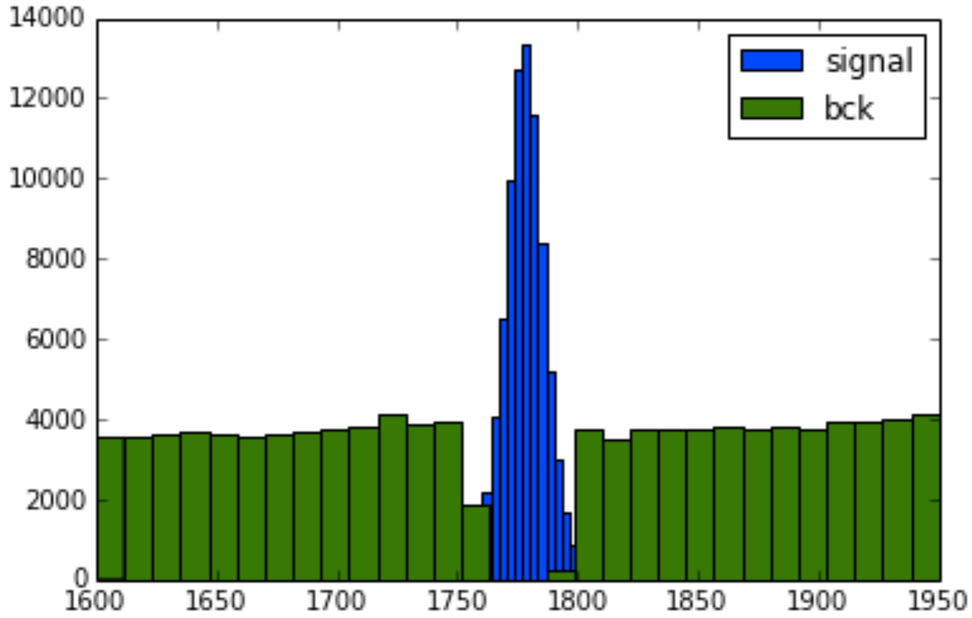
- Flatness loss approximates non-differentiable CvM measure:

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

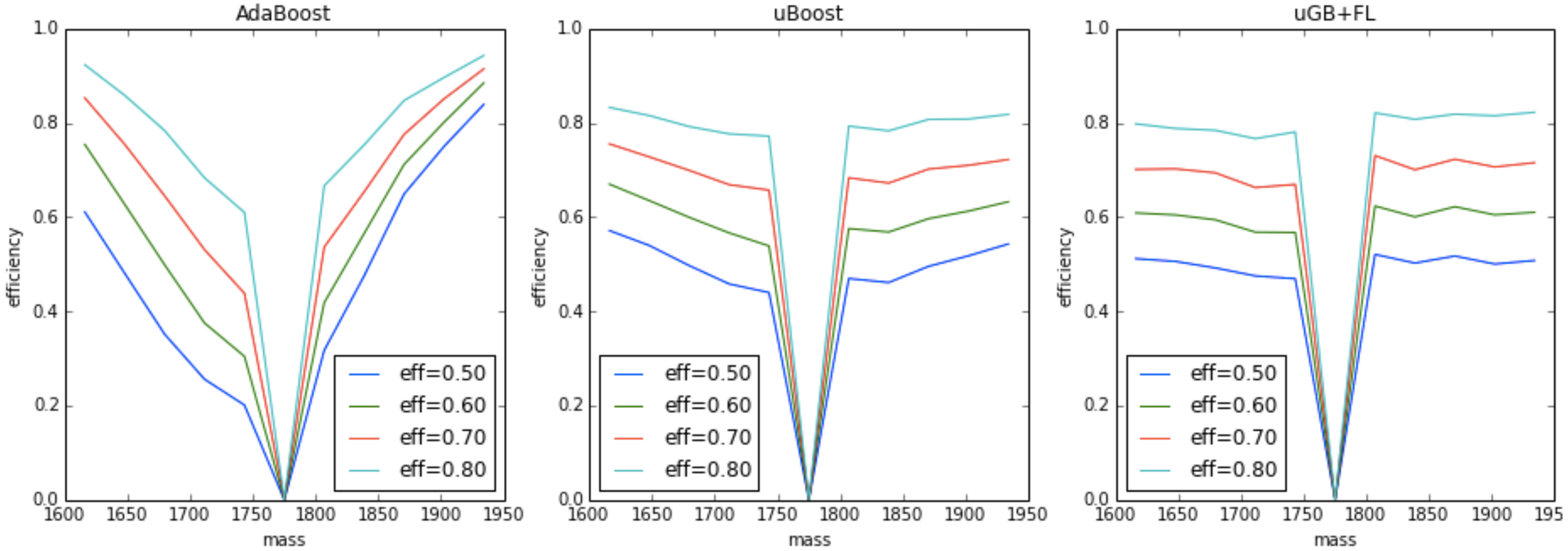
$$\frac{\partial}{\partial D(x_i)} \mathcal{L}_{\text{FL}} \sim 2(F_{\text{region}}(s) - F_{\text{global}}(s))\Big|_{s=D(x_i)}$$

