MACHINE LEARNING IN HIGH ENERGY PHYSICS

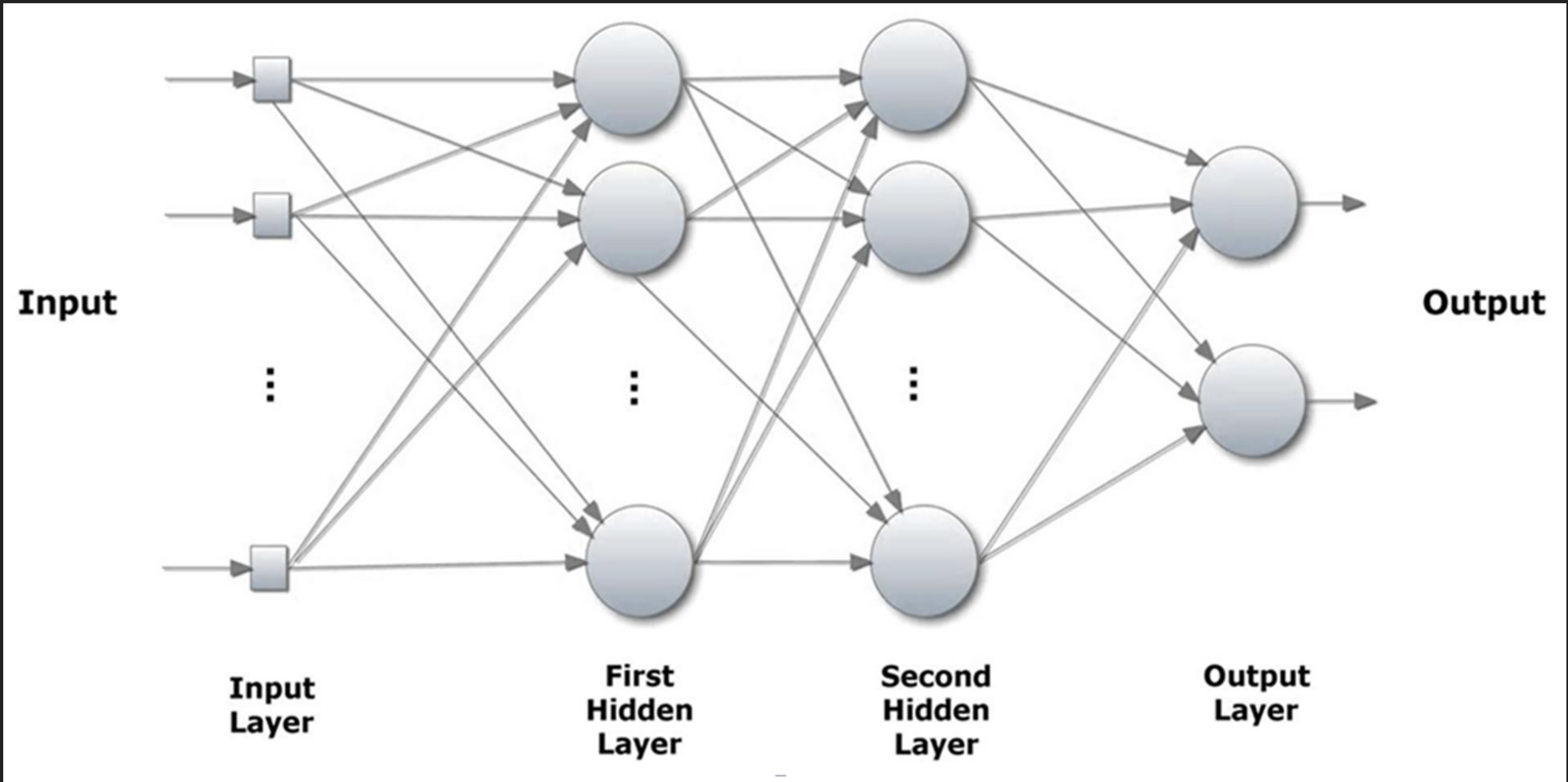
LECTURE #3

Alex Rogozhnikov, 2015

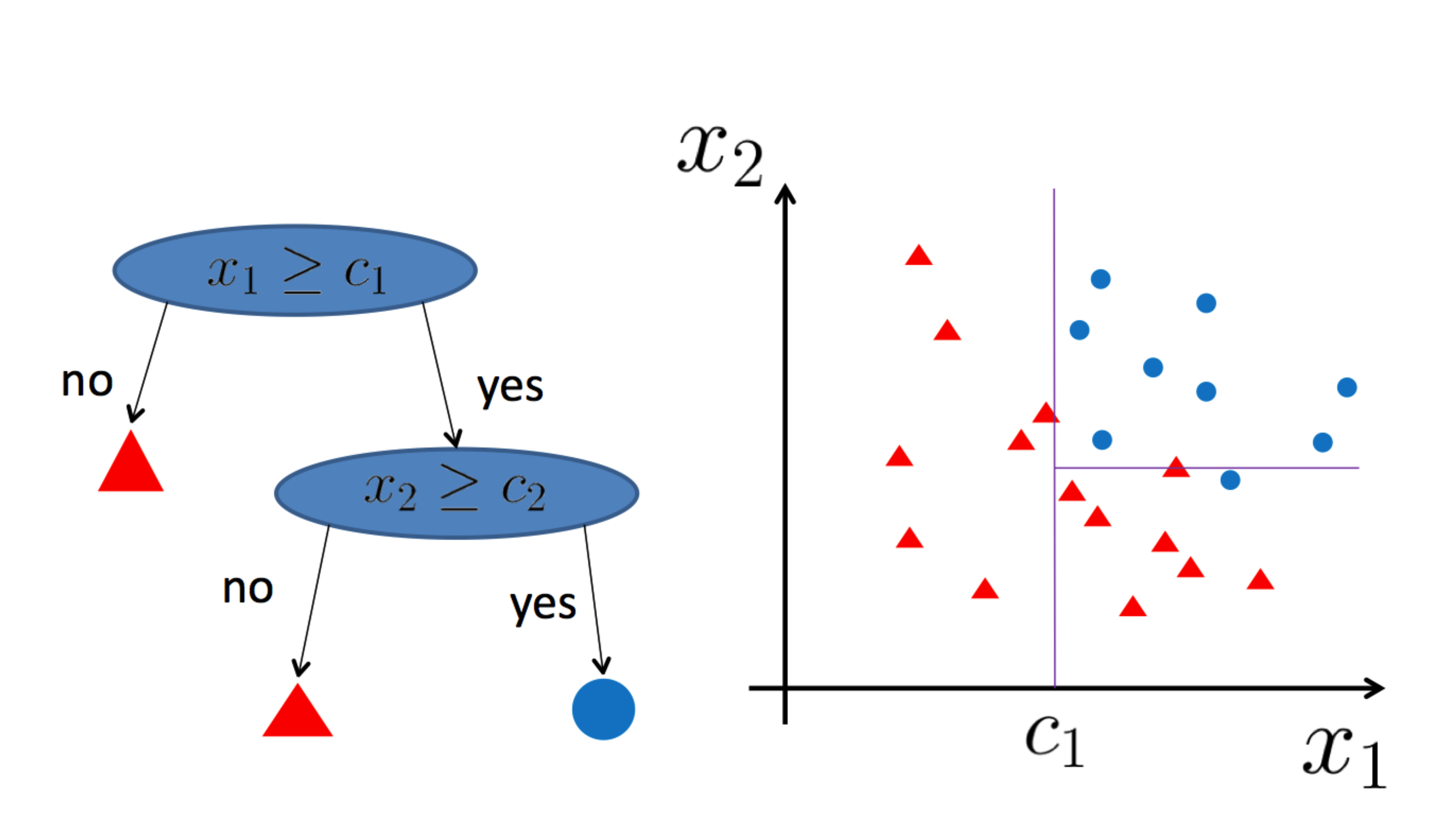
RECAPITULATION

- logistic regression and SVM
- projections, kernels and regularizations
- overfitting (2 definitions)
- stochastic optimization

