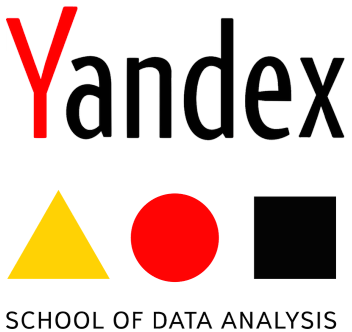
Machine Learning in Science and Industry

Day 2, lectures 3 & 4

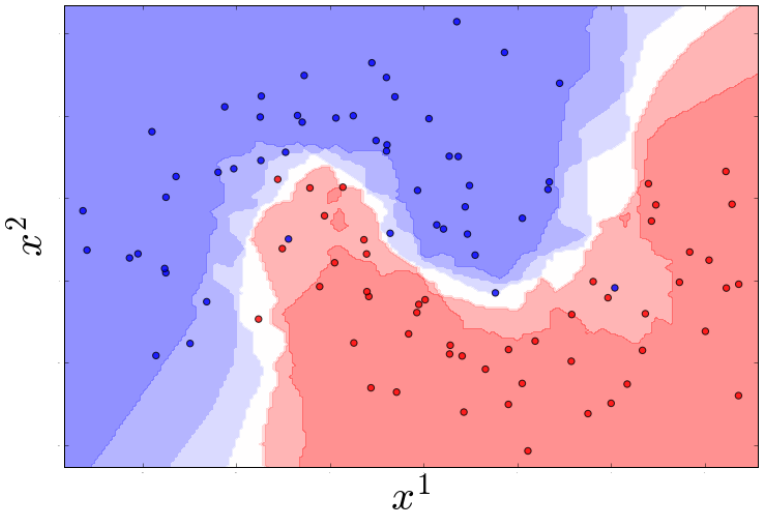
Alex Rogozhnikov, Tatiana Likhomanenko

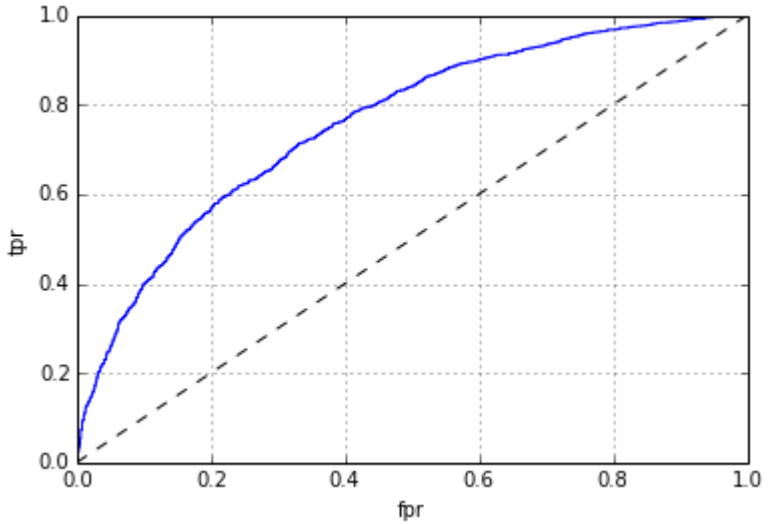
Heidelberg, GradDays 2017



Recapitulation

- Statistical ML: applications and problems
- k nearest neighbours classifier and regressor
- Binary classification: ROC curve, ROC AUC



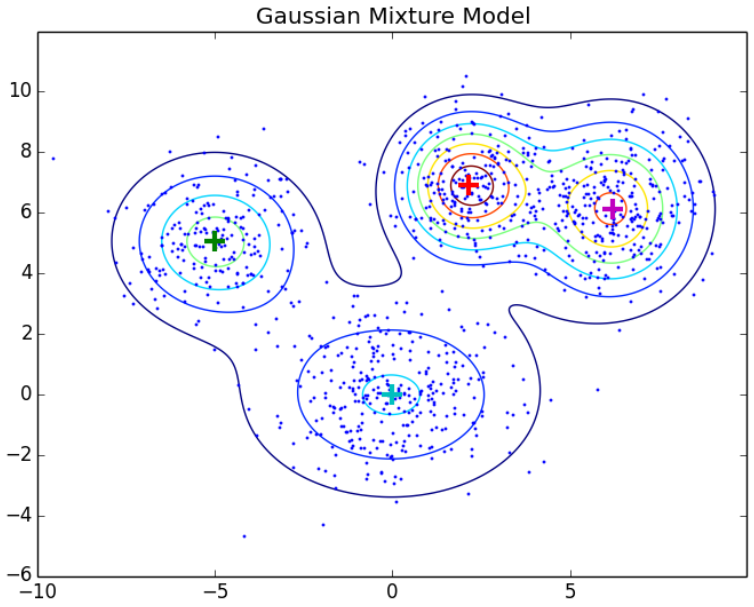


Recapitulation

- Optimal regression and optimal classification
- Bayes optimal classifier

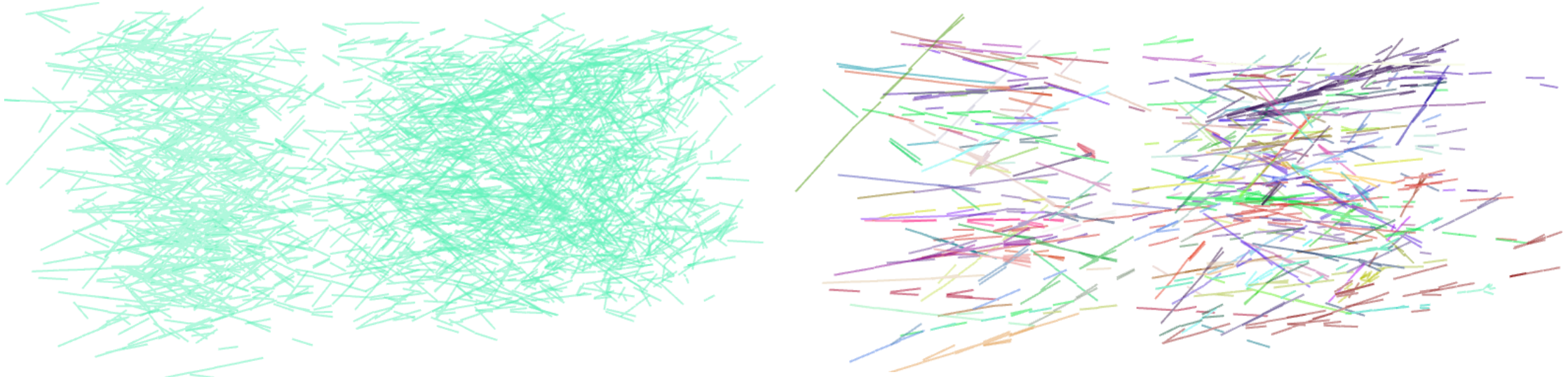
$$\frac{p(y=1 \mid x)}{p(y=0 \mid x)} = \frac{p(y=1)\, p(x \mid y=1)}{p(y=0)\, p(x \mid y=0)}$$

- Generative approach: density estimation, QDA, Gaussian mixtures



Recapitulation

- Clustering — unsupervised technique

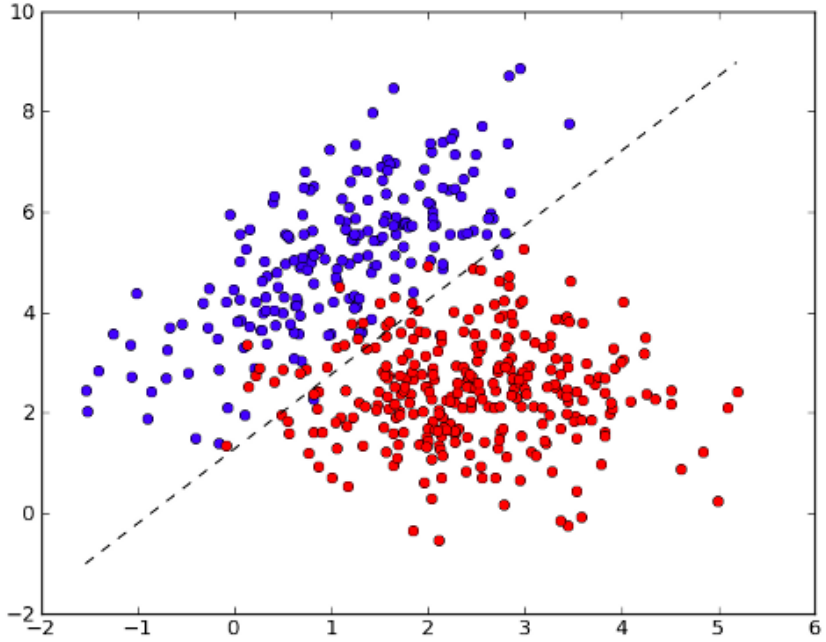


Recapitulation

- Discriminative approach: logistic regression (classification model), linear regression

$$p(y=+1|x) = p_{+1}(x) = \sigma(d(x))$$
$$p(y=-1|x) = p_{-1}(x) = \sigma(-d(x))$$

$$\text{with } d(x) =< w, x > +w_0$$



Recapitulation

- Regularization

$$\mathcal{L} = \frac{1}{N} \sum_i L(x_i, y_i) + \mathcal{L}_{\text{reg}} \to \min$$