# 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)

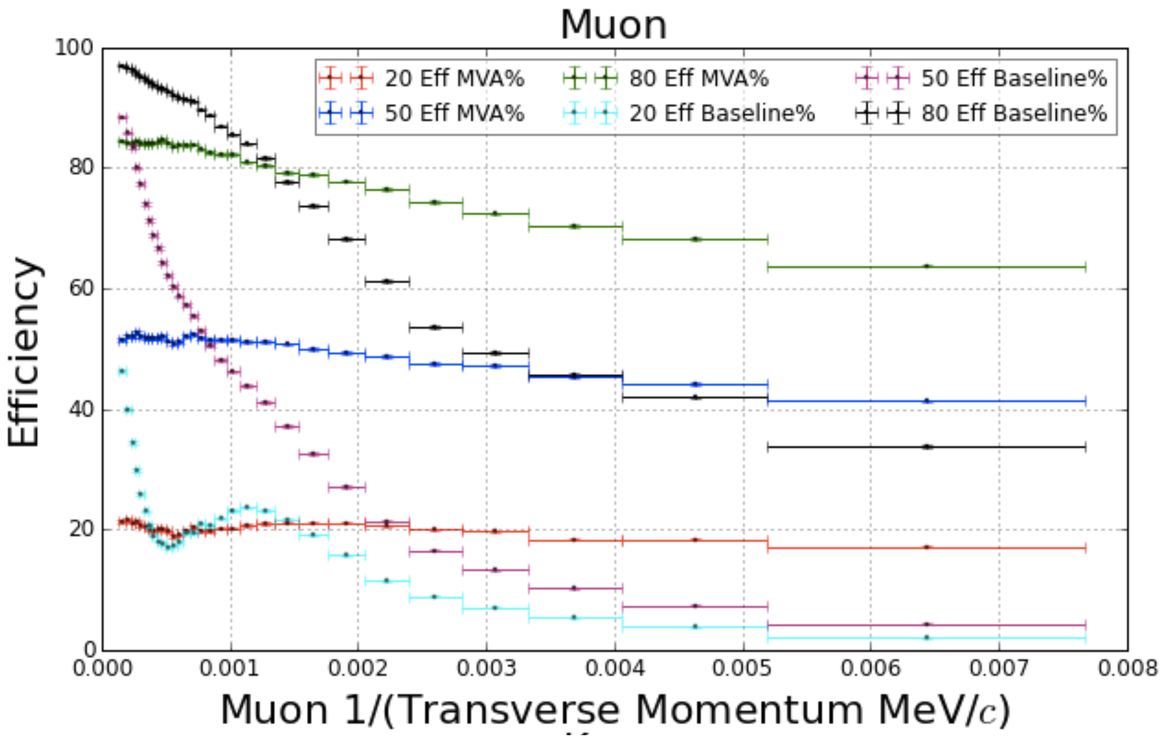


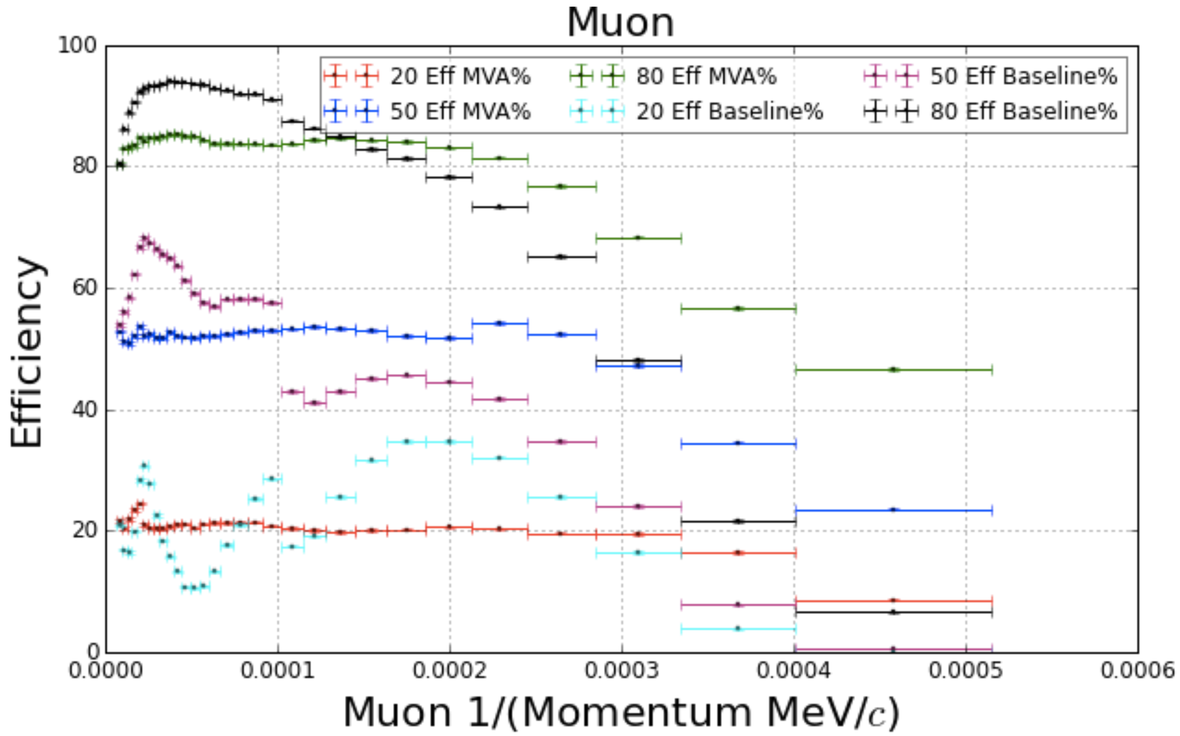