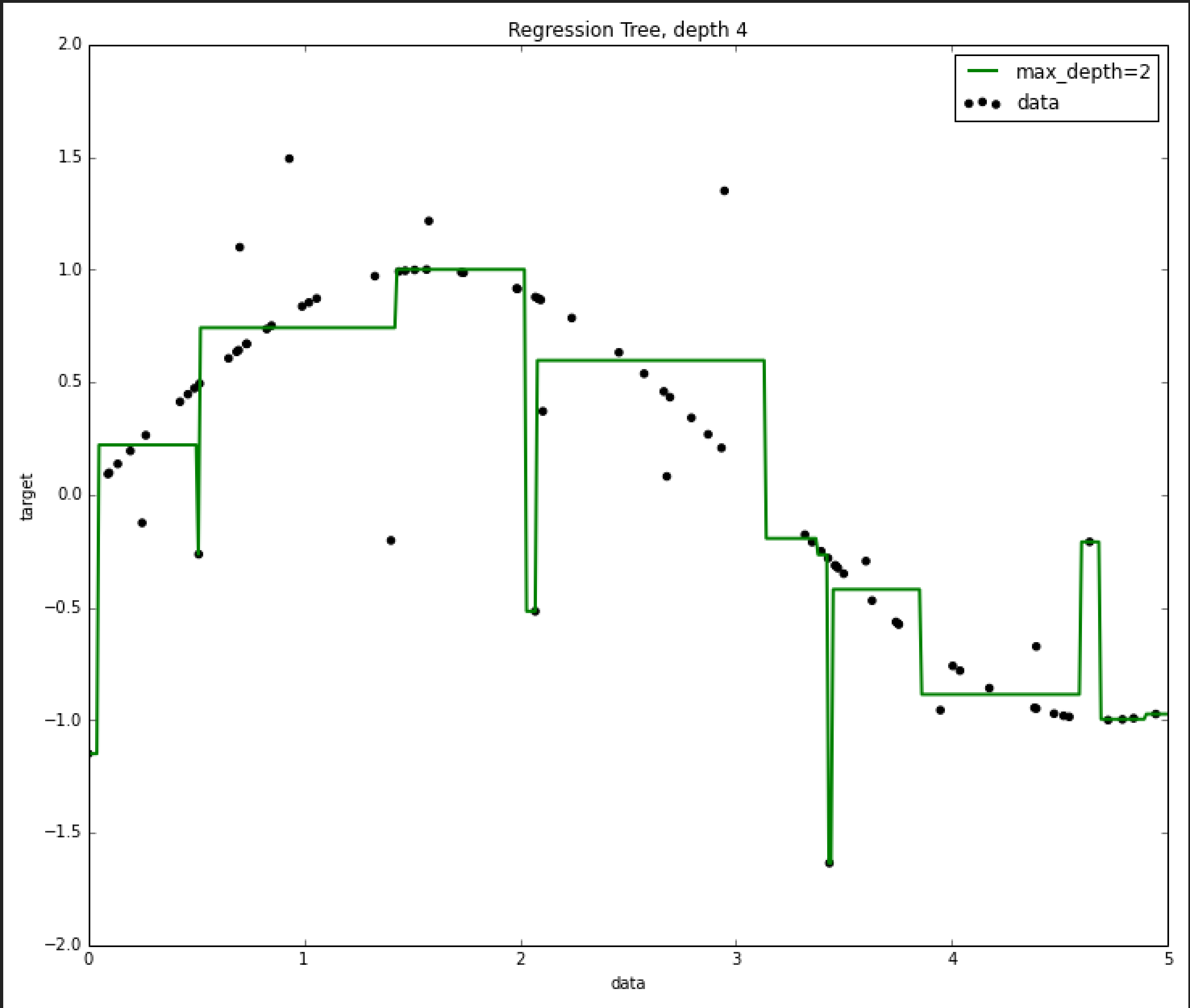
NEURAL NETWORKS



DECISION TREES FOR CLASSIFICATION

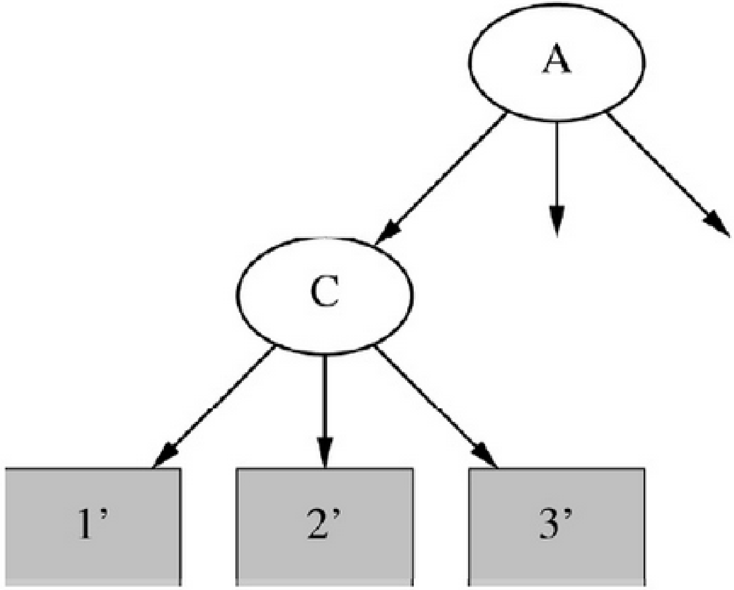


DECISION TREES FOR REGRESSION

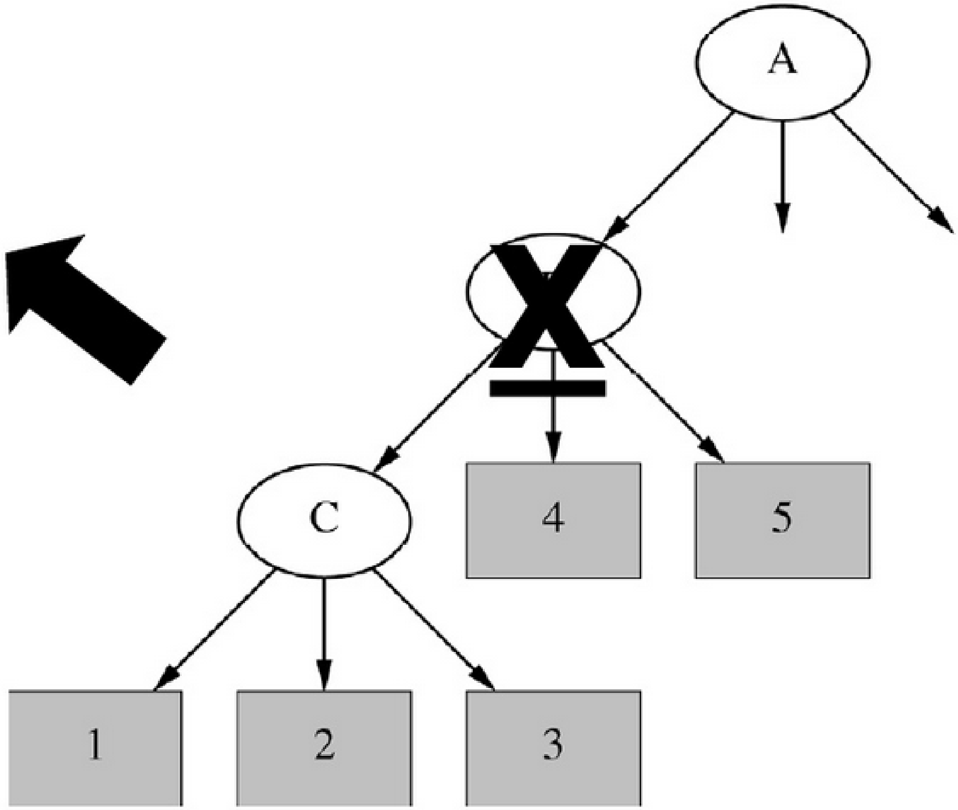


PRUNING

Subtree Raising

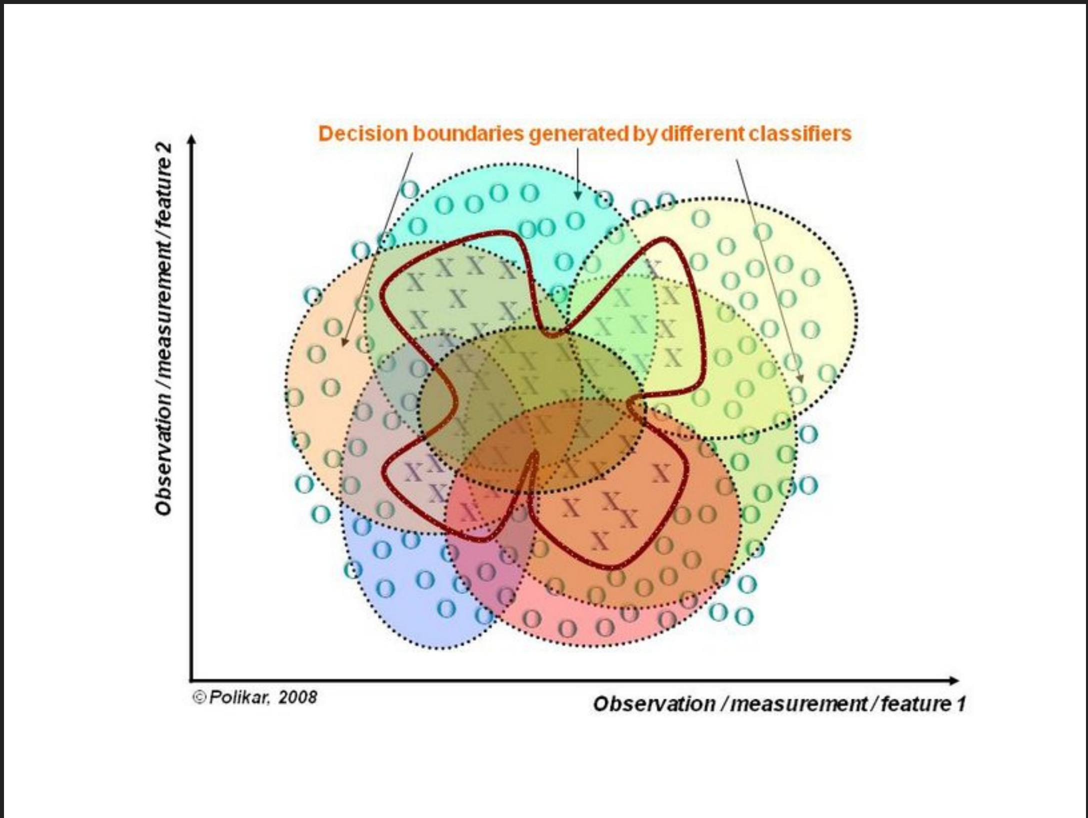


- Delete node
- Redistribute instances
- Slower than subtree replacement
  (Worthwhile?)



COMPOSITIONS

Basic motivation: improve quality of classification by reusing strong sides of classifiers.



SIMPLE VOTING

- Averaging predictions of

$$\hat{y} = [-1, +1, +1, +1, -1] \;\Rightarrow\; P_{+1} = 0.6, P_{-1} = 0.4$$

- Averaging predicted probabilities

$$P_{\pm 1}(x) = \frac{1}{J} \sum_{j=1}^{J} p_{\pm 1, j}(x)$$

- Averaging decision function

$$D(x) = \frac{1}{J} \sum_{j=1}^{J} d_j(x)$$

WEIGHTED VOTING

The way to introduce importance of classifiers

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

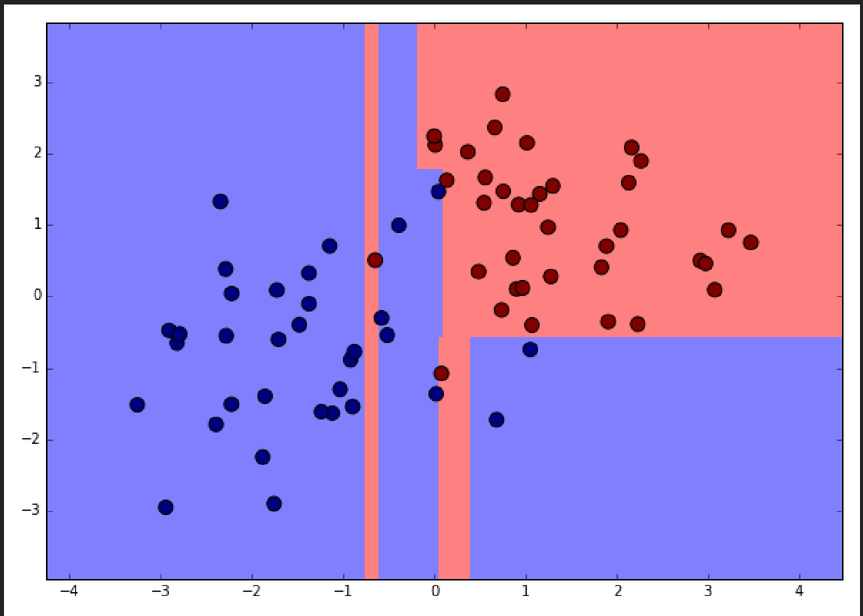
GENERAL CASE OF ENSEMBLING:

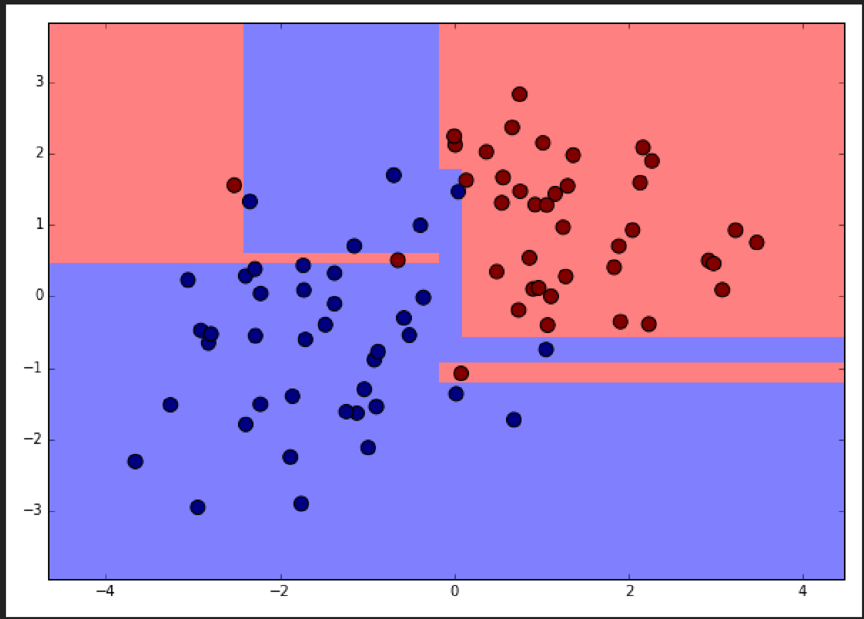
$$D(x) = f(d_1(x), d_2(x), \dots, d_J(x))$$

PROBLEMS

- very close base classifiers
- need to keep variation
- and still have good quality of basic classifiers

DECISION TREE





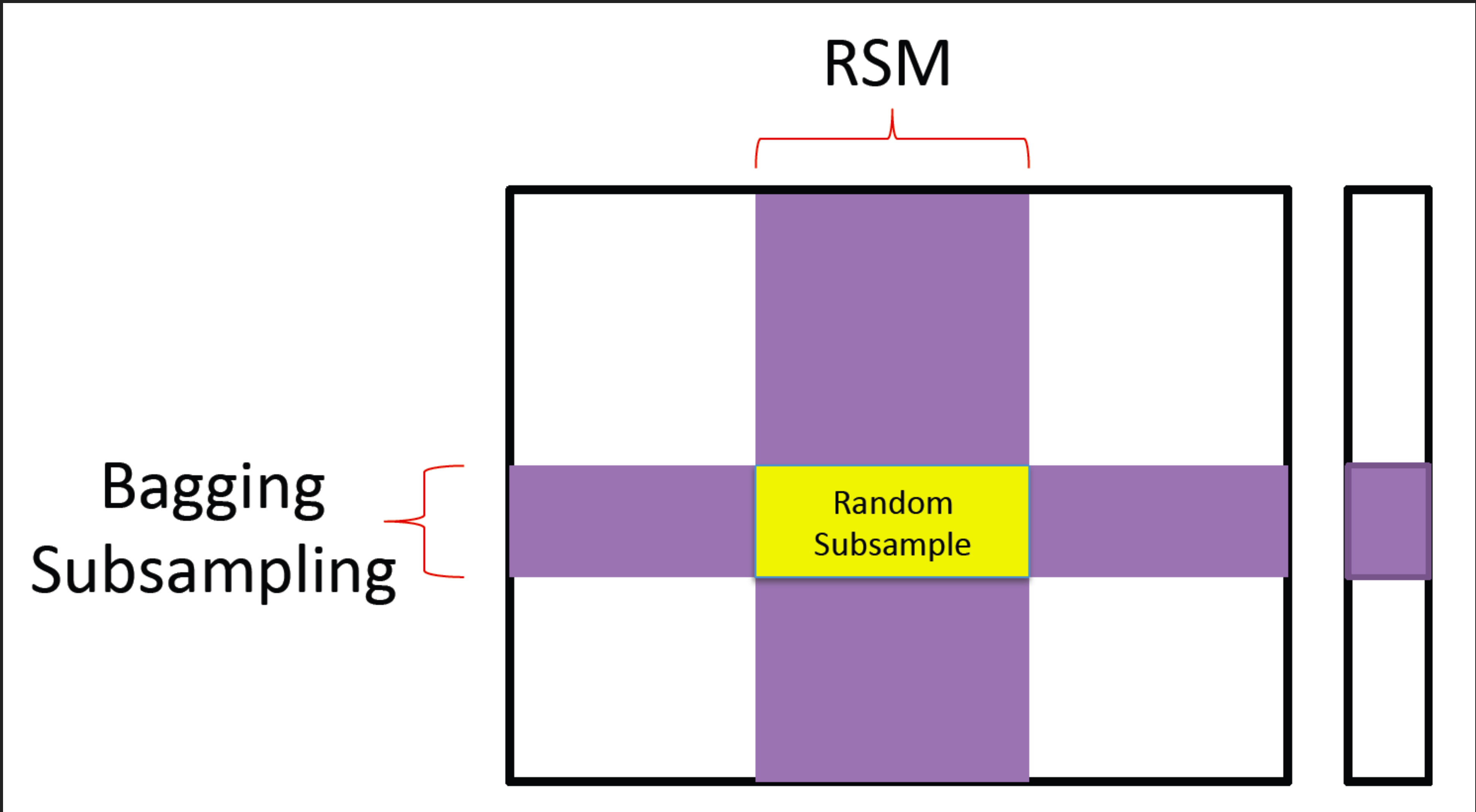
GENERATING TRAINING SUBSET

- **subsampling**
  taking fixed part of samples (sampling without replacement)
- **bagging**(Bootstrap AGGregating) sampling with replacement,