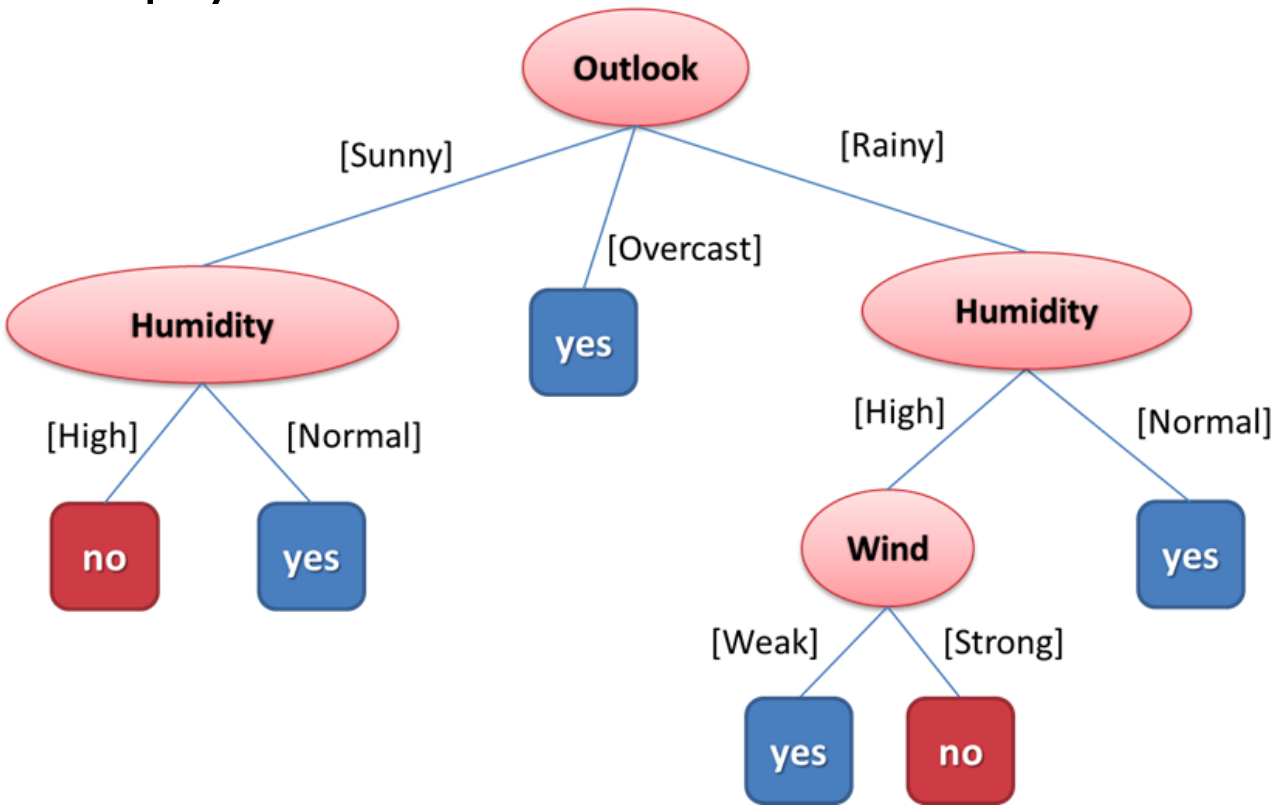
$$\mathcal{L}_p = \sum_i |w_i|^p$$

- Today: continue discriminative approach
  - reconstruct $p(y = 1 \mid x)$, $p(y = 0 \mid x)$
  - ignore $p(x, y)$

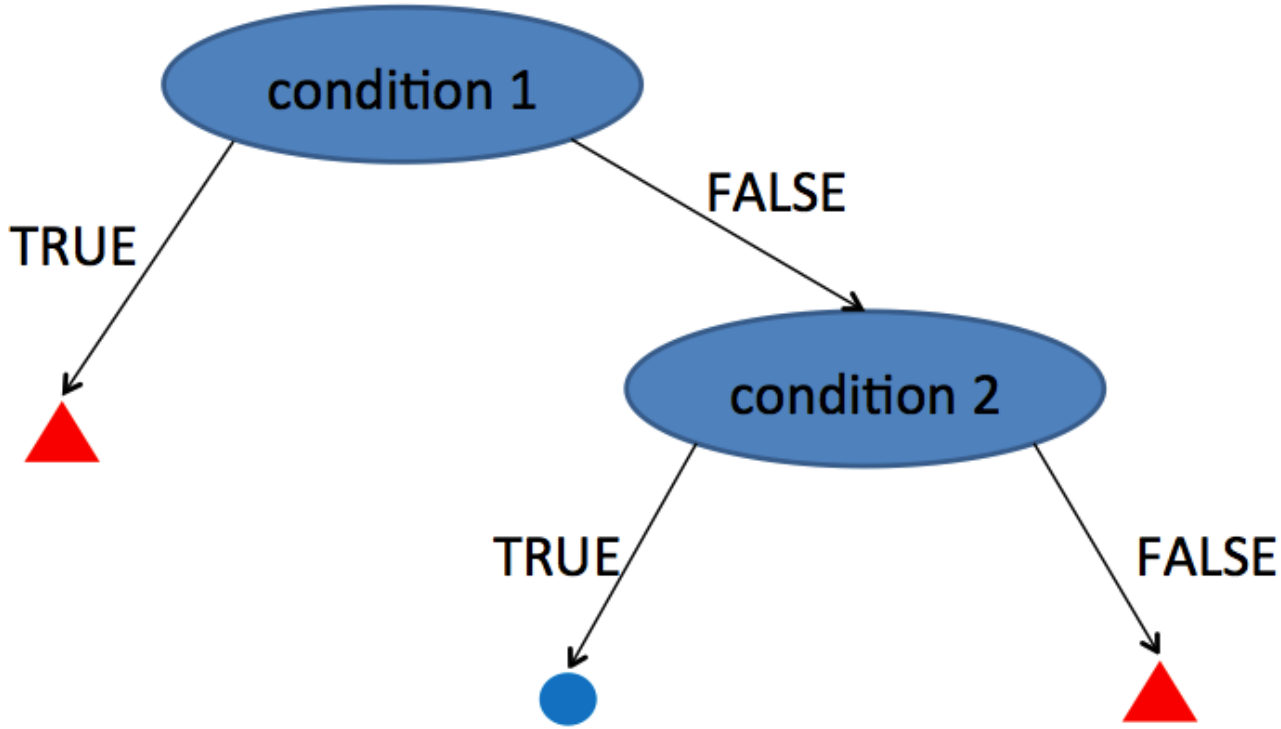
Decision Trees

Decision tree

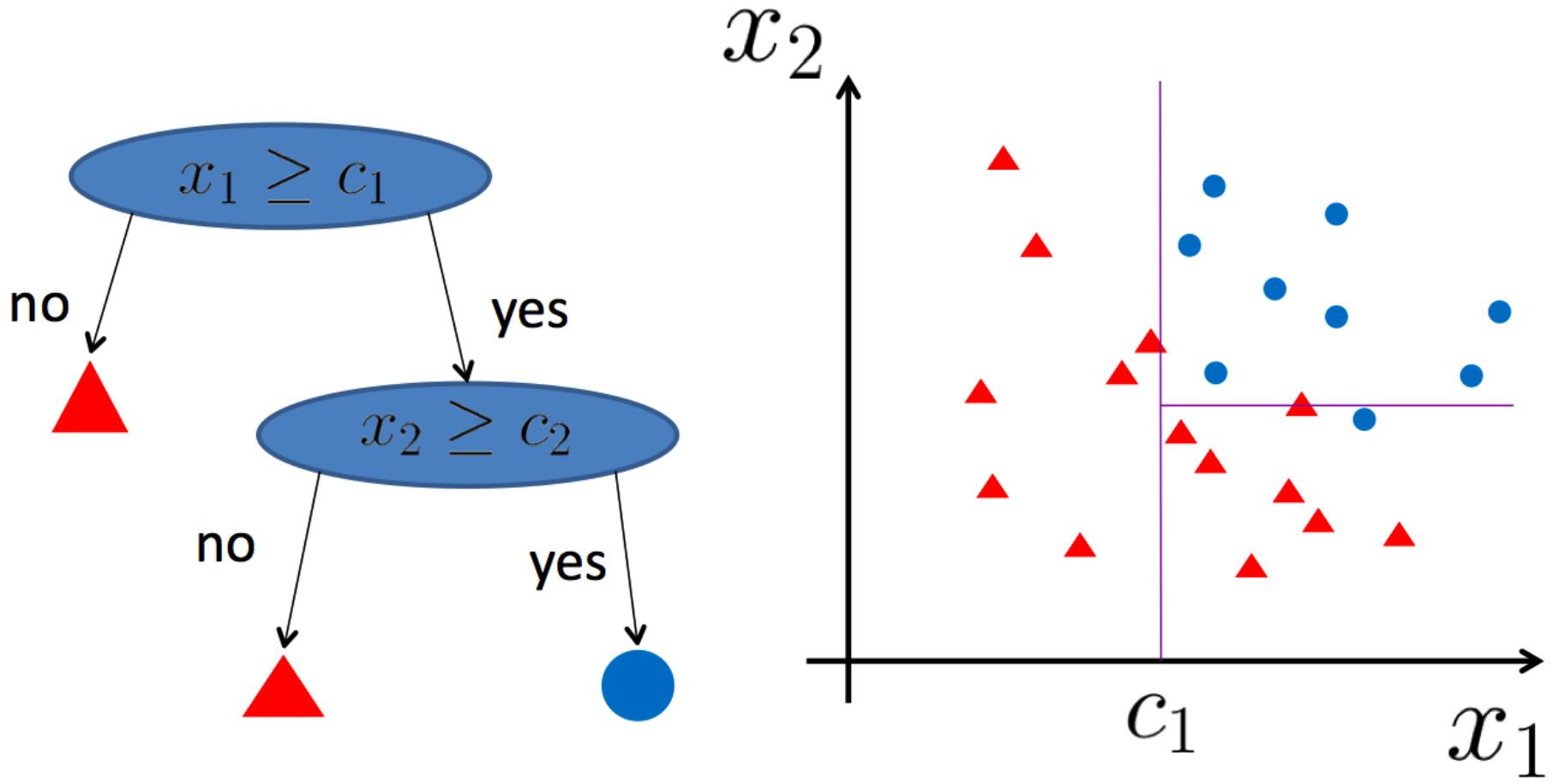
Example: predict outside play based on weather conditions.