# Rare decay analysis DEMO




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

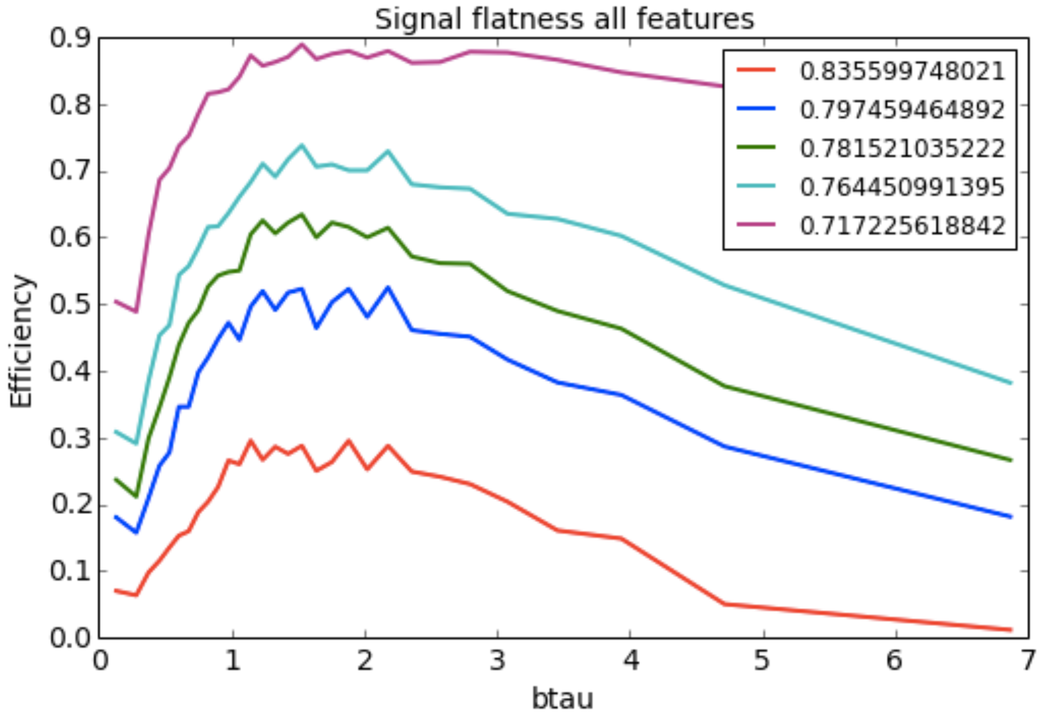
# PID DEMO

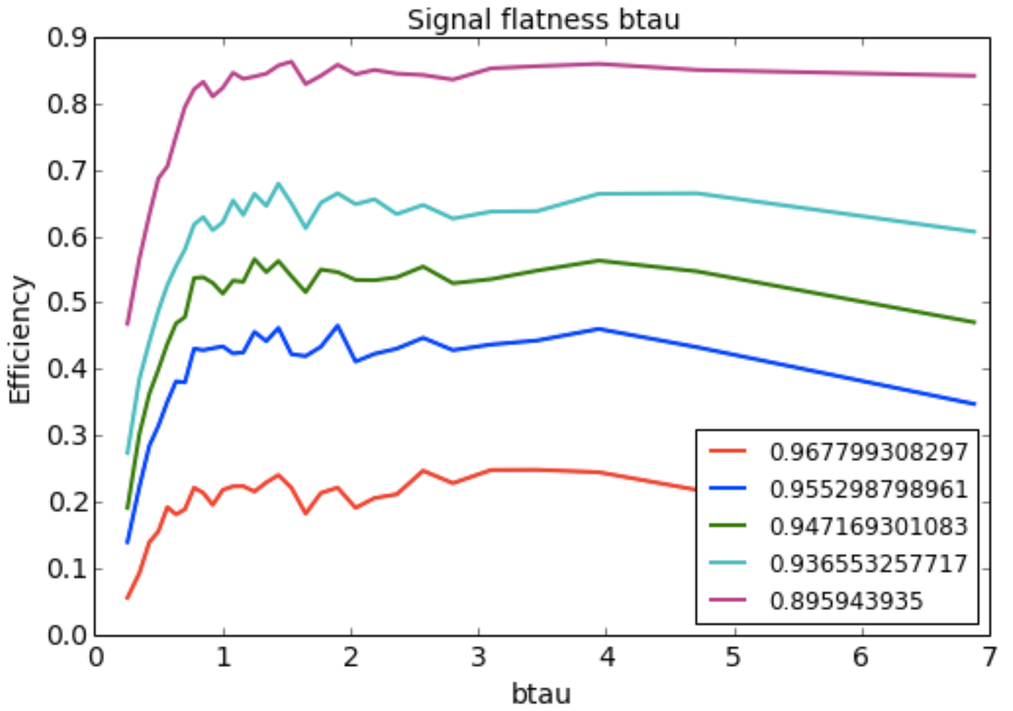



Features strongly depend on the momentum and transverse momentum. Both algorithms use the same set of features.
Used MVA is a specific BDT implementation with flatness loss.