  If #generated samples = length of dataset, the fraction of unique samples in new dataset is $1 - \frac{1}{e} \sim 63.2$

RANDOM SUBSPACE MODEL (RSM)

Generating subspace of features by taking random subset of features

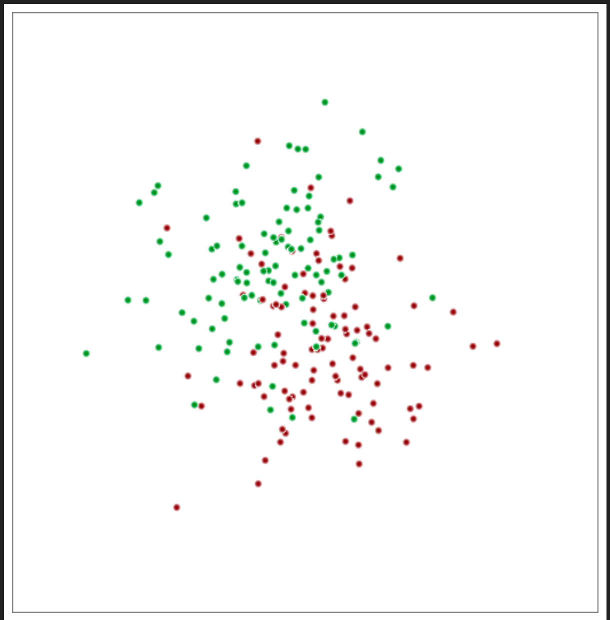


RANDOM FOREST [LEO BREIMAN, 2001]

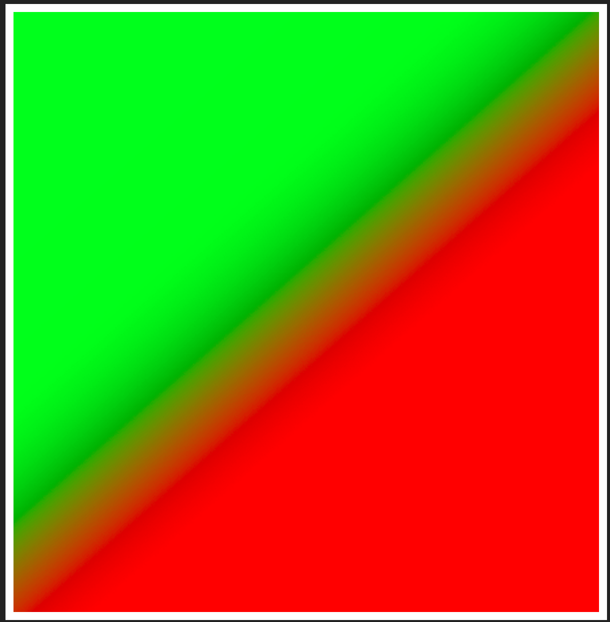
Random forest is composition of decision trees.
For each tree is trained by

- bagging samples
- taking m random features

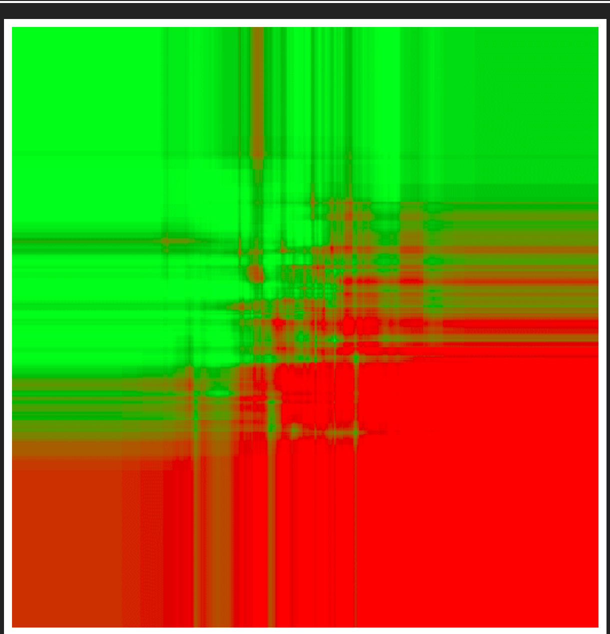
Predictions are obtained via simple voting.

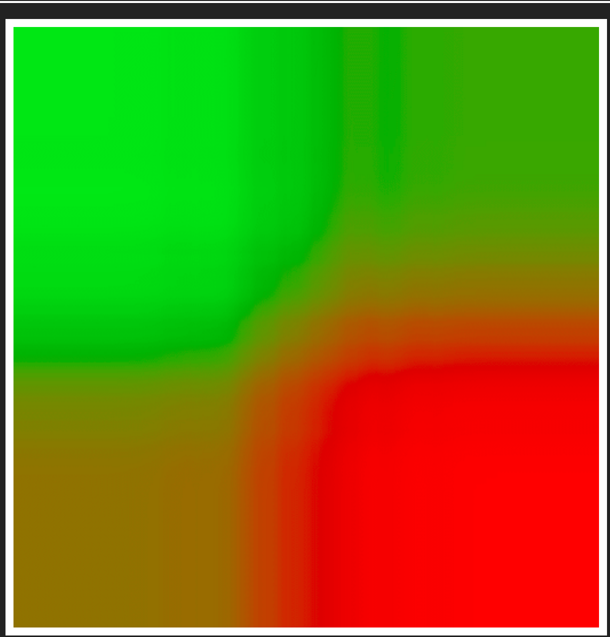


data



optimal boundary

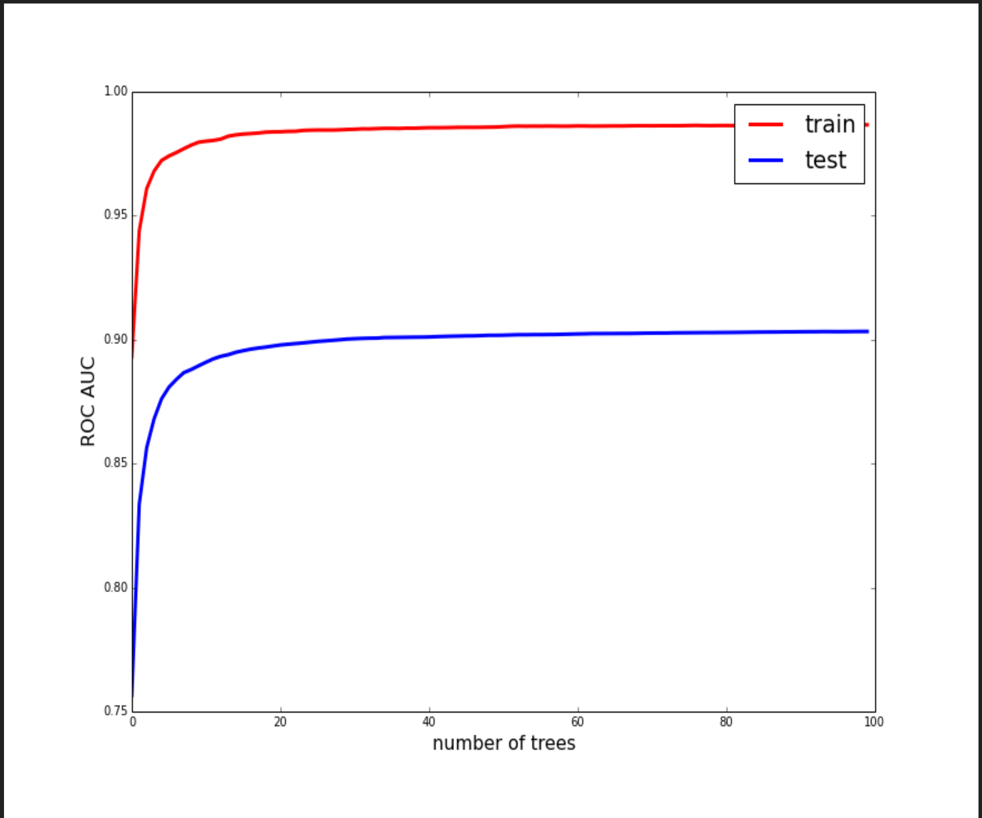


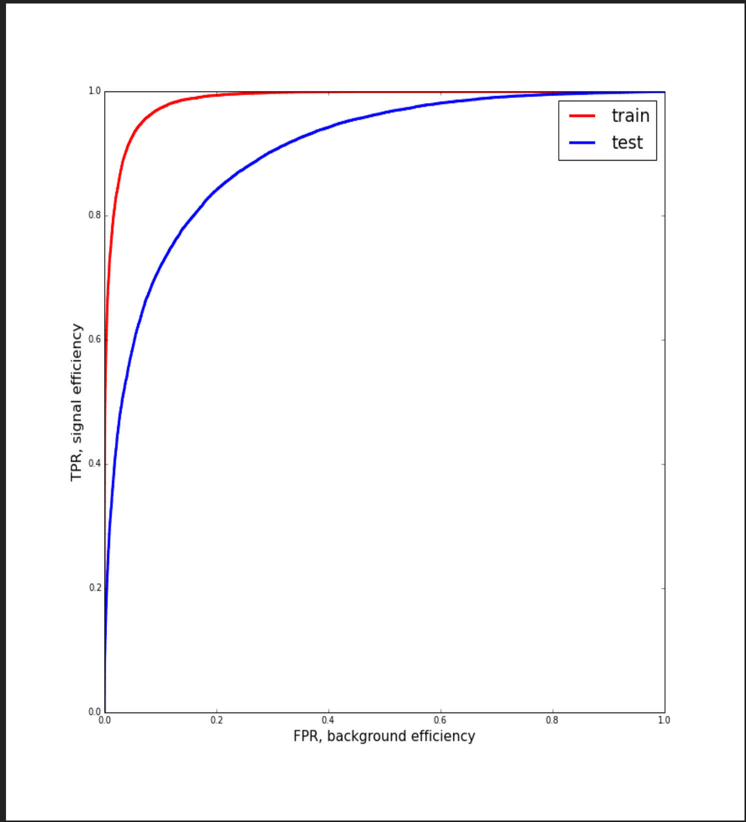


50 trees

2000 trees

OVERFITTING





- overfitted (in the sense that predictions for train and test are different)
- doesn't overfit: increasing complexity (adding more trees) doesn't spoil classifier

- Works with features of different nature
- Stable to noise in data

From 'Testing 179 Classifiers on 121 Datasets'

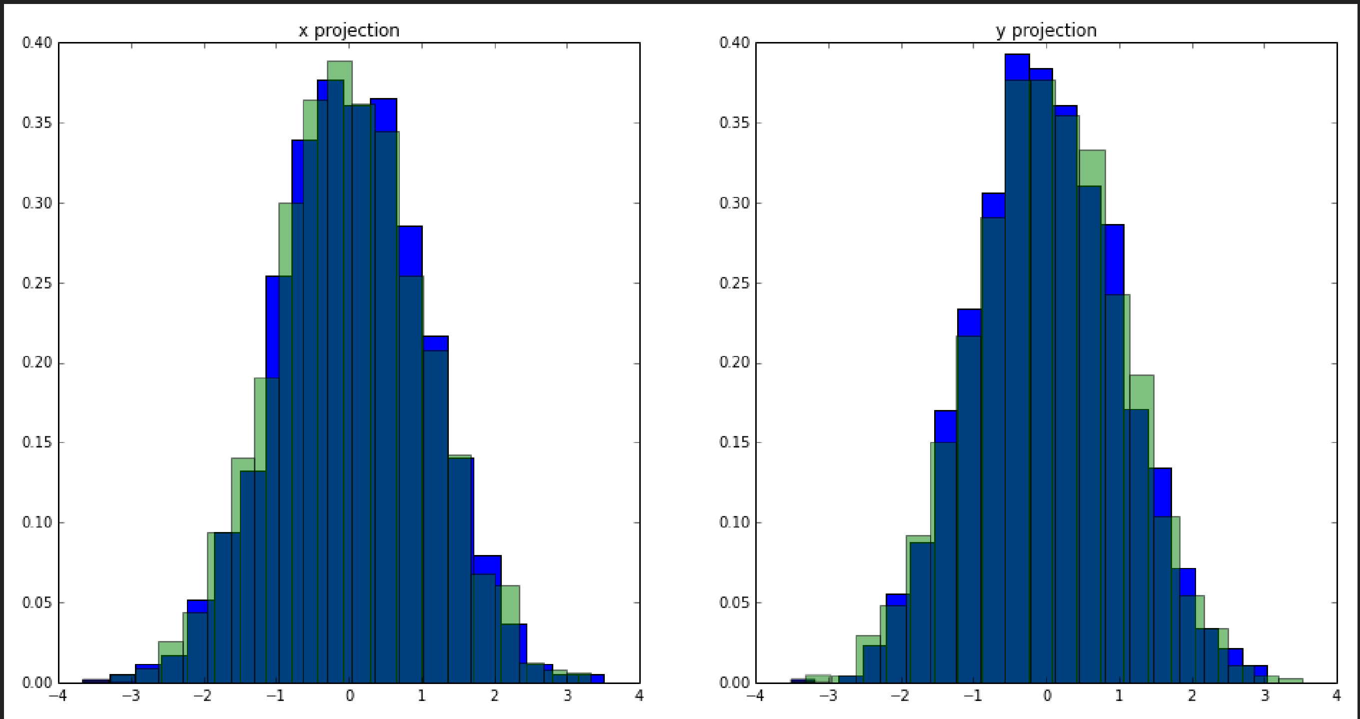
> *The classifiers most likely to be the bests are the random forest (RF) versions, the best of which [...] achieves 94.1% of the maximum accuracy overcoming 90% in the 84.3% of the data sets.*