Decision tree: binary tree



Decision tree: splitting space



Decision tree

- fast & intuitive prediction
- but building an optimal decision tree is an NP complete problem

Decision tree

- fast & intuitive prediction
- but building an optimal decision tree is an NP complete problem
- building a tree using a greedy optimization
  - start from the root (a tree with only one leaf)
  - each time split one leaf into two
  - repeat process for children if needed

Decision tree

- fast & intuitive prediction
- but building an optimal decision tree is an NP complete problem
- building a tree using a greedy optimization
  - start from the root (a tree with only one leaf)
  - each time split one leaf into two
  - repeat process for children if needed
- need a criterion to select best splitting (feature and threshold)

Splitting criterion

$$\text{TreeImpurity} = \sum_{\text{leaf}} \text{impurity}(\text{leaf}) \times \text{size}(\text{leaf})$$

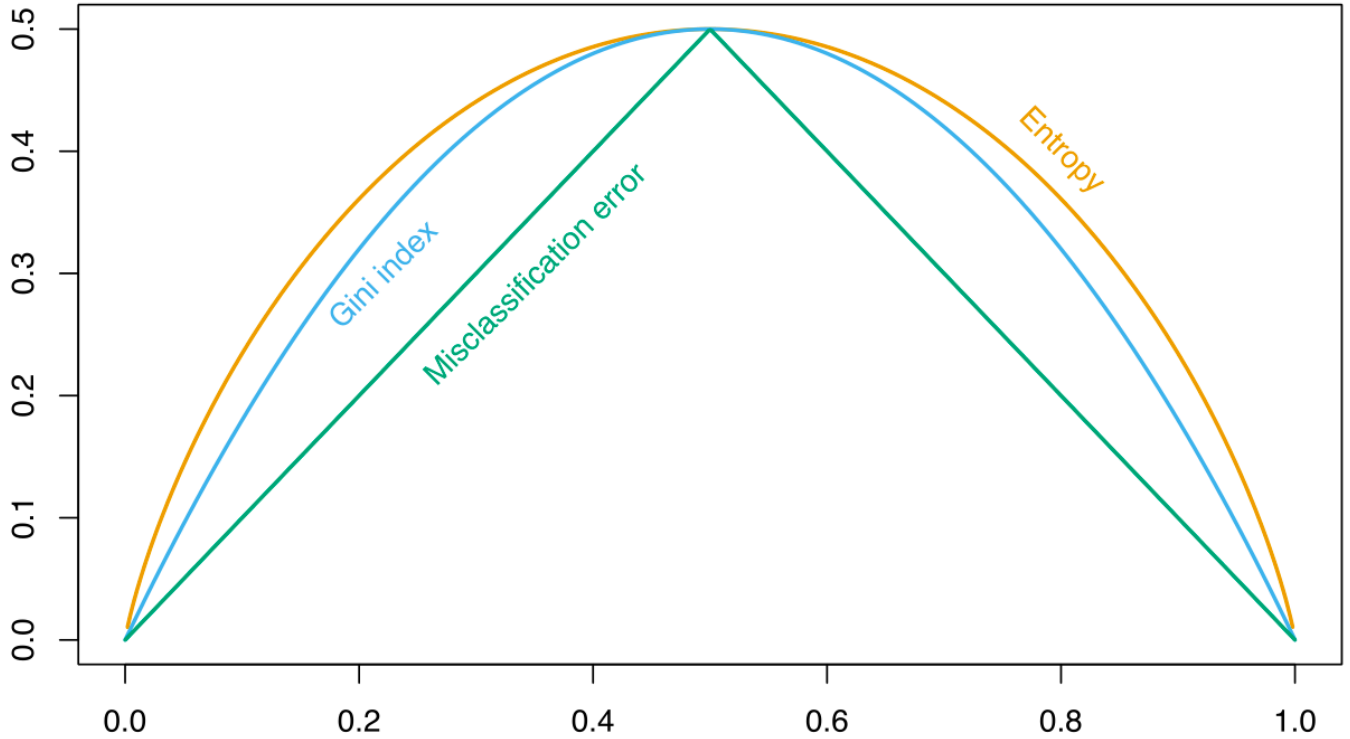
Several impurity functions:

$$\begin{aligned} \text{Misclassification} &= \min(p, 1-p) \\ \text{Gini Index} &= p(1-p) \\ \text{Entropy} &= -p \log p - (1-p) \log(1-p) \end{aligned}$$

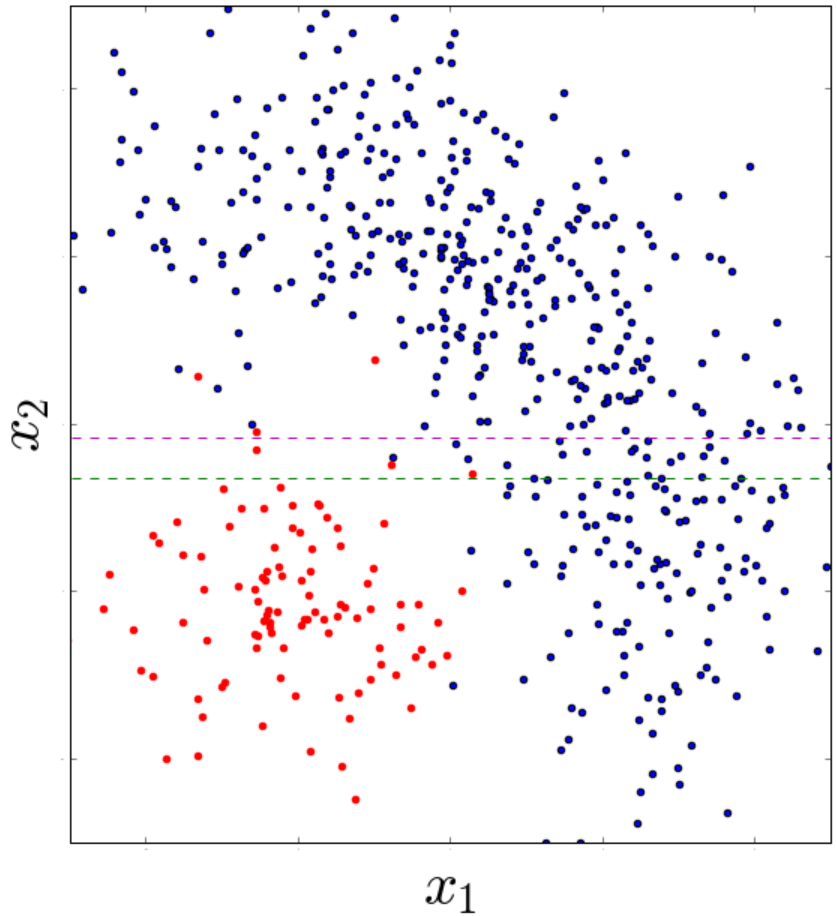
where p is a portion of signal observations in a leaf, and $1 - p$ is a portion of background observations, $\text{size}(\text{leaf})$ is number of training observations in a leaf.

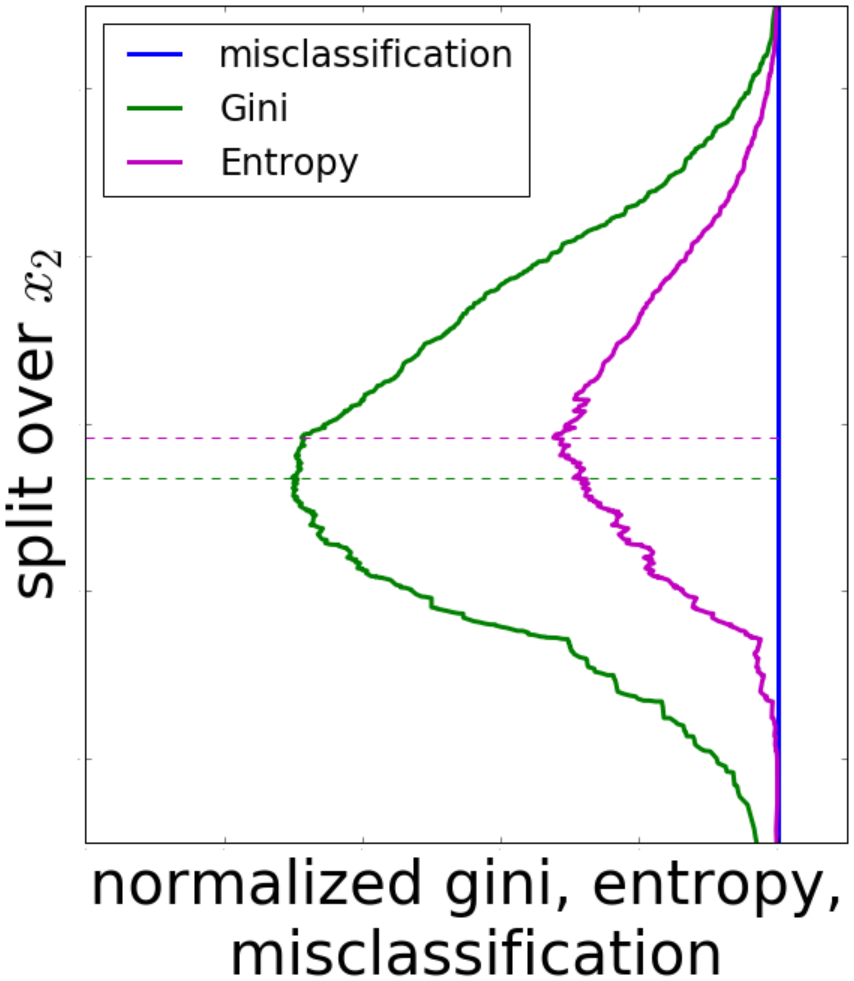
Splitting criterion

Impurity as a function of p



Splitting criterion: why not misclassification?