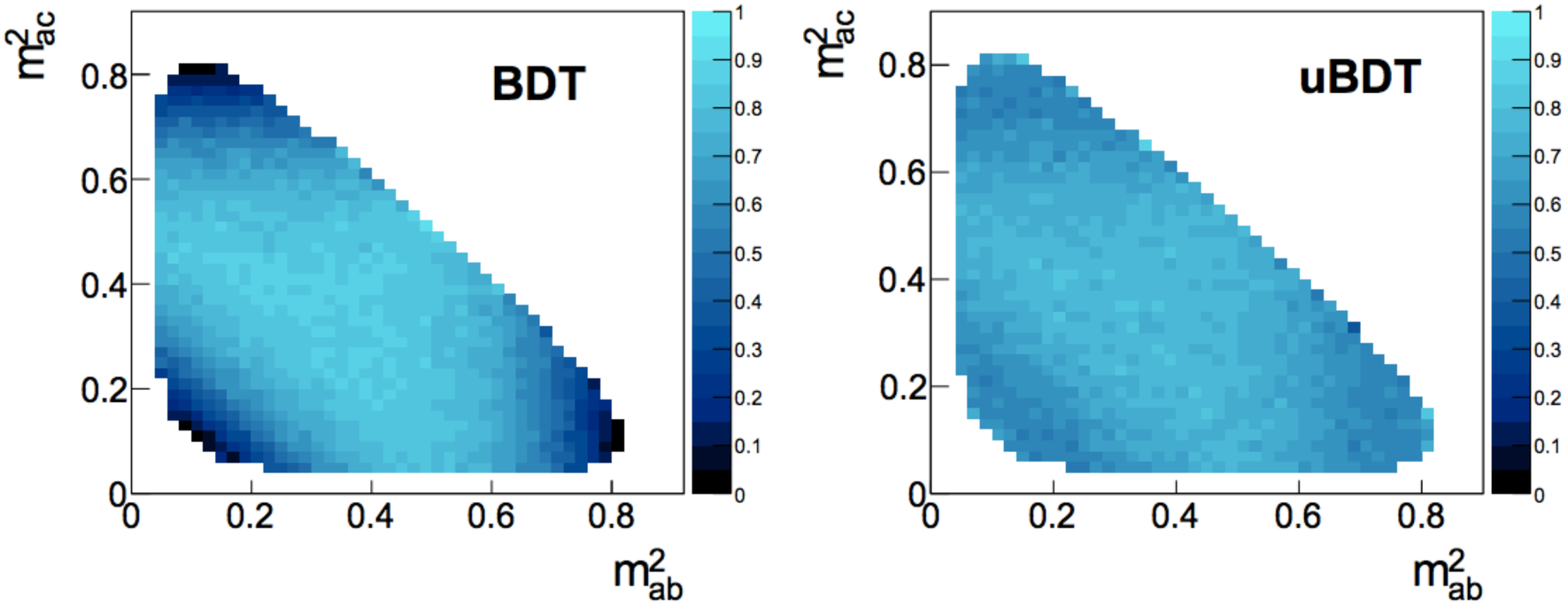
# Trigger DEMO




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

# One-hot encoding

$$\text{[proton, pion, kaon]} \rightarrow [[1,0,0],[0,1,0],[0,0,1]]$$

# 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

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

› Write the expected risk

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

# Minimum of the expected risk: the proof

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

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

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

$$\begin{aligned}
&\mathbb{E}_x \mathbb{E}_y \Big[ \big(y - \mathbb{E}(y \mid x)\big) \big(\mathbb{E}(y \mid x) - h(x)\big) \mid x \Big] = \\
&= \mathbb{E}_x \Big( \big(\mathbb{E}(y \mid x) - h(x)\big) \mathbb{E}_y \Big[ \big(y - \mathbb{E}(y \mid x)\big) \mid x \Big] \Big) = \\
&= \mathbb{E}_x \Big( \big(\mathbb{E}(y \mid x) - h(x)\big) \big(\mathbb{E}(y \mid x) - \mathbb{E}(y \mid x)\big) \Big) = \\
&= 0
\end{aligned}$$

# Minimum of the expected risk: the proof

› We obtain that the expected risk has the form

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

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

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

# Bias-variance decomposition

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

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

$$= \int_{(\mathbb{X}\times\mathbb{Y})^\ell} \int_{\mathbb{X}\times\mathbb{Y}} \left(y - \mu(X^\ell)(x)\right)^2 p(x,y) \prod_{i=1}^{\ell} p(x_i, y_i) dx dy dx_i dy_i$$

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

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

# Bias-variance decomposition

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

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

› Transforming the second summand

$$\mathbb{E}_{x,y}\Big[\mathbb{E}_{X^\ell}\Big[\big(\mathbb{E}[y \mid x] - \mu(X^\ell)\big)^2\Big]\Big] =$$

$$= \mathbb{E}_{x,y}\Big[\mathbb{E}_{X^\ell}\Big[\big(\mathbb{E}[y \mid x] - \mathbb{E}_{X^\ell}\big[\mu(X^\ell)\big] + \mathbb{E}_{X^\ell}\big[\mu(X^\ell)\big] - \mu(X^\ell)\big)^2\Big]\Big] =$$

$$= \mathbb{E}_{x,y}\Big[\mathbb{E}_{X^\ell}\Big[\underbrace{\big(\mathbb{E}[y \mid x] - \mathbb{E}_{X^\ell}\big[\mu(X^\ell)\big]\big)^2}_{\text{independent of } X^\ell}\Big]\Big] + \mathbb{E}_{x,y}\Big[\mathbb{E}_{X^\ell}\Big[\big(\mathbb{E}_{X^\ell}\big[\mu(X^\ell)\big] - \mu(X^\ell)\big)$$

$$+ 2\mathbb{E}_{x,y}\Big[\mathbb{E}_{X^\ell}\Big[\big(\mathbb{E}[y \mid x] - \mathbb{E}_{X^\ell}\big[\mu(X^\ell)\big]\big)\big(\mathbb{E}_{X^\ell}\big[\mu(X^\ell)\big] - \mu(X^\ell)\big)\Big]\Big]$$

# Bias-variance decomposition

› We prove that the last term is zero:

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