RANDOM FOREST SUMMARY

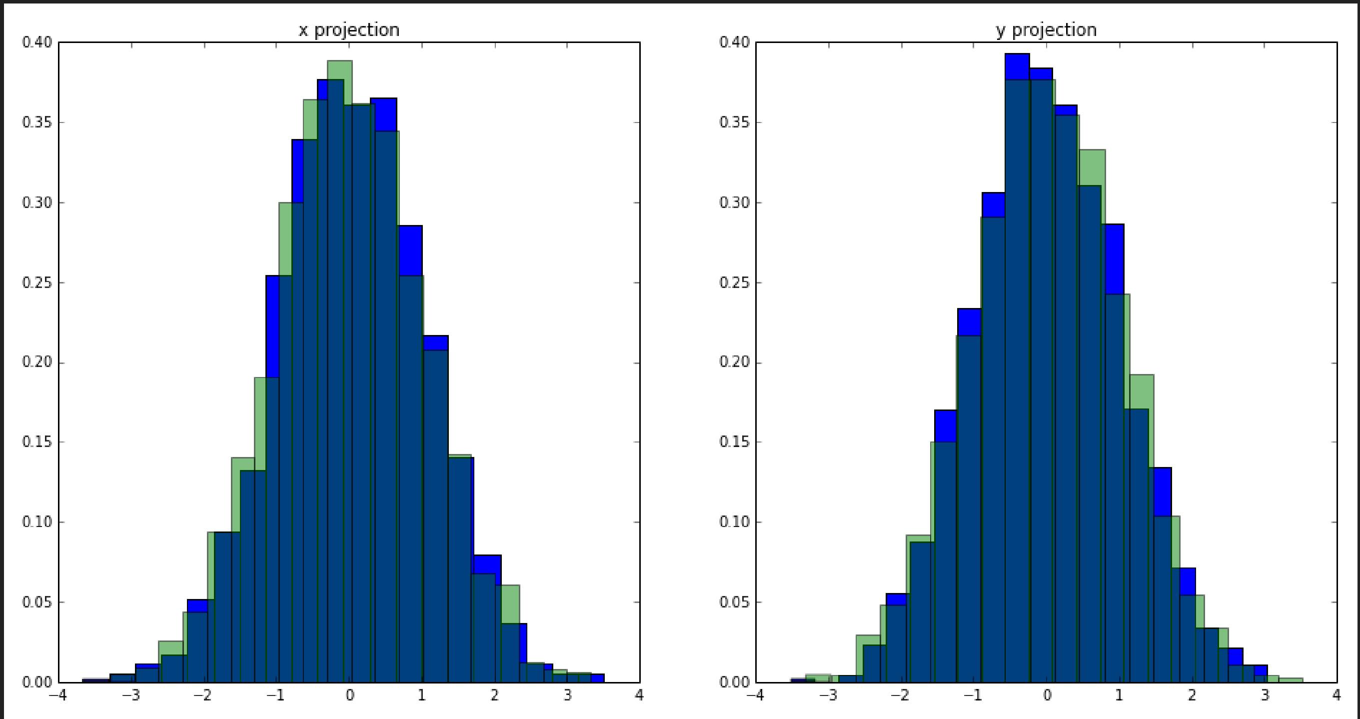
- Impressively simple
- Trees can be trained in parallel
- Doesn't overfit
- Doesn't require much tuning
  Effectively only one parameter:
  number of features used in each tree
  Recommendation: $N_{\text{used}} = \sqrt{N_{\text{features}}}$
- Hardly interpretable

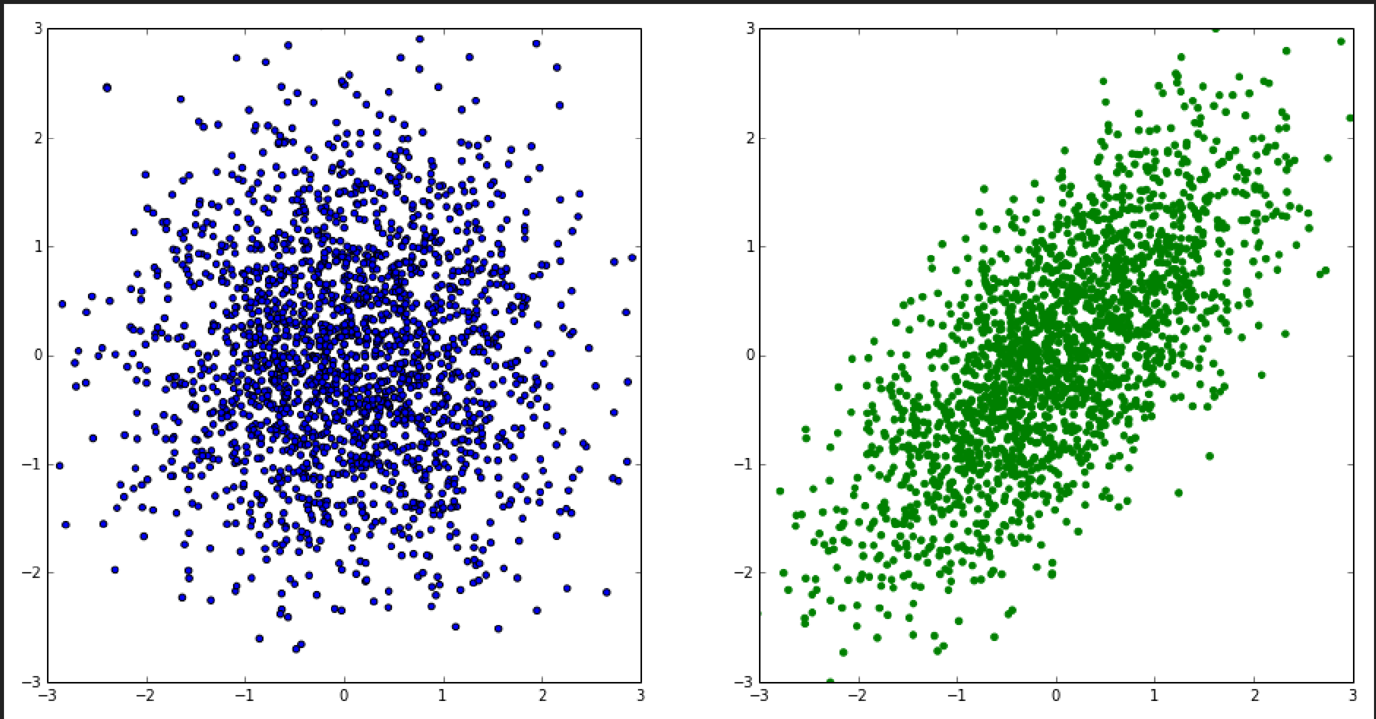
COMPARING DISTRIBUTIONS

- 1d: Kolmogorov-Smirnov
- more features is a problem, but we can compute KS over each of variables
- hardly 1d results can be combined together



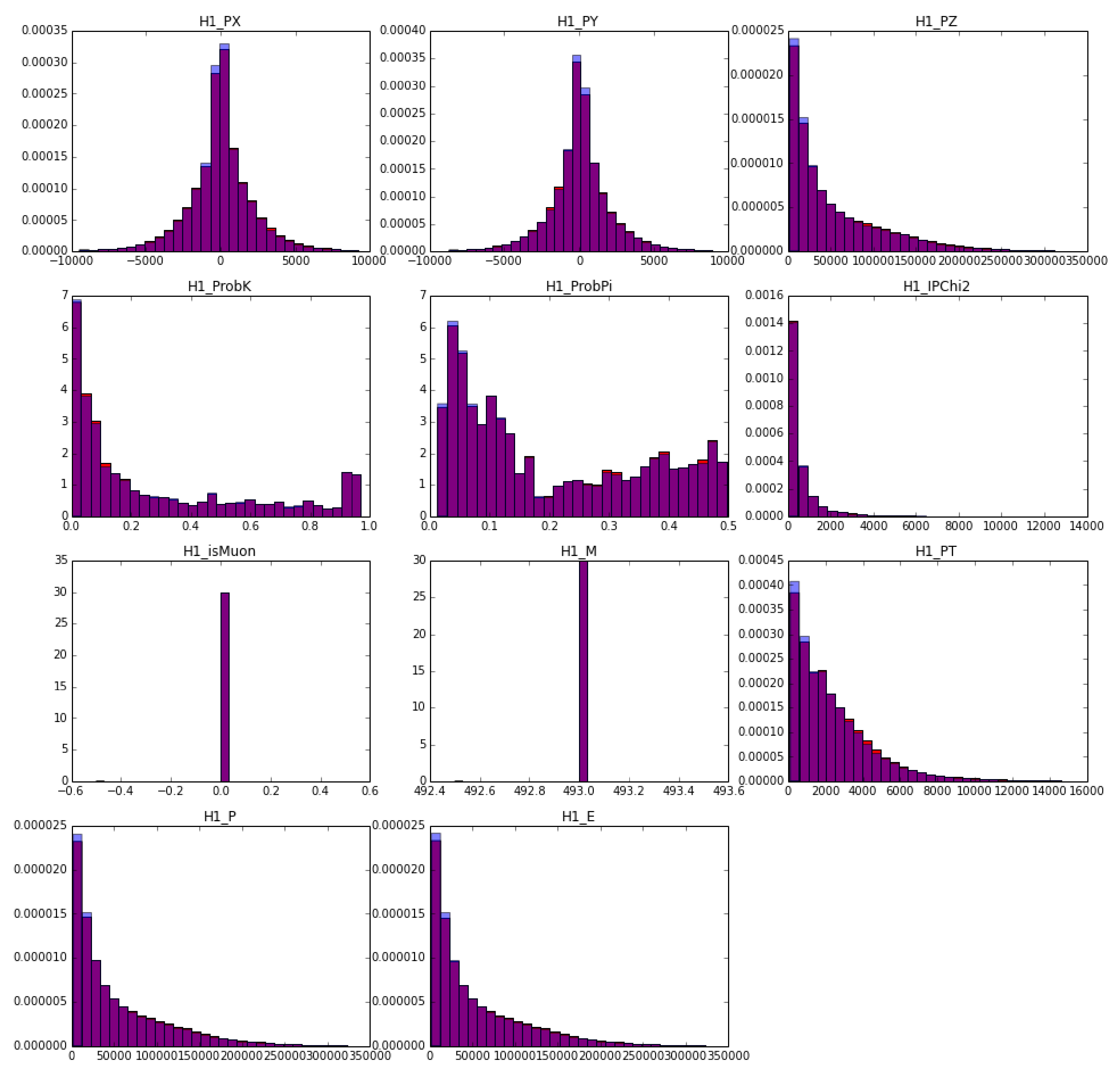
COMPARING DISTRIBUTIONS



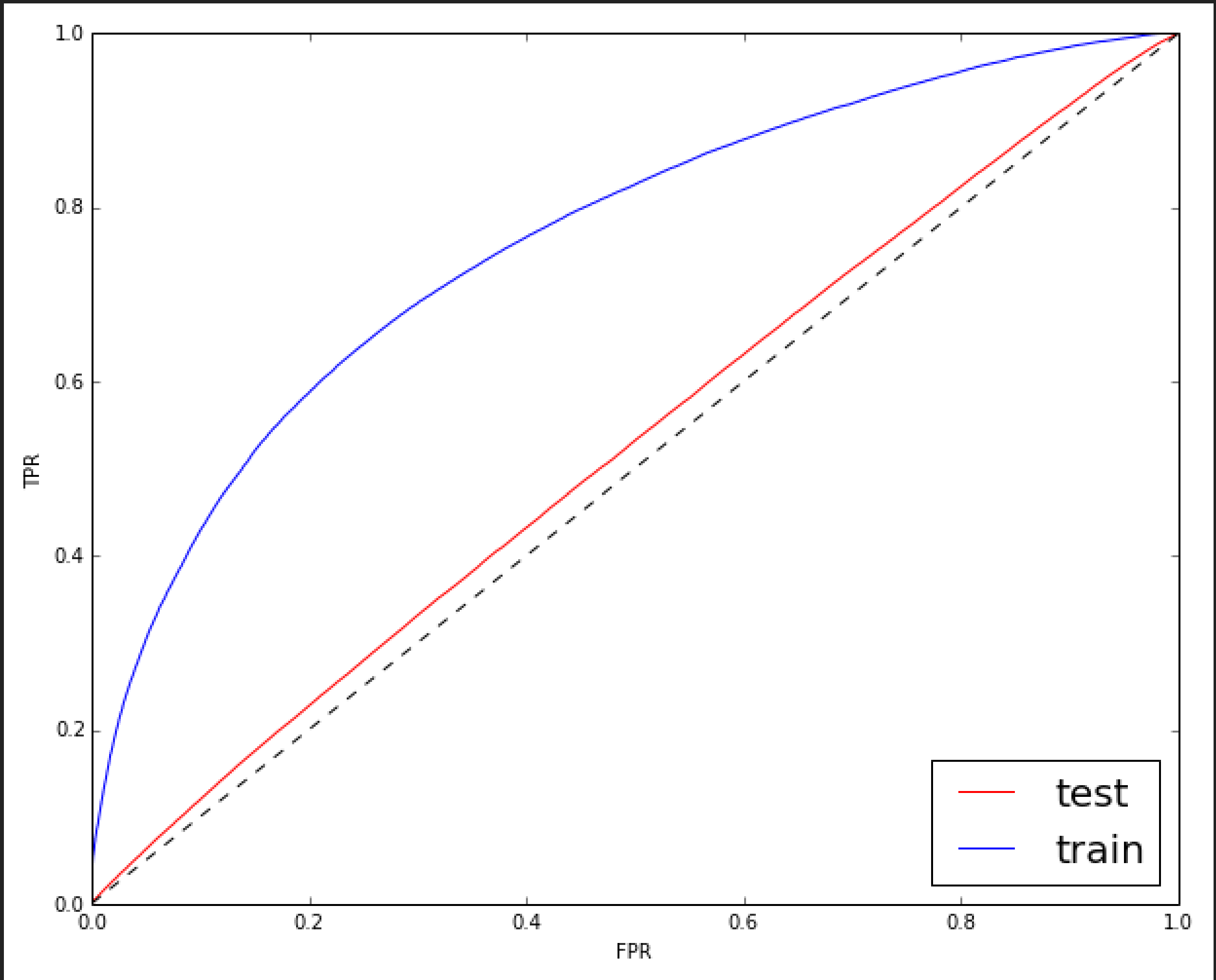


3
2
1
0
−1
−2
−3
−3 −2 −1 0 1 2 3

COMPARING DISTRIBUTIONS OF POSITIVE AND NEGATIVE TRACKS


H1_PX
H1_PY
H1_PZ
H1_ProbK
H1_ProbPi
H1_IPChi2
H1_isMuon
H1_M
H1_PT
H1_P
H1_E

USING CLASSIFIER



Want to compute significance?