Decision trees for regression

Greedy optimization (minimizing MSE):

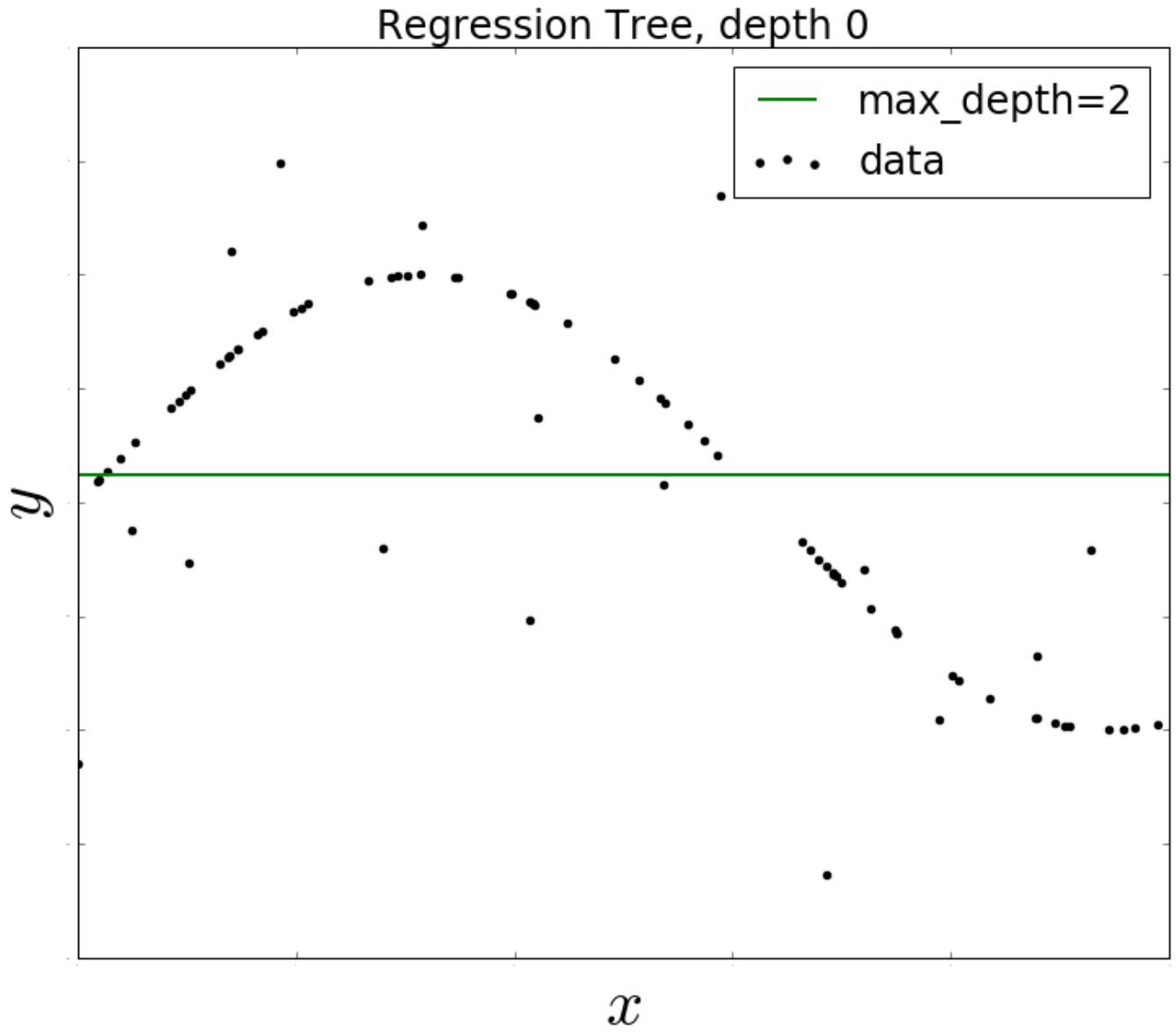
$$\text{TreeMSE} \sim \sum_i (y_i - \hat{y}_i)^2$$

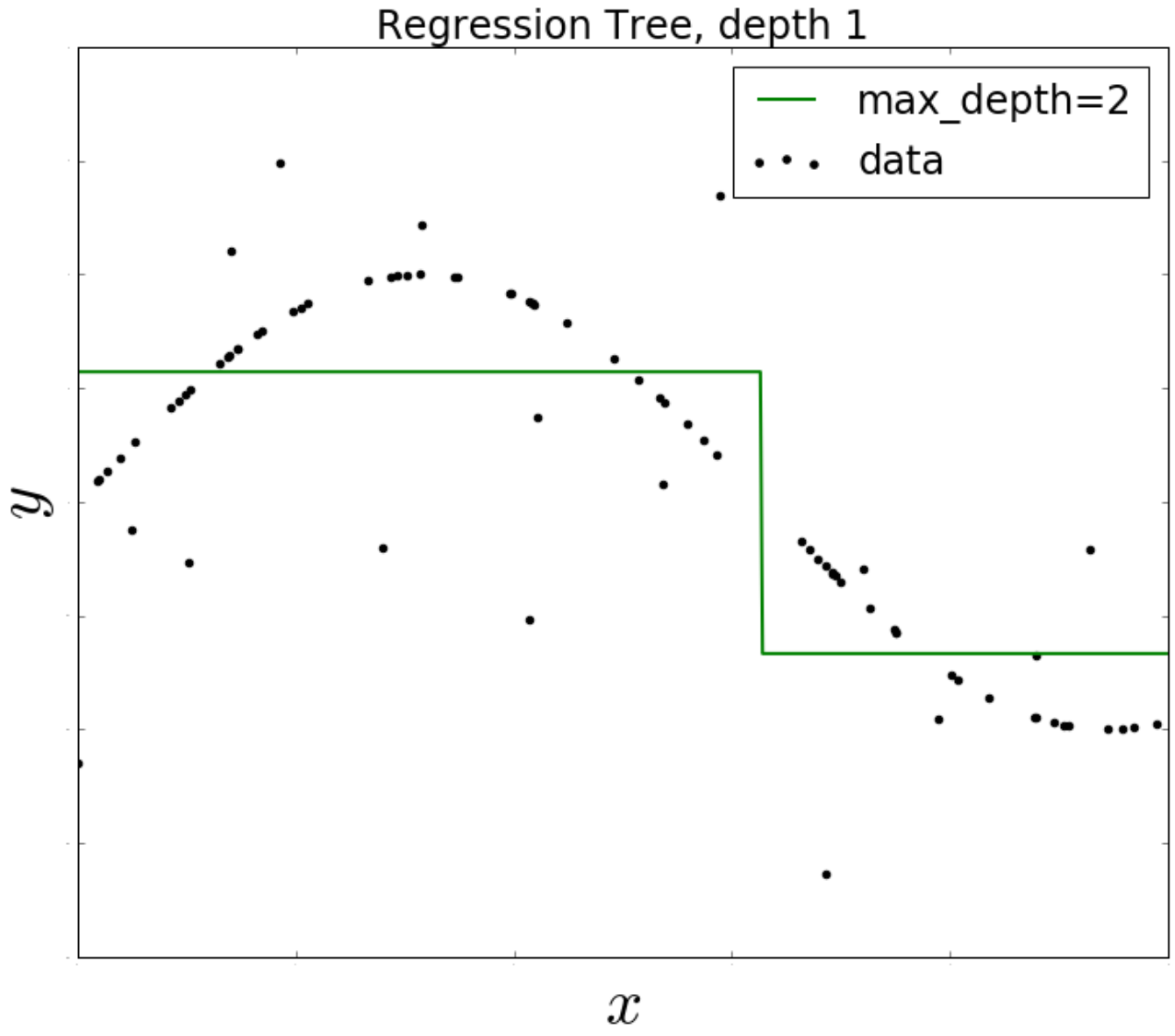
Can be rewritten as:

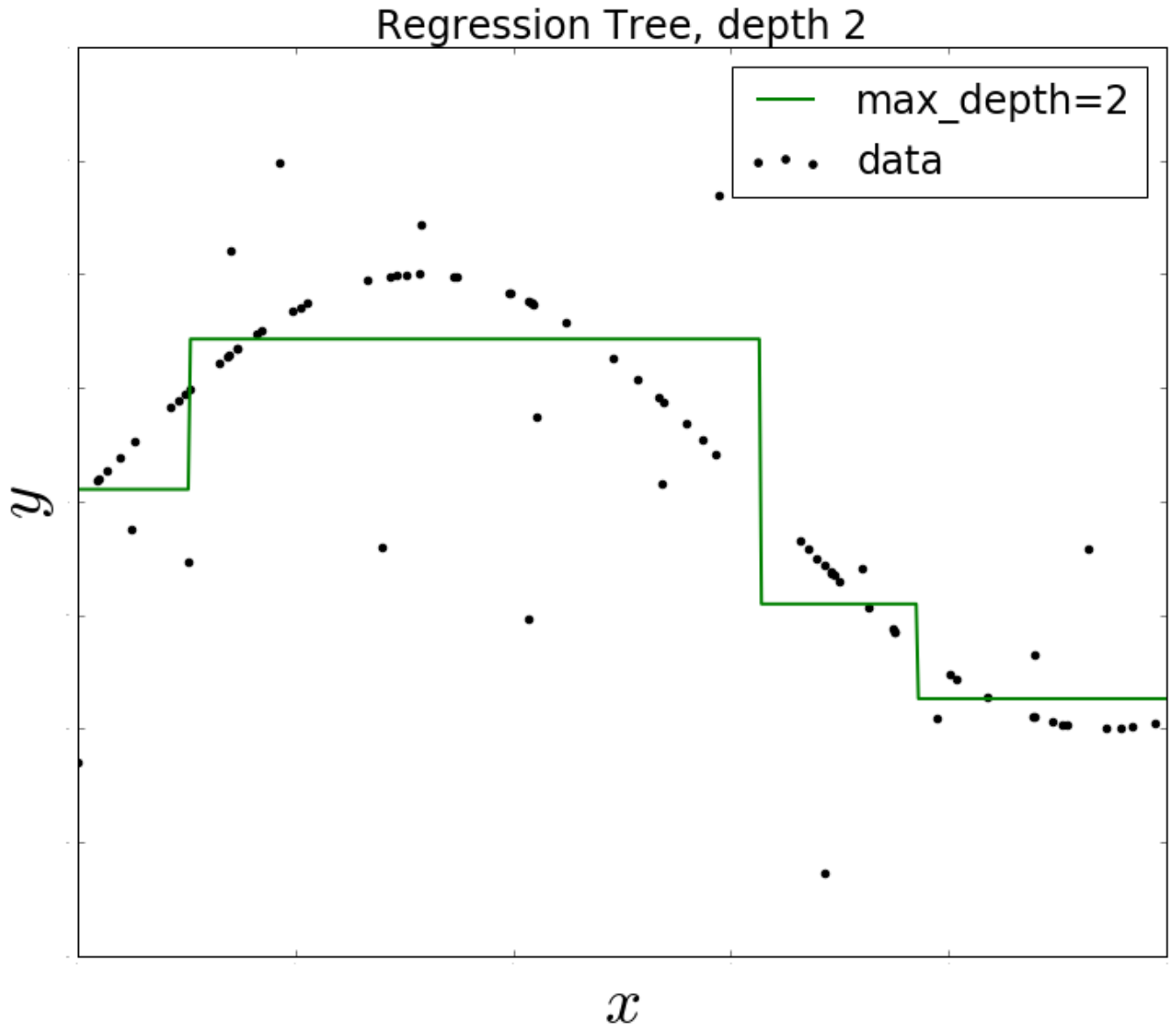
$$\text{TreeMSE} \sim \sum_{\text{leaf}} \text{MSE}(\text{leaf}) \times \text{size}(\text{leaf})$$