Use ROC AUC + Mann-Whitney U test

SAMPLE WEIGHTS IN ML

Can be used with many estimators.

$$x_i, y_i, w_i \qquad i - \text{ index of event}$$

- weight corresponds to frequency of observation
- expected behavior: $w_i = n$ is the same as having n copies of ith event
- global normalization doesn't matter

Example for logistic regression:

$$\mathcal{L} = \sum_i w_i L(x_i, y_i) \to \min$$

Weights (parameters) of classifier $\neq$ sample weights

In code:

```
tree = DecisionTreeClassifier(max_depth=4)
tree.fit(X, y, sample_weight=weights)
```

Sample weights are convenient way to regulate importance of training events.

Only sample weights in this lecture.

ADABOOST [FREUND, SHAPIRE, 1995]

Bagging: information from previous trees not taken into account.

Adaptive Boosting is weighted composition of weak learners:

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

We assume $d_j(x) = \pm 1$, labels $y_i = \pm 1$,
jth weak learner misclassified ith event iff $y_i d_j(x_i) = -1$

ADABOOST

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

Weak learners are built in sequence

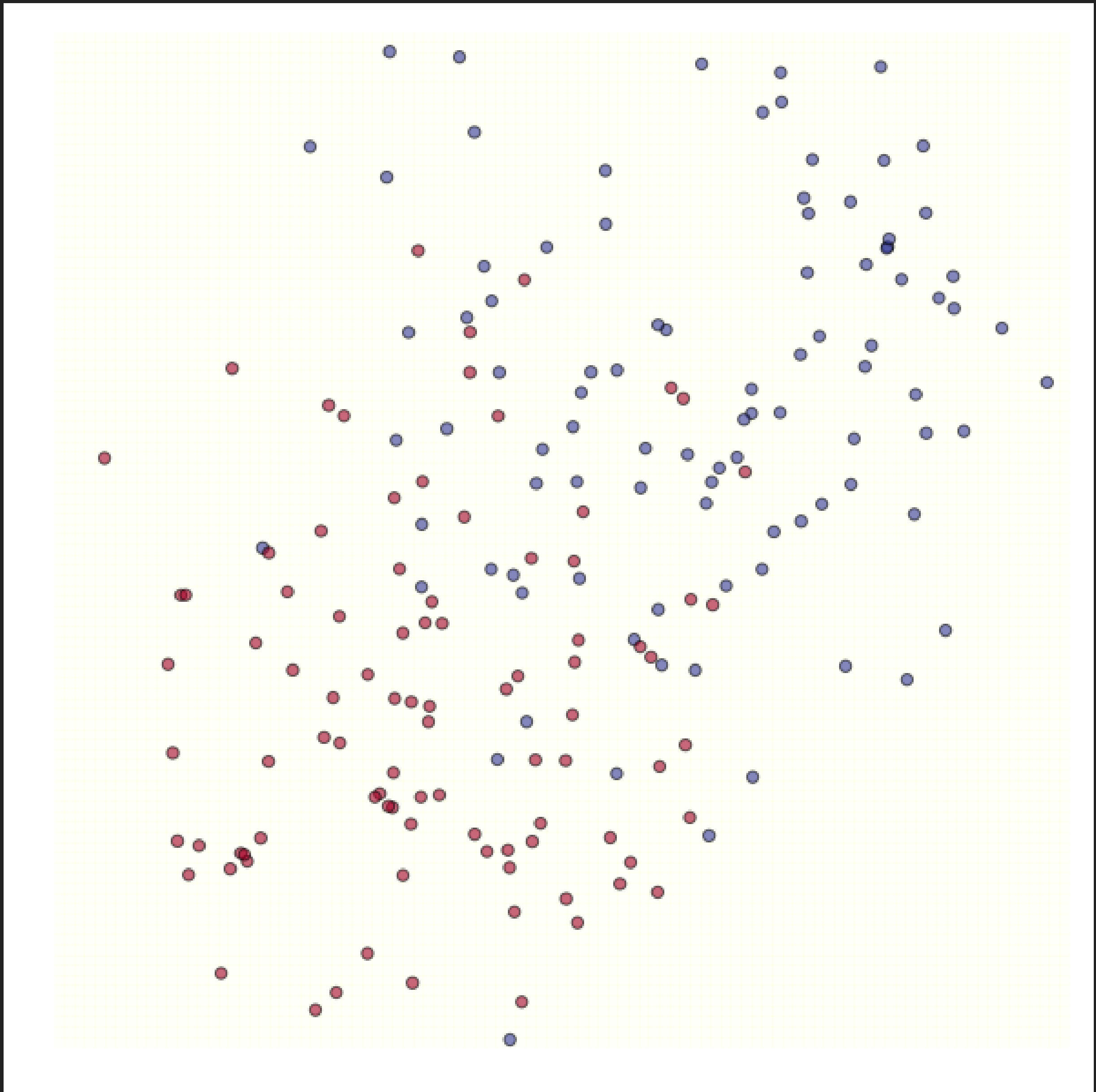
- each next classifier is trained using different weights
- initially w_i = 1 for each training sample
- After building jth base classifier:

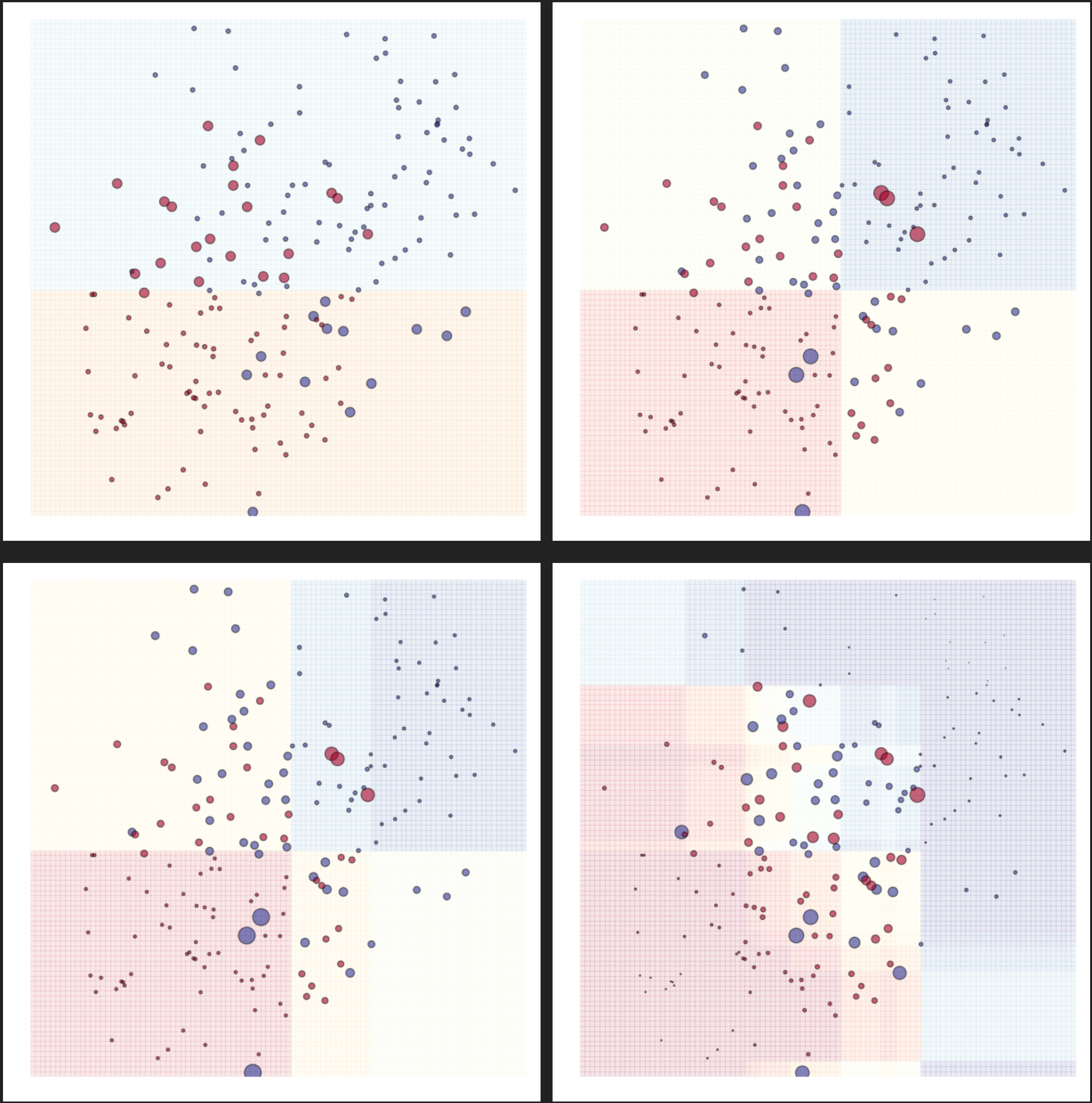
1. $\alpha_j = \frac{1}{2} \ln \left(\frac{w_{\text{correct}}}{w_{\text{wrong}}} \right)$
2. increase weight of misclassified

   $w_i \leftarrow w_i \times e^{-\alpha_j y_i d_j(x_i)}$

ADABOOST EXAMPLE

Decision trees of depth 1 will be used.





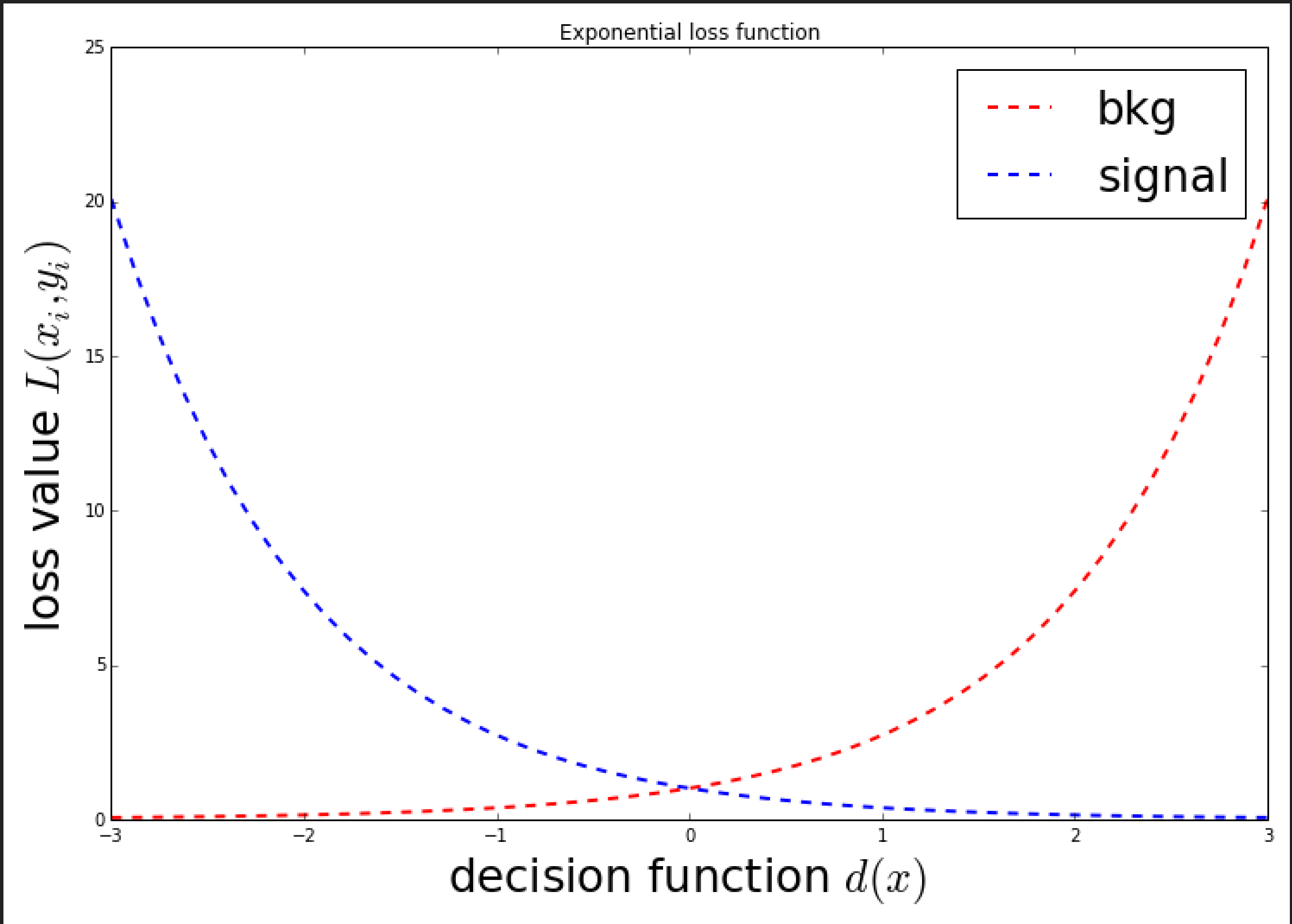
ADABOOST SECRET

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

$$\mathcal{L} = \sum_i L(x_i, y_i) = \sum_i \exp(-y_i D(x_i)) \to \min$$

- α_j is obtained as result analytical optimization
- sample weight is equal to penalty
  $w_i = L(x_i, y_i) = \exp(-y_i D(x_i))$ for event

LOSS FUNCTION OF ADABOOST

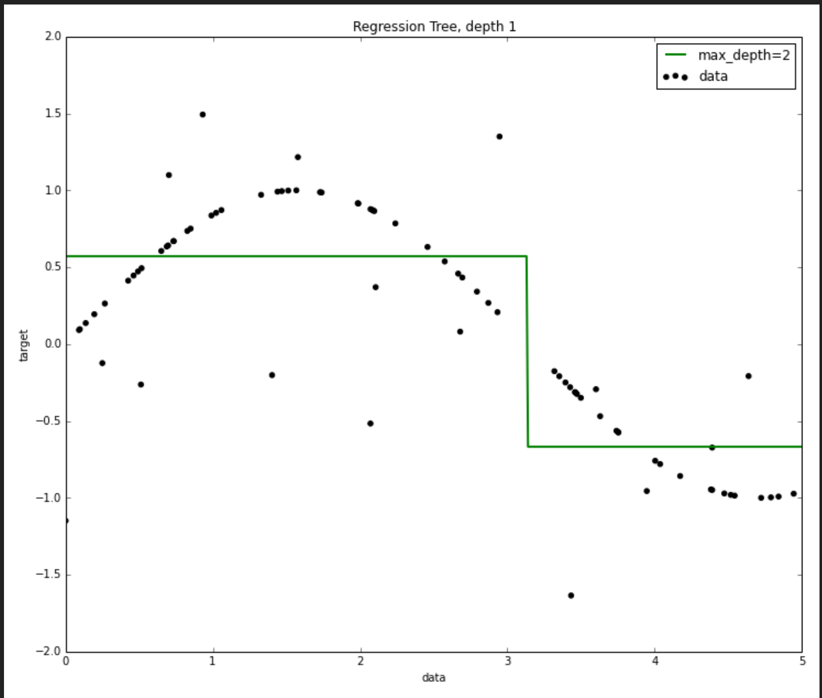


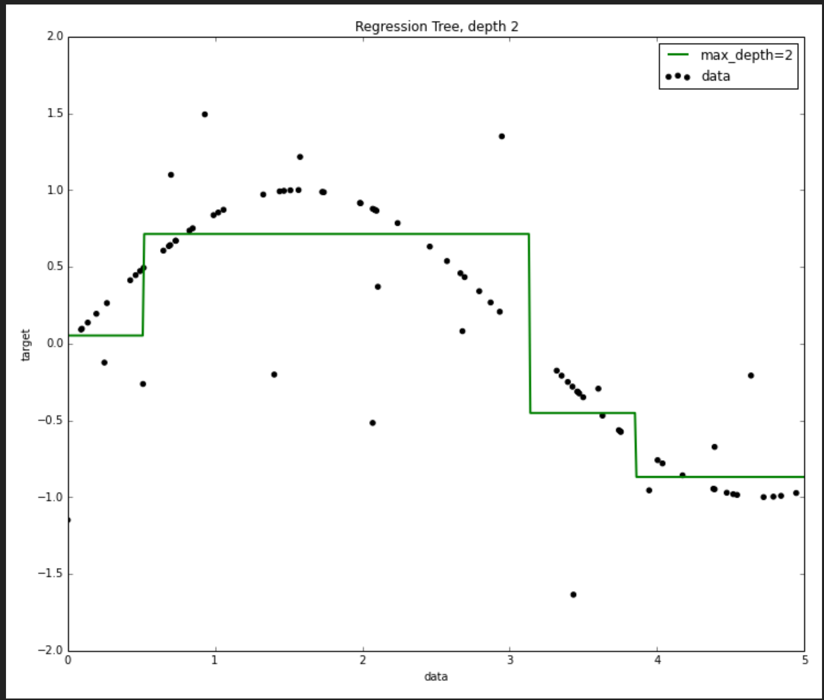
ADABOOST SUMMARY

- able to combine many weak learners
- takes mistakes into account
- simple, overhead is negligible
- too sensitive to outliers

x MINUTES BREAK

DECISION TREES FOR REGRESSION





GRADIENT BOOSTING [FRIEDMAN, 1999]

composition of weak learners,

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

$$\begin{aligned} p_{+1}(x) &= \sigma(D(x)) \\ p_{-1}(x) &= \sigma(-D(x)) \end{aligned}$$

Optimization of log-likelihood:

$$\mathcal{L} = \sum_i L(x_i, y_i) = \sum_i \ln\left(1 + e^{-y_i D(x_i)}\right) \to \min$$

GRADIENT BOOSTING

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

$$\mathcal{L} = \sum_i \ln\left(1 + e^{-y_i D(x_i)}\right) \rightarrow \min$$

- Optimization problem: find all α_j and weak leaners d_j
- Mission impossible
- Main point: greedy optimization of loss function by training one more weak learner d_j
- Each new estimator follows the gradient of loss function

GRADIENT BOOSTING

Gradient boosting ~ steepest gradient descent.

$$D_j(x) = \sum_{j'=1}^{j} \alpha_{j'} d_{j'}(x)$$

$$D_j(x) = D_{j-1}(x) + \alpha_j d_j(x)$$

At jth iteration:

- pseudo-residual $z_i = -\frac{\partial}{\partial D(x_i)} \mathcal{L} \Big|_{D(x)=D_{j-1}(x)}$
- train regressor d_j to minimize MSE: $\sum_i (d_j(x_i) - z_i)^2 \to min$
- find optimal α_j

ADDITIONAL GB TRICKS

to make training more stable, add learning rate
$D_j(x) = \sum_j \eta \alpha_j d_j(x)$

randomization to fight noise and build different trees:
subsampling of features and training samples

AdaBoost is particular case of gradient boosting with different target loss function*:

$$\mathcal{L} = \sum_i e^{-y_i D(x_i)} \to \min$$

This loss function is called ExpLoss or AdaLoss.

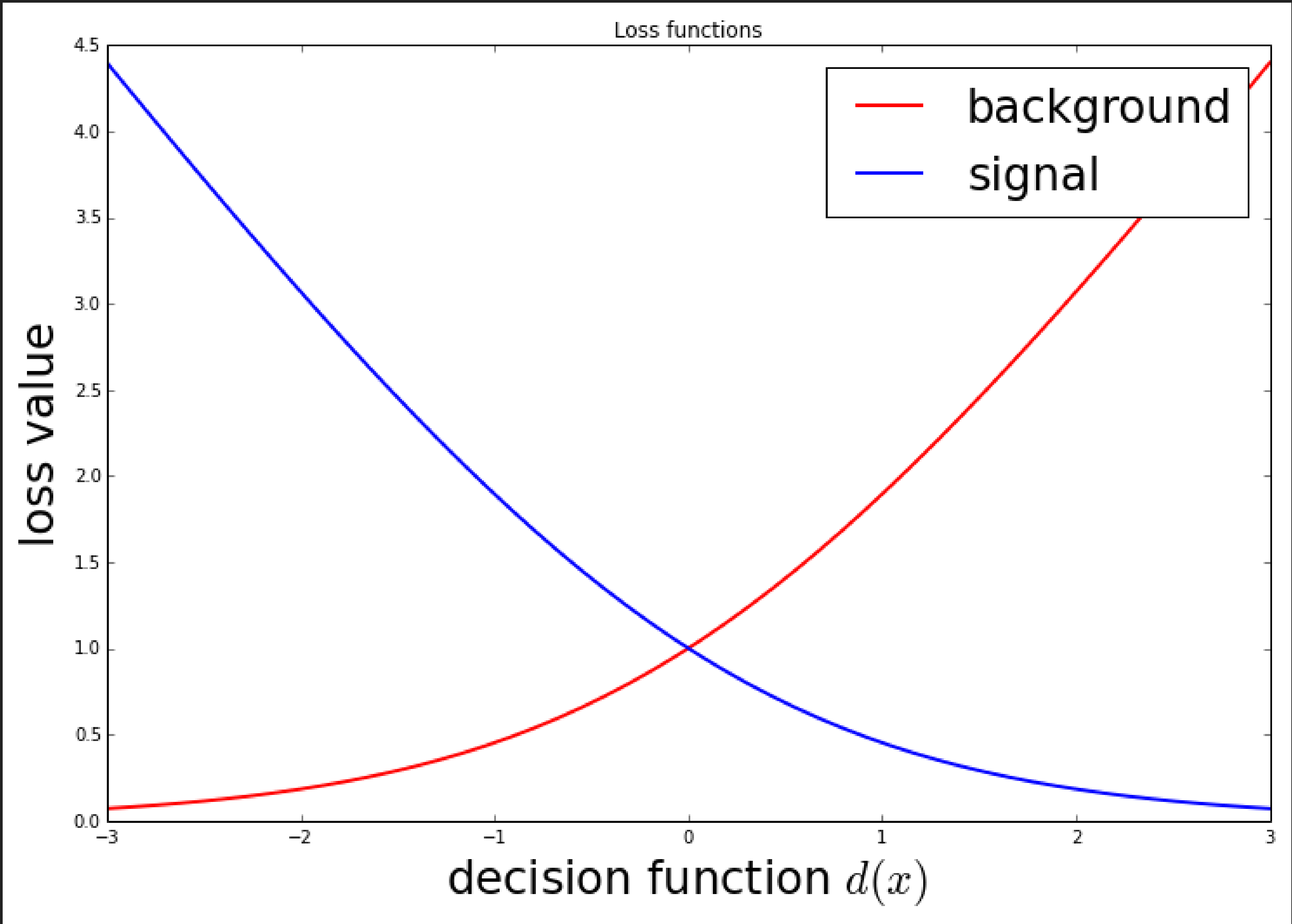
*(also AdaBoost expects that $d_j(x_i) = \pm 1$)

LOSS FUNCTIONS

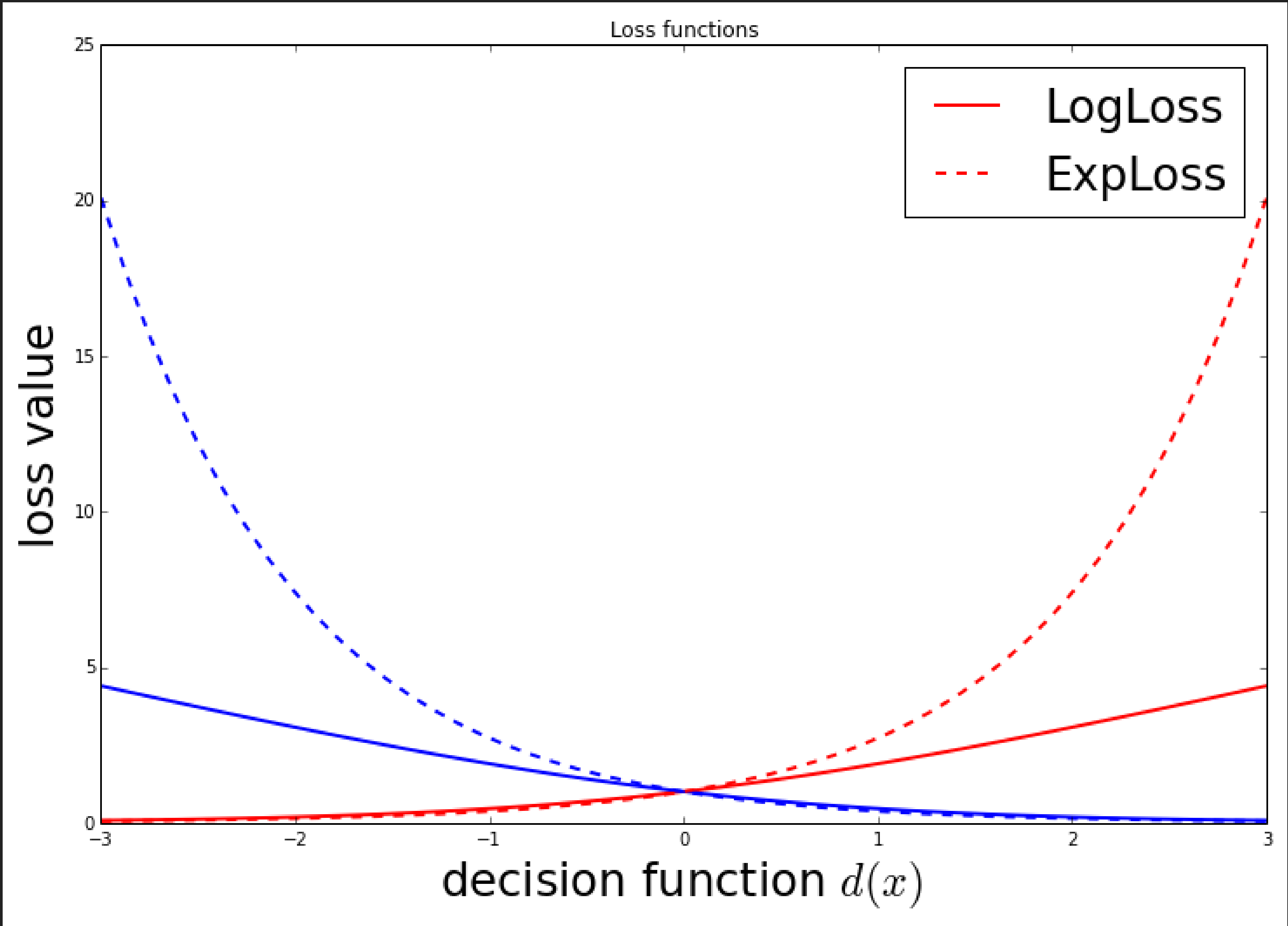
Gradient boosting can optimize different smooth loss function.