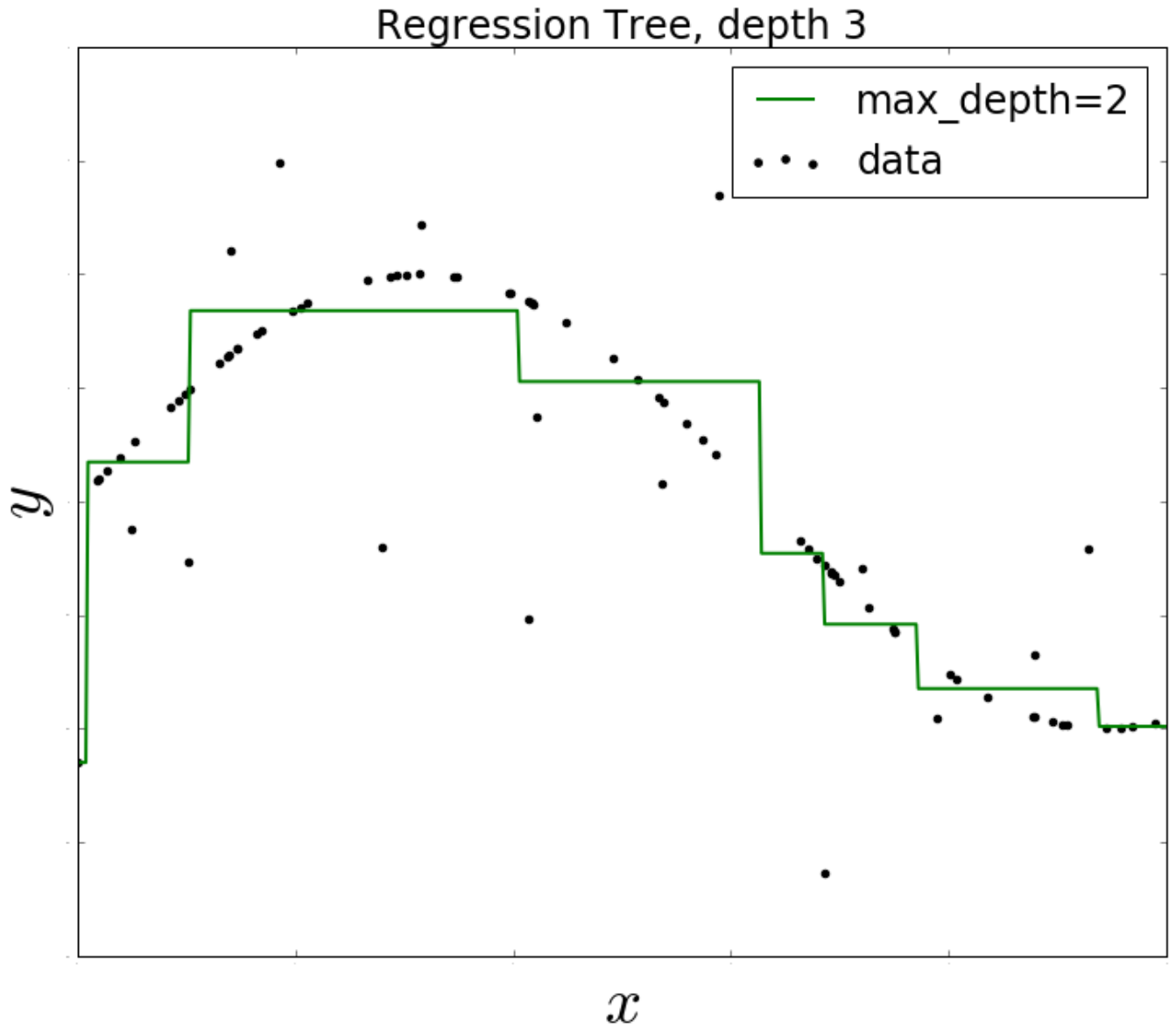
$\text{MSE}(\text{leaf})$ is like an 'impurity' of the leaf:

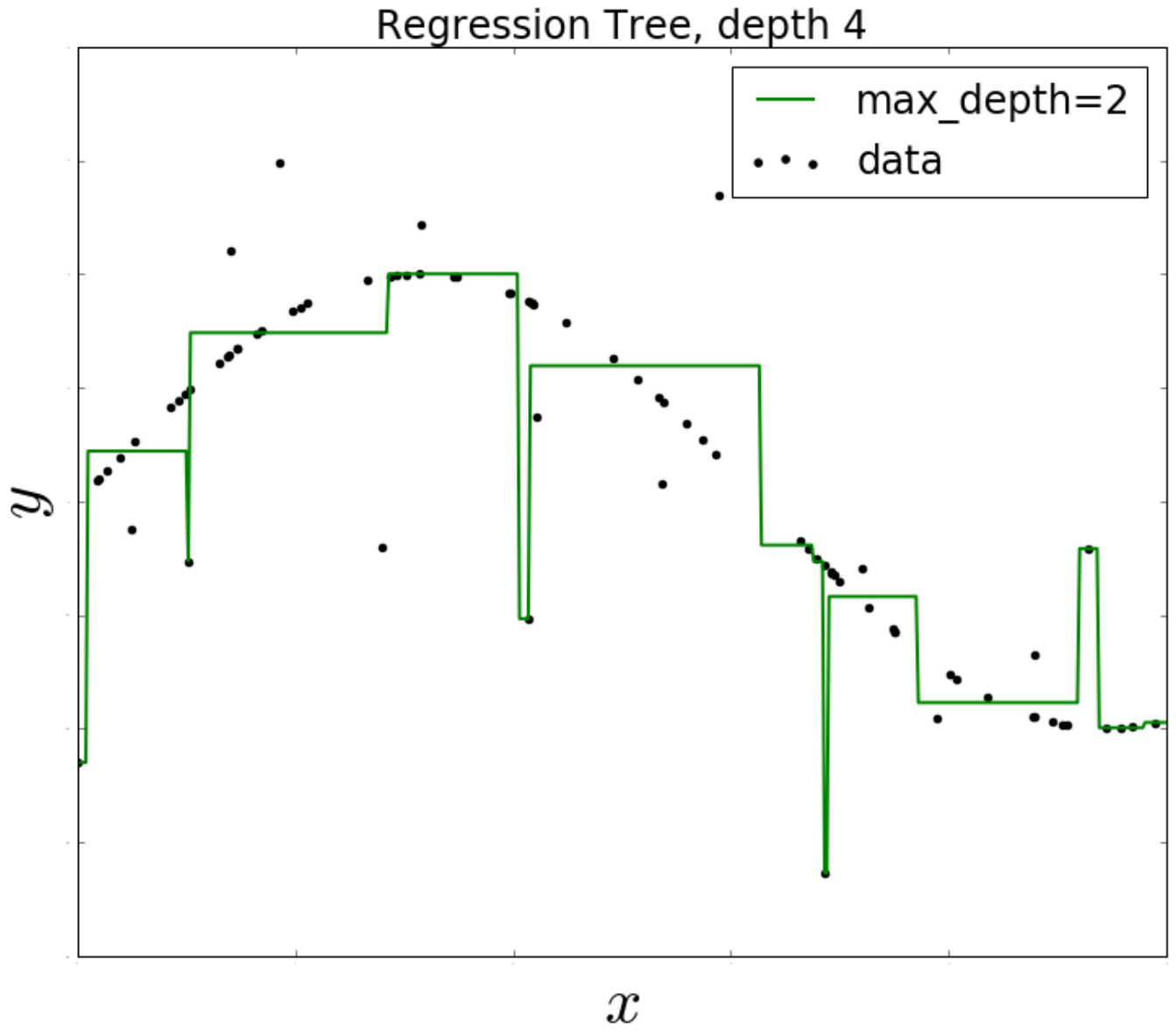
$$\text{MSE}(\text{leaf}) = \frac{1}{\text{size}(\text{leaf})} \sum_{i \in \text{leaf}} (y_i - \hat{y}_i)^2$$


Regression Tree, depth 0
max_depth=2
data
y
x


Regression Tree, depth 1
max_depth=2
data
y
x

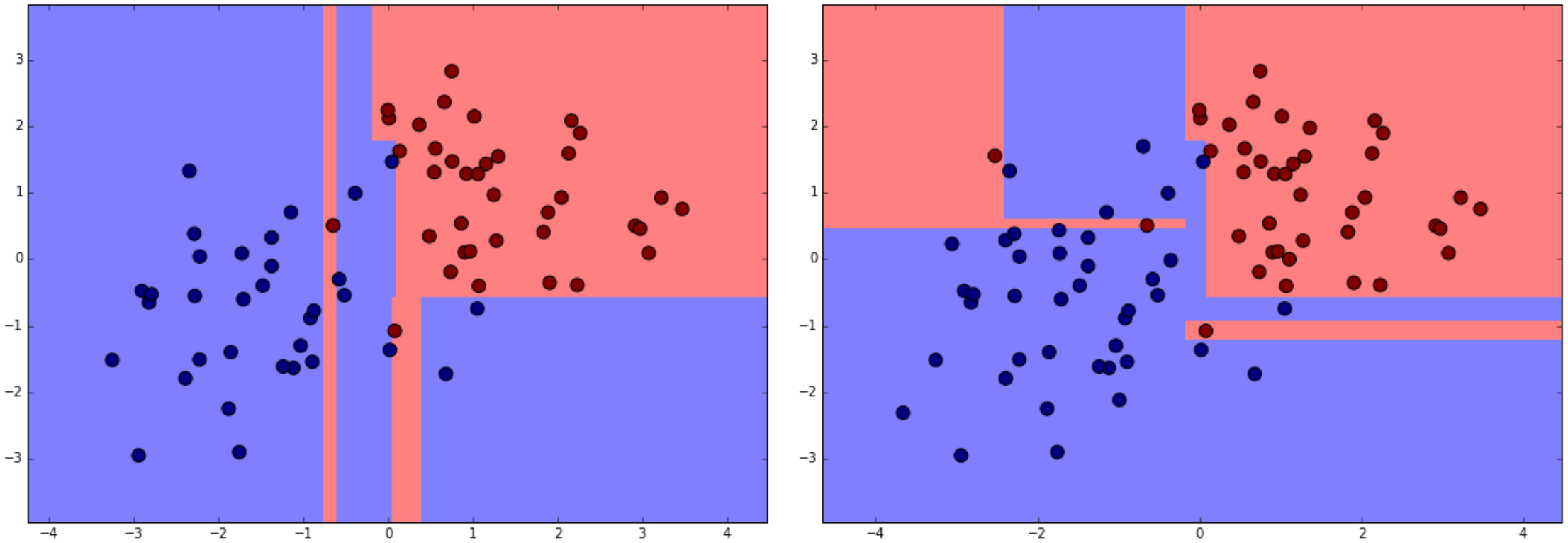

Regression Tree, depth 2
max_depth=2
data
y
x


Regression Tree, depth 3
max_depth=2
data
y
x

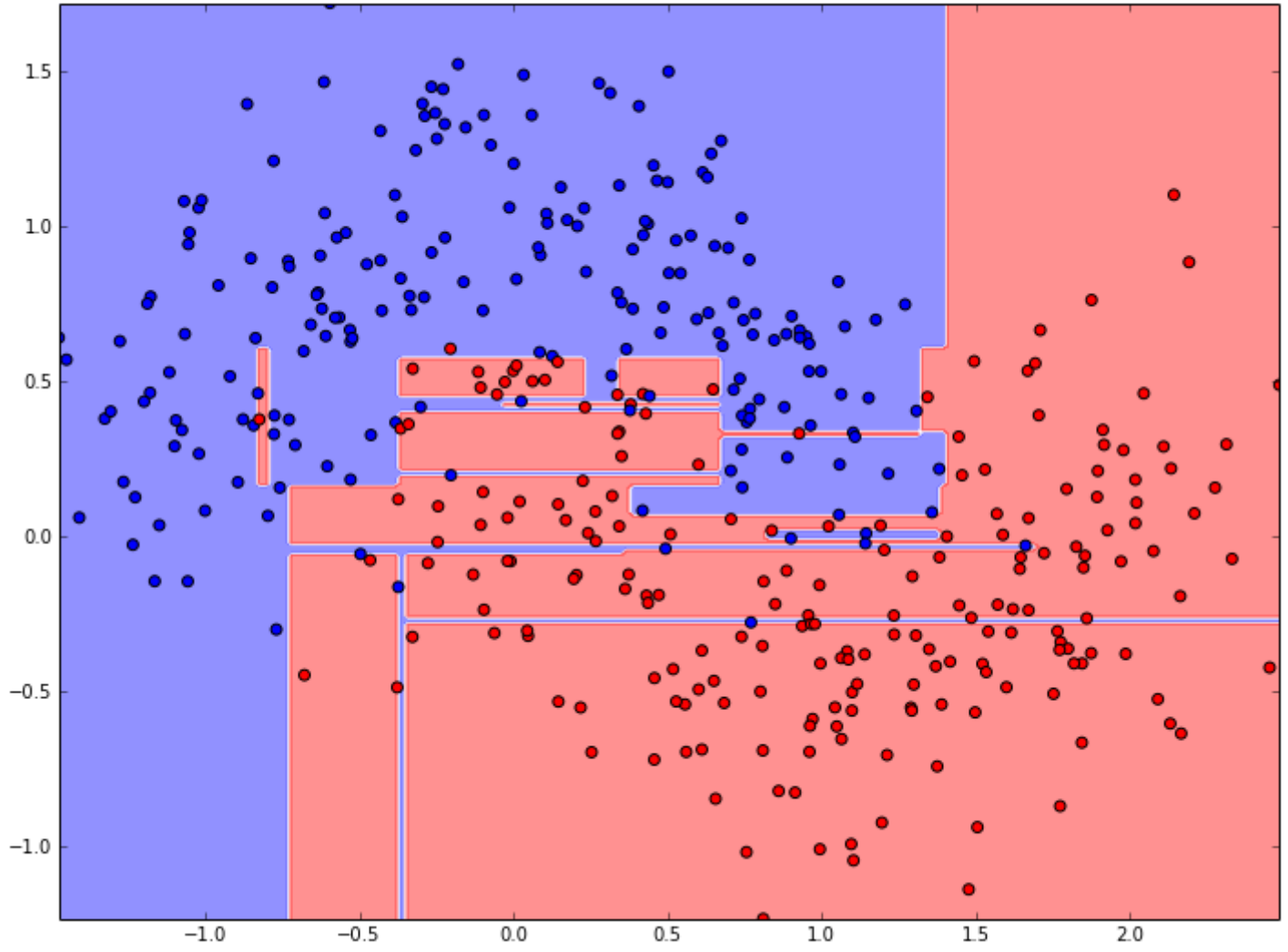

Regression Tree, depth 4
max_depth=2
data
y
x

Decision trees instability

Little variation in training dataset produce different classification rule.



Tree keeps splitting until each observation is correctly classified:



Pre-stopping

We can stop the process of splitting by imposing different restrictions:

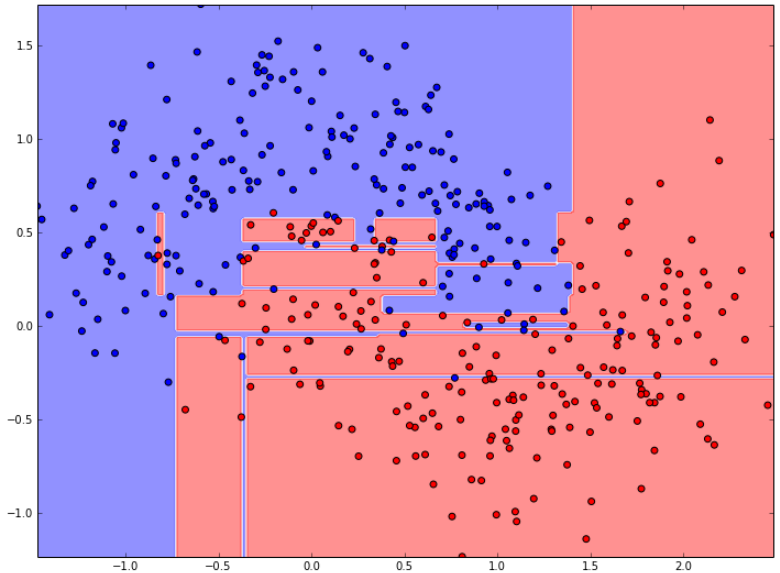
- limit the depth of tree
- set minimal number of samples needed to split the leaf
- limit the minimal number of samples in a leaf
- more advanced: maximal number of leaves in the tree

Pre-stopping

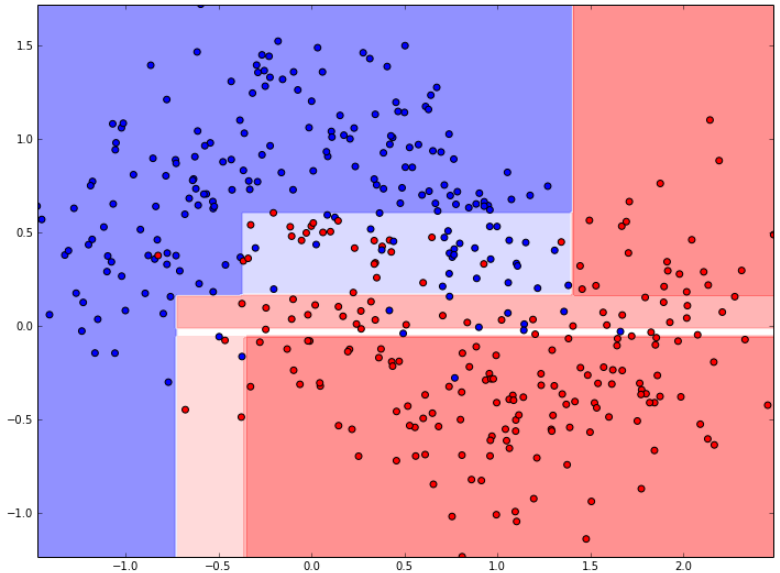
We can stop the process of splitting by imposing different restrictions:

- limit the depth of tree
- set minimal number of samples needed to split the leaf
- limit the minimal number of samples in a leaf
- more advanced: maximal number of leaves in the tree

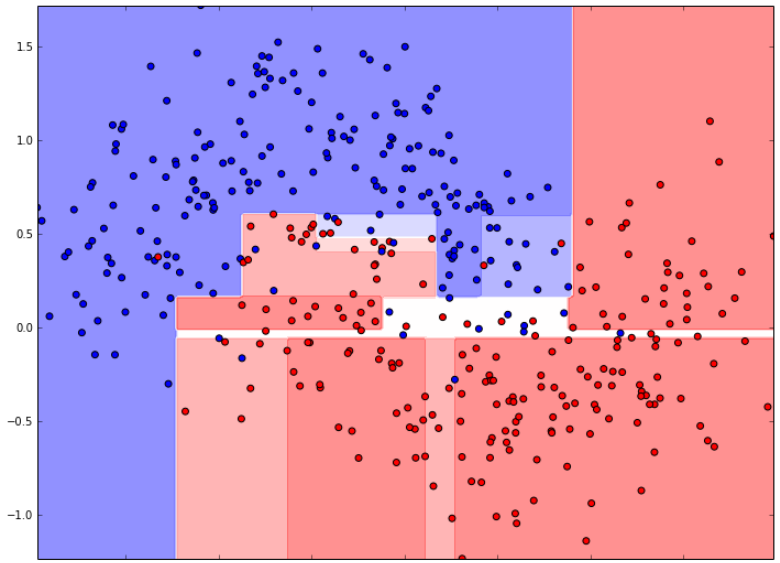
Any combinations of rules above is possible.