- **regression, $\mathbf{y} \in \mathbb{R}$**
  - Mean Squared Error $\sum_i (d(x_i) - y_i)^2$
  - Mean Absolute Error $\sum_i |d(x_i) - y_i|$

- **binary classification, $\mathbf{y_i = \pm 1}$**
  - ExpLoss (ada AdaLoss) $\sum_i e^{-y_i d(x_i)}$
  - LogLoss $\sum_i \log(1 + e^{-y_i d(x_i)})$

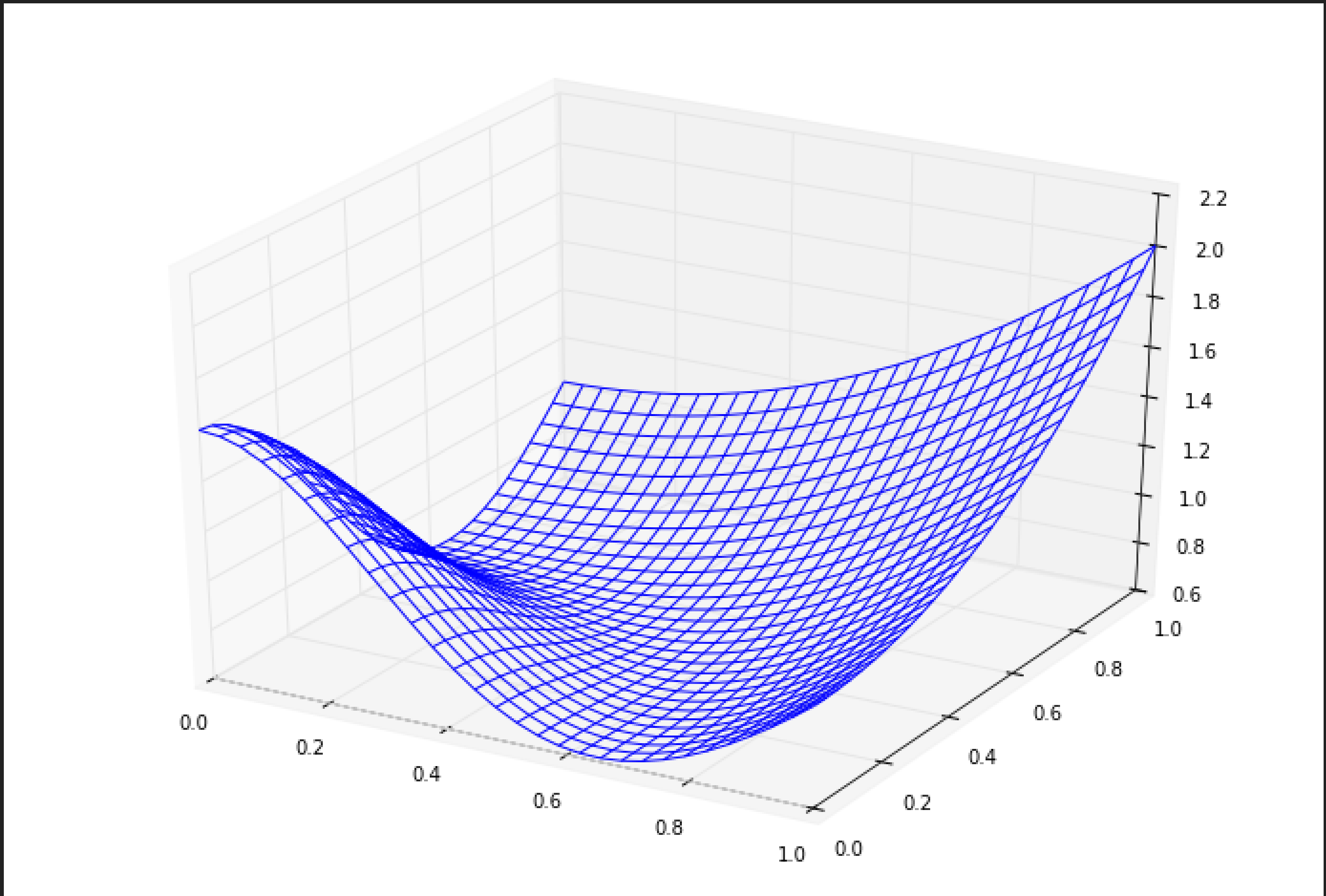


Loss functions
background
signal
loss value
decision function $d(x)$
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−3
−2
−1
0
1
2
3

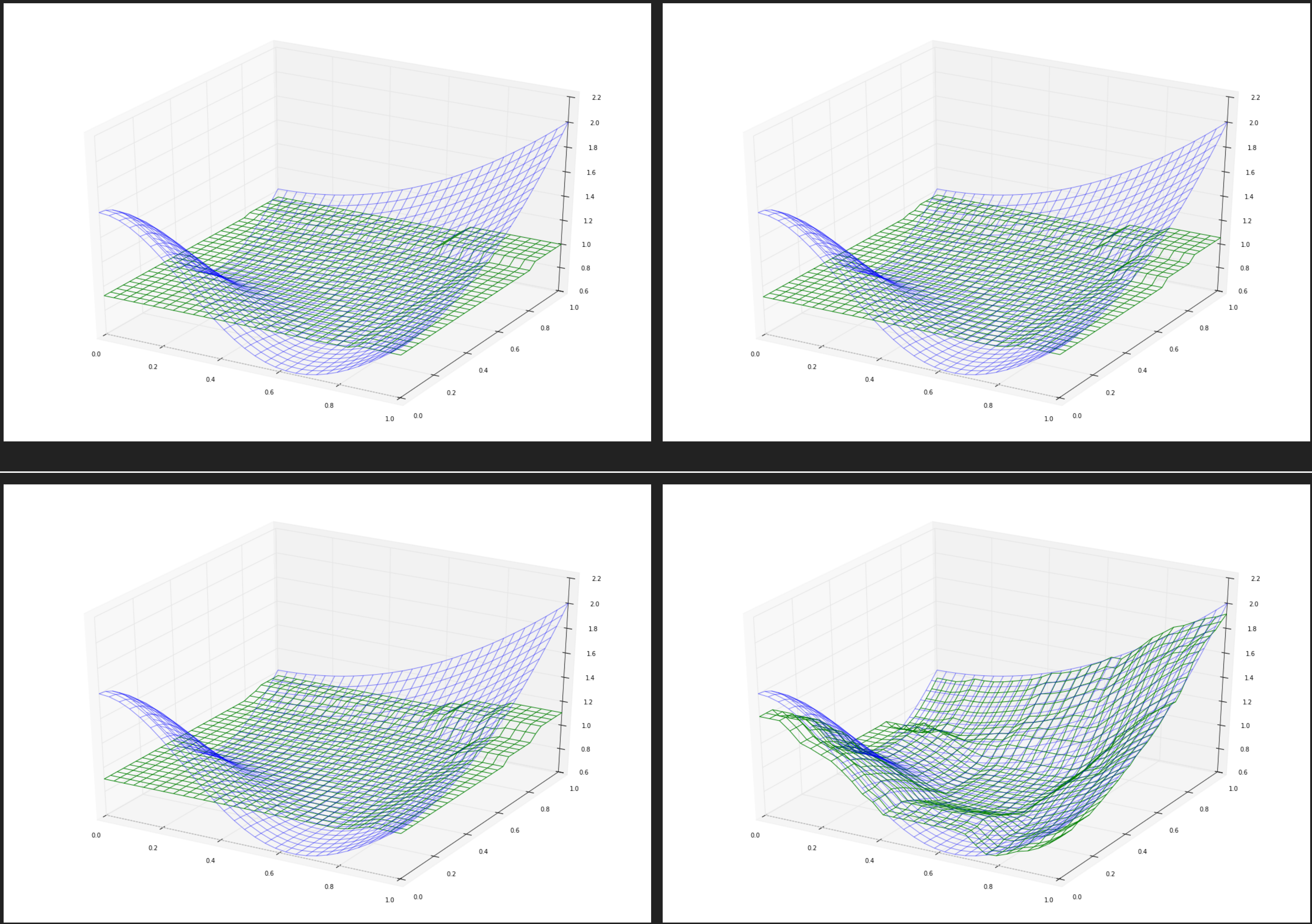


Loss functions
LogLoss
ExpLoss
loss value
decision function $d(x)$
25
20
15
10
5
0
−3
−2
−1
0
1
2
3

EXAMPLE: REGRESSION WITH GB



using regression trees of depth=2



number of trees = 1, 2, 3, 100

ADAPTING BOOSTING

By modifying boosting or changing loss function we can solve different problems

- classification
- regression
- ranking

Also we can add restrictions, i.e. fight correlation with mass

LOSS FUNCTION: RANKING EXAMPLE

In ranking we need to order items by y_i:

$y_i < y_j \;\; \Rightarrow \;\; d(x_i) < d(x_j)$

We can add penalization term for misordering:

$$\mathcal{L} = \sum_{ij} L(x_i, x_j, y_i, y_j)$$

$$L(x_i, x_j, y_i, y_j) = \begin{cases} \sigma(d(x_j) - d(x_i)), & y_i < y_j \\ 0, & \text{otherwise} \end{cases}$$

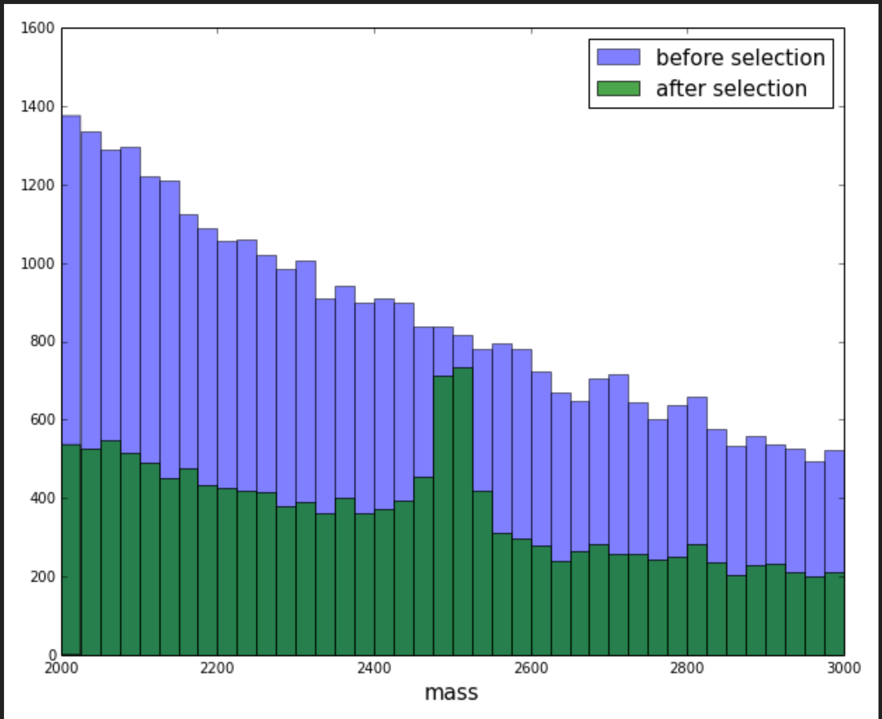
BOOSTING TO UNIFORMITY

Point of uniform boosting - have constant efficiency against some variable.