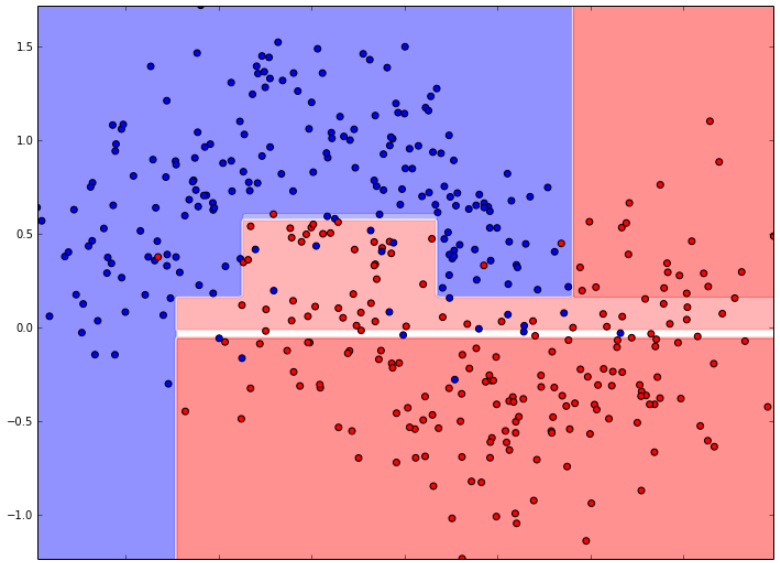
no pre-stopping



maximal depth



min # of samples in a leaf



maximal number of leaves

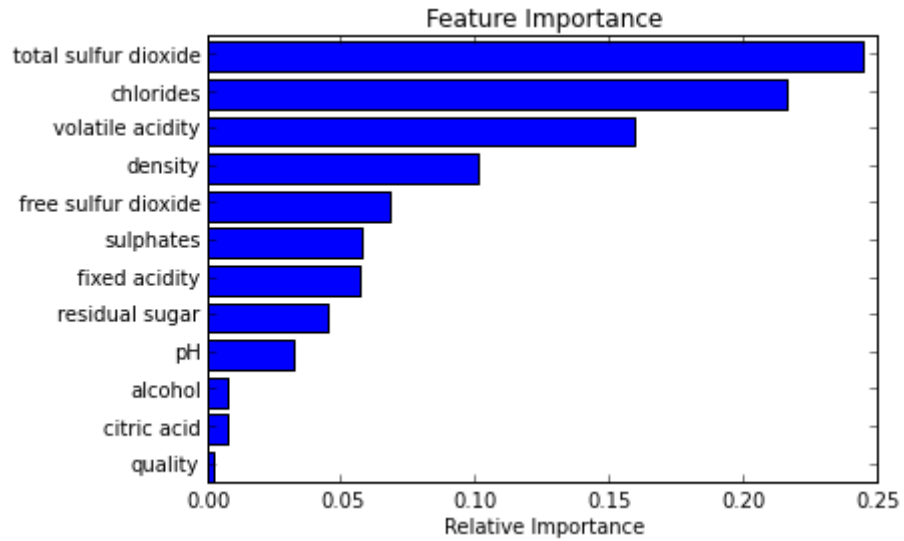
Decision tree overview

1. Very intuitive algorithm for regression and classification
2. Fast prediction
3. Scale-independent
4. Supports multiclassification

But

1. Training optimal tree is NP-complex
    - Trained greedily by optimizing Gini index or entropy (fast!)
2. Non-stable
3. Uses only trivial conditions

Feature importances



Different approaches exist to measure an importance of feature in the final model.

Importance of feature $\neq$ quality provided by one feature

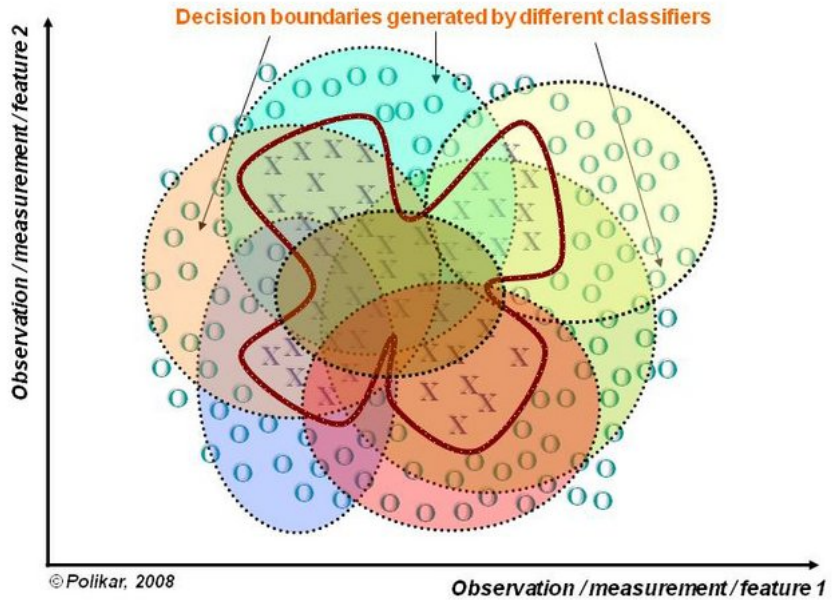
Computing feature importances

- tree: counting number of splits made over this feature
- tree: counting gain in purity (e.g. Gini)
  fast and adequate
- model-agnostic recipe: train without one feature,
  compare quality on test with/without one feature
  - requires many evaluations
- model-agnostic recipe: feature shuffling take one column in test dataset and shuffle it.
  Compare quality with/without shuffling.

Ensembles

Composition of models

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



Simple Voting

- Averaging predictions

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

- Averaging predicted probabilities

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

- Averaging decision functions

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

Weighted voting

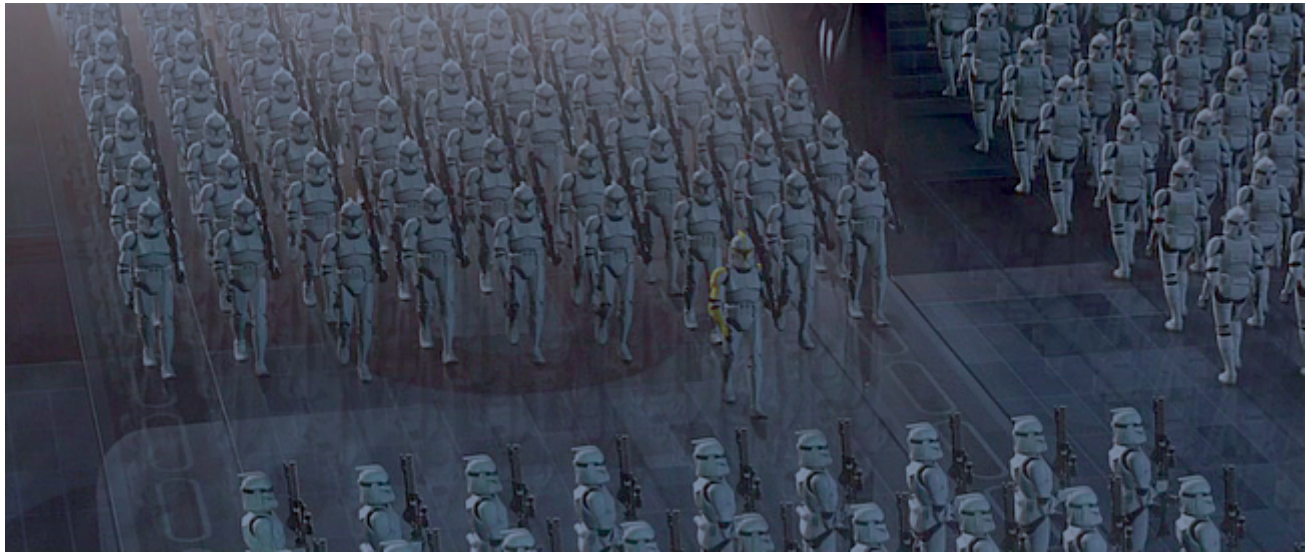
The way to introduce importance of classifiers