Examples:

- flat background efficiency along mass
- flat signal efficiency for different flight time
- flat signal efficiency along Dalitz variable

EXAMPLE: NON-FLAT BACKGROUND EFFICIENCY ALONG MASS



High correlation with mass will create from pure background false peaking signal

Aim: $\mathrm{FPR} = const$ for different regions in mass.

uBoostBDT

Variation of AdaBoost approach,
aim $\text{FPR}_{\text{region}} = const$.

fix target efficiency (say $\text{FPR}_{\text{target}} = 30\%$), find corresponding threshold

- Train a tree, its decision function $d_j(x)$
- increase weight for misclassification:
  $w_i \leftarrow w_i \exp\left(-\alpha y_i d_j(x)\right)$
- increase weight of signal events in regions with high FPR
  $w_i \leftarrow w_i \exp\left(\beta(\text{FPR}_{\text{region}} - \text{FPR}_{target})\right)$

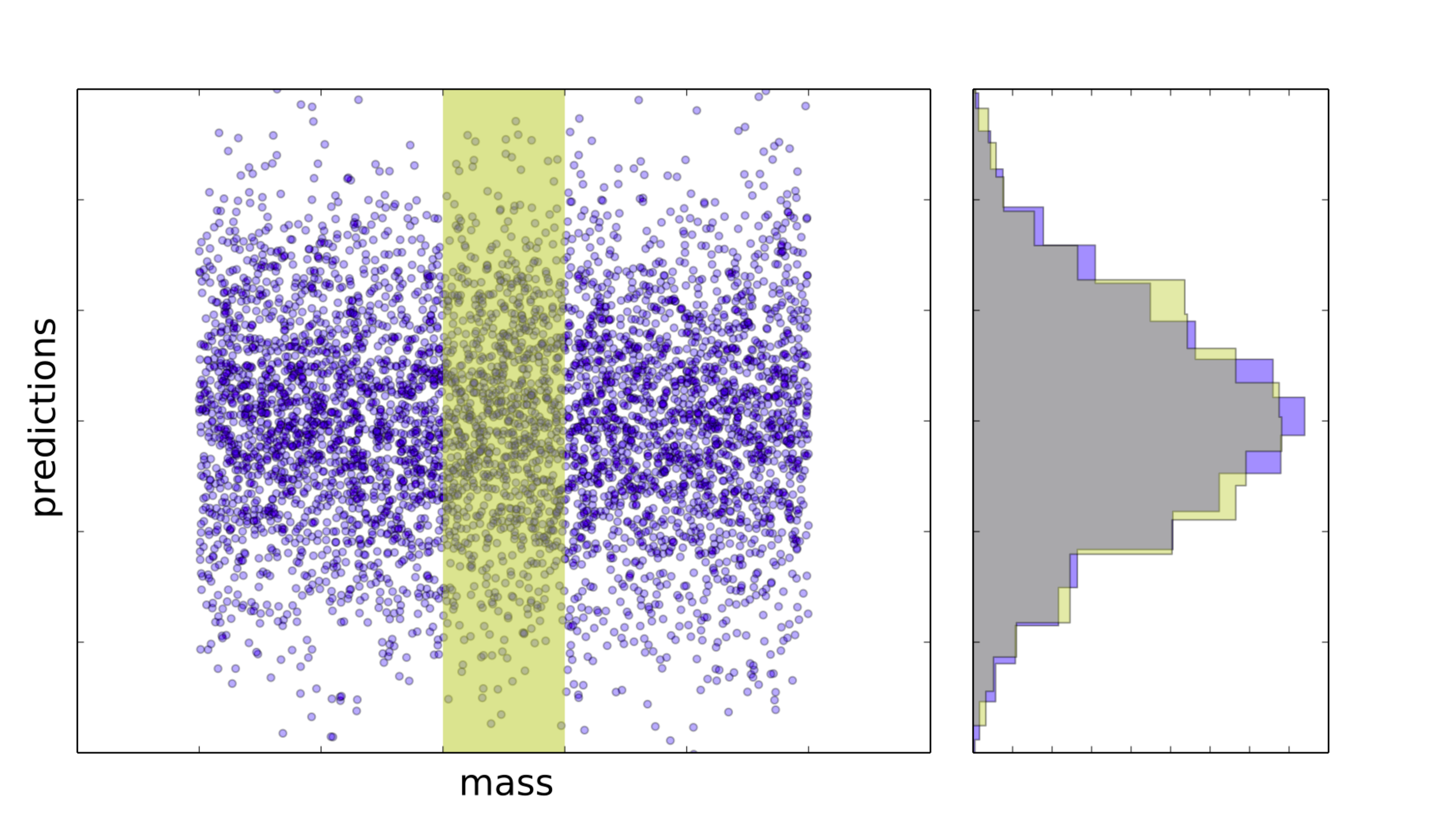
uBoost

uBoost is an ensemble over uBoostBDT, each uBoostBDT uses own global FPR.

uBoostBDT returns 0 or 1 (passed or not the threshold corresponding to target FPR), simple voting is used to obtain predictions.

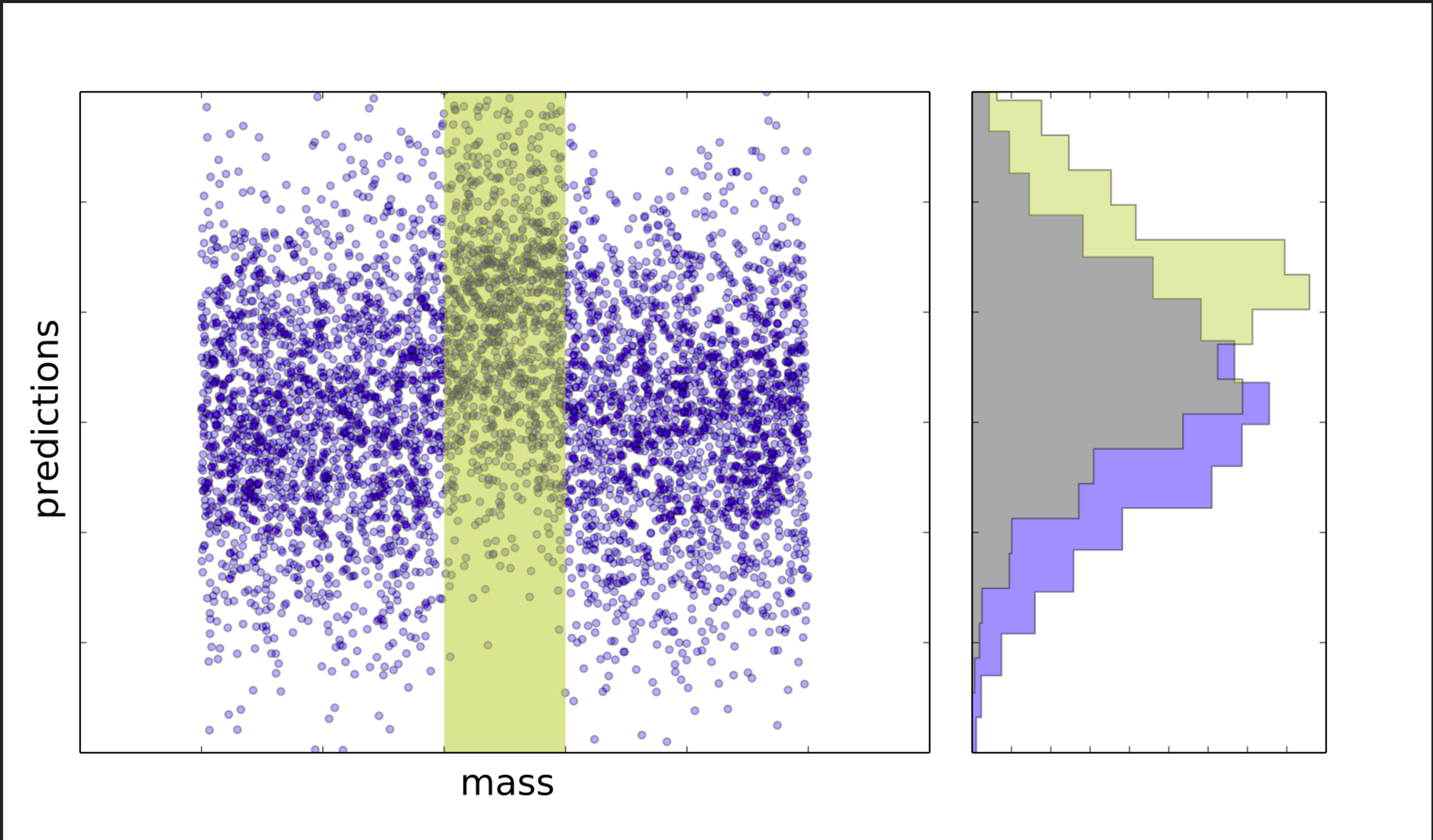
- drives to uniform selection
- very complex training
- many classifiers
- estimation of threshold in uBoostBDT may be biased

MEASURING NON-UNIFORMITY



MEASURING NON-UNIFORMITY

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



FLATNESS LOSS

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

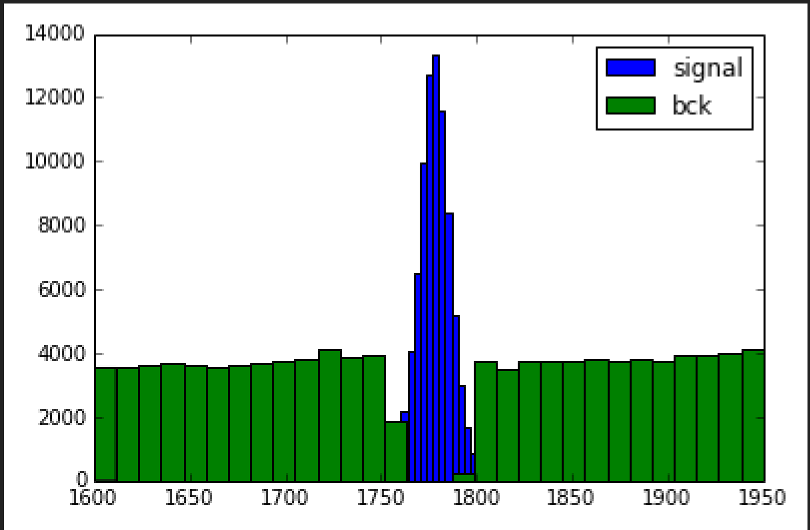
$$\mathcal{L} = \mathcal{L}_{\text{exploss}} + c\mathcal{L}_{\text{FL}}$$

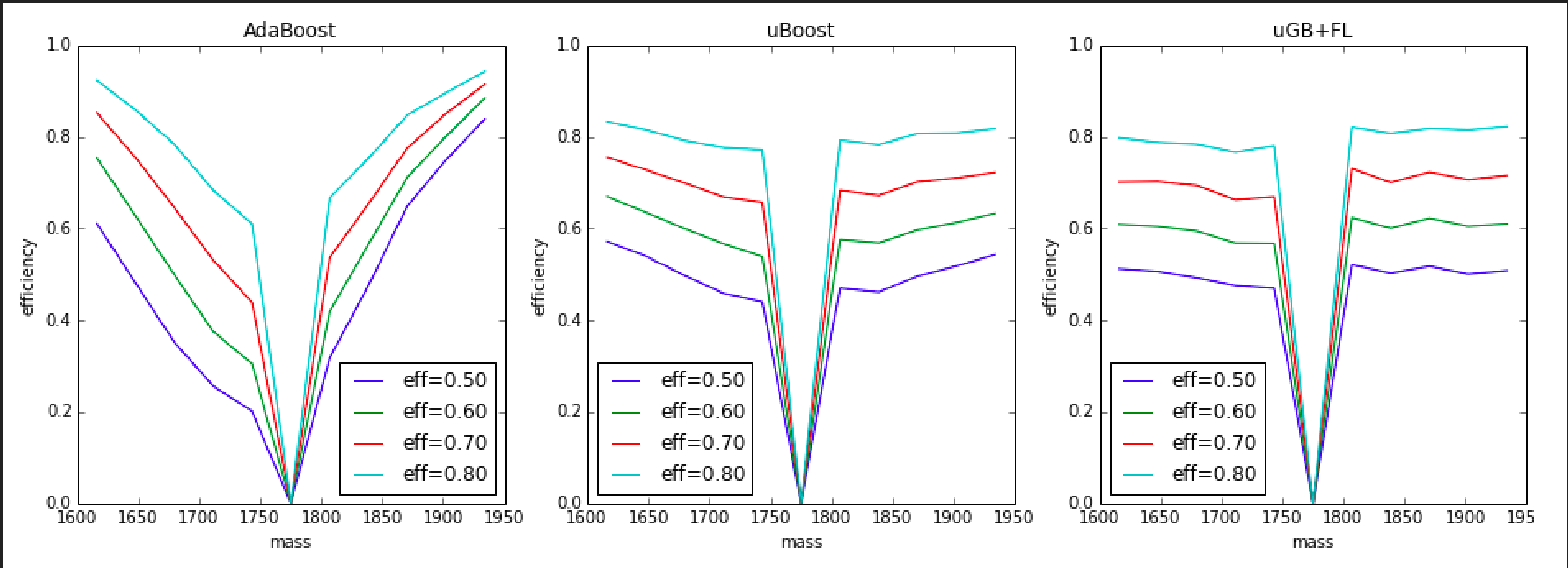
Flatness loss approximates (non-differentiable) CvM metrics:

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

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

EXAMPLE (EFFICIENCY OVER BACKGROUND)





GRADIENT BOOSTING

- general-purpose flexible algorithm
- usually over trees
- state-of-art results in many areas
- can overfit
- usually needs tuning

THE END