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

General case of ensembling

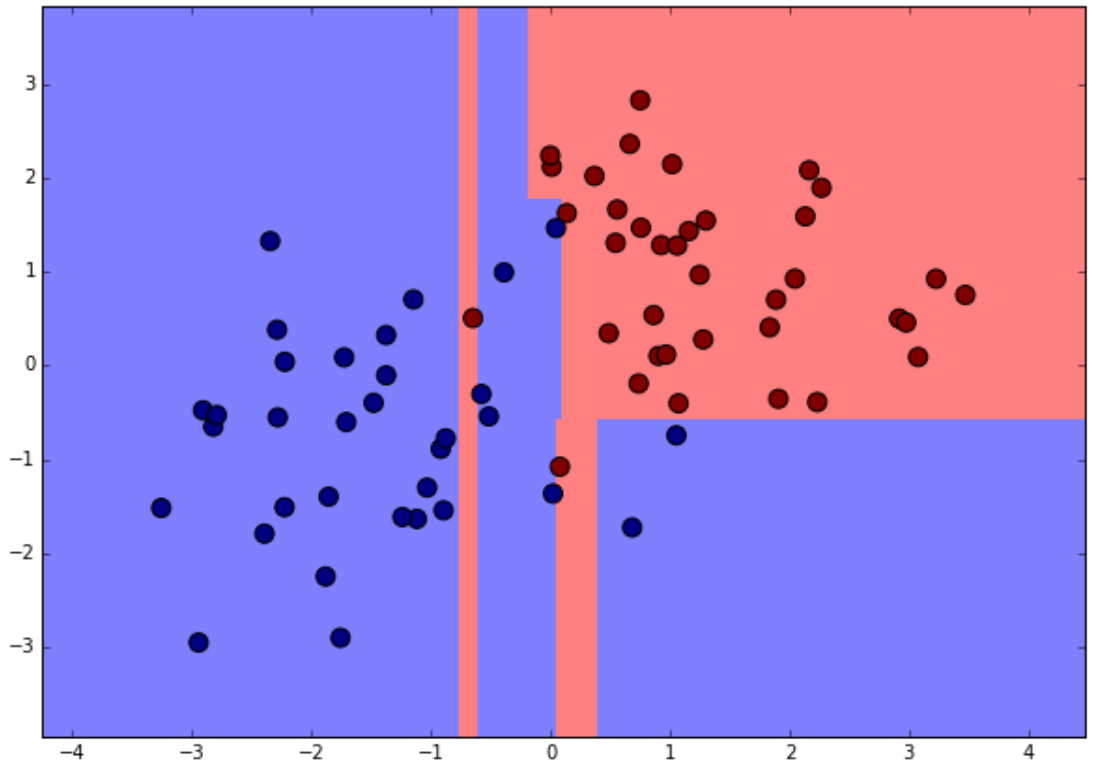
$D(x) = f(d_1(x), d_2(x), ..., d_J(x))$

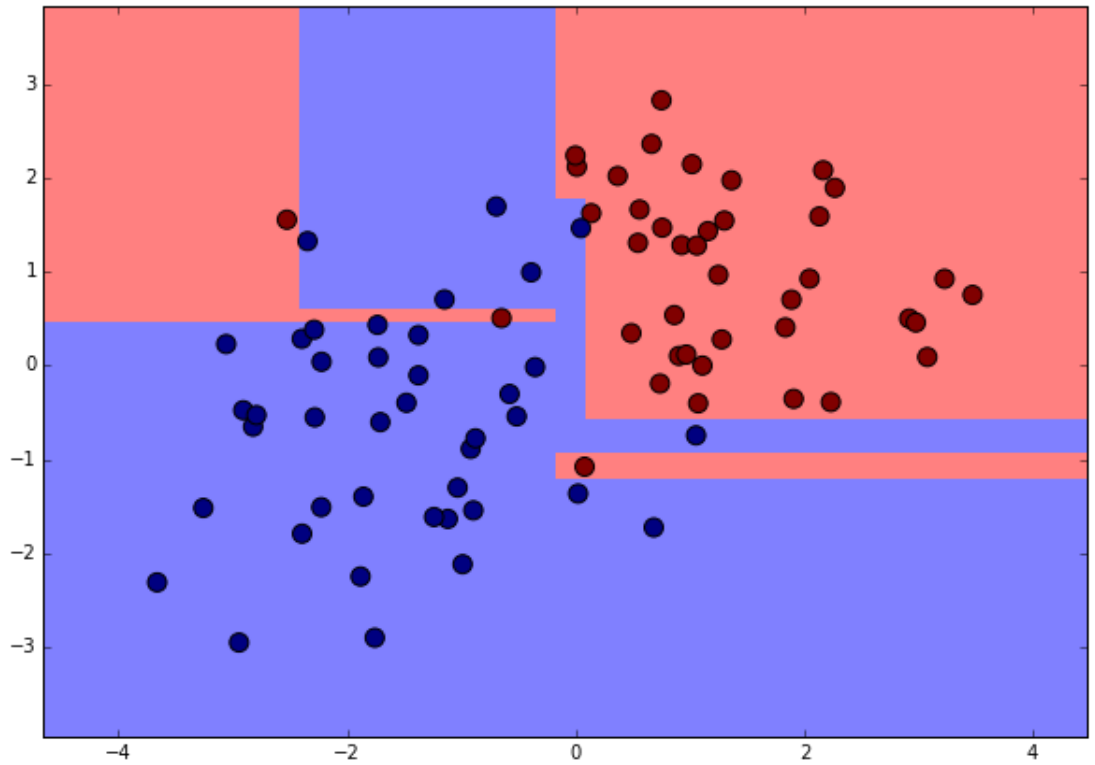
Problems



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

Decision tree reminder





Generating training subset

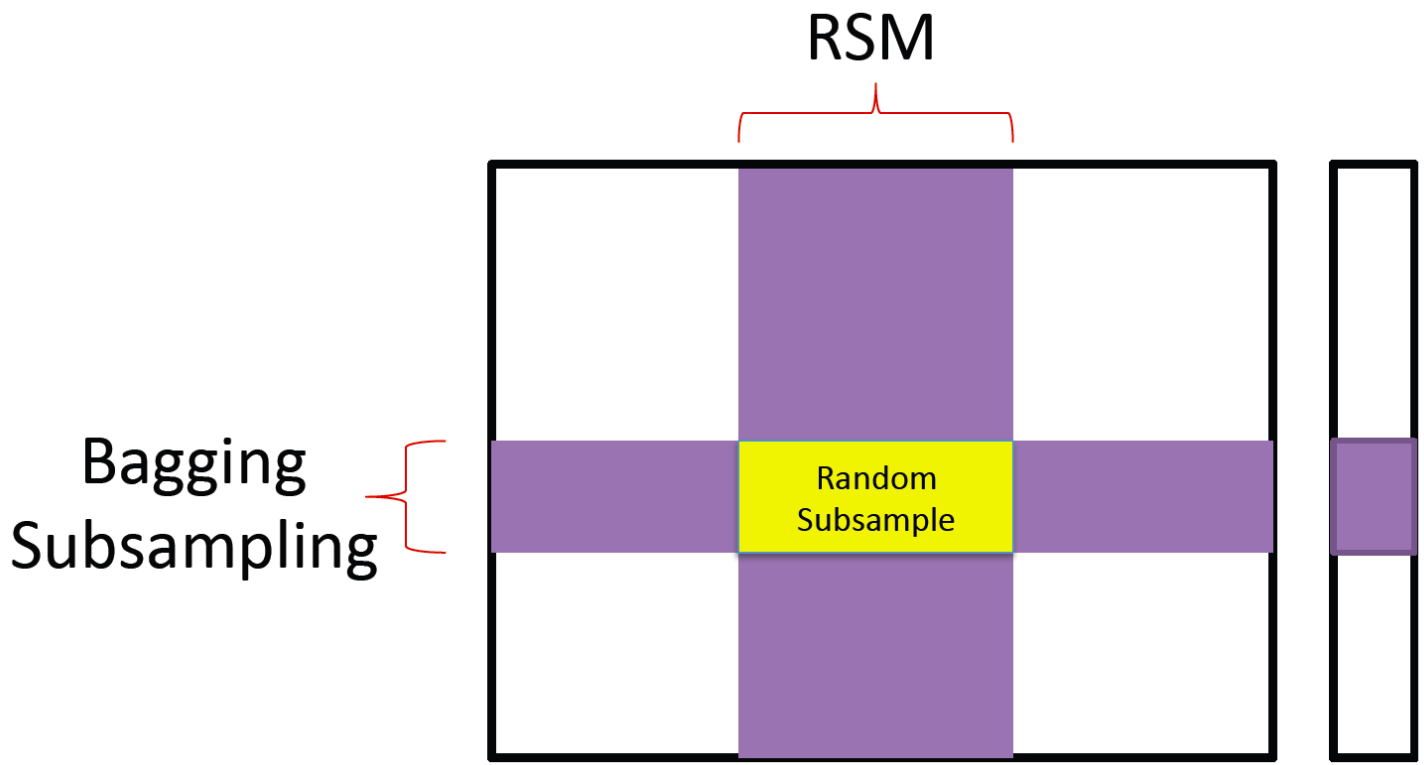
- *subsampling*
  taking fixed part of samples (sampling without replacement)

- *bagging* (Bootstrap AGGregating)
  sampling with replacement,

  If #generated samples = length of the 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



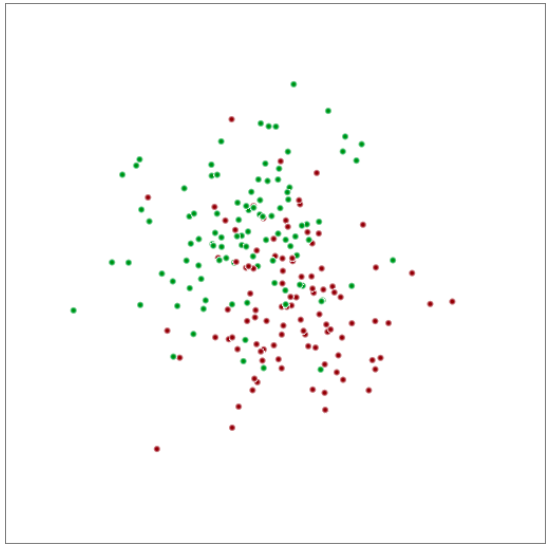
Random Forest [Leo Breiman, 2001]

Random forest is a composition of decision trees.

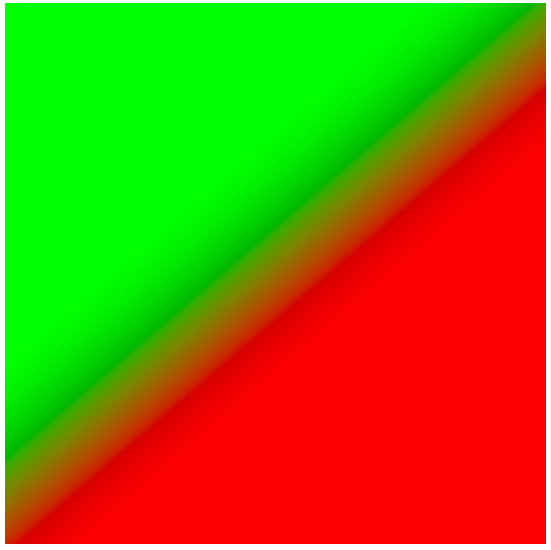
Each individual tree is trained on a subset of training data obtained by

- bagging samples
- taking random subset of features

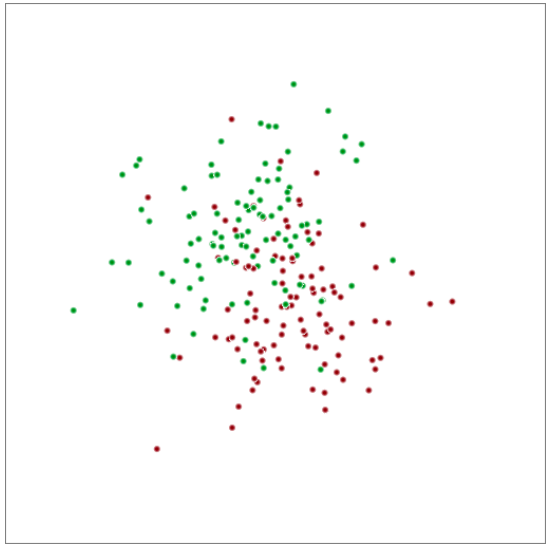
Predictions of random forest are obtained via simple voting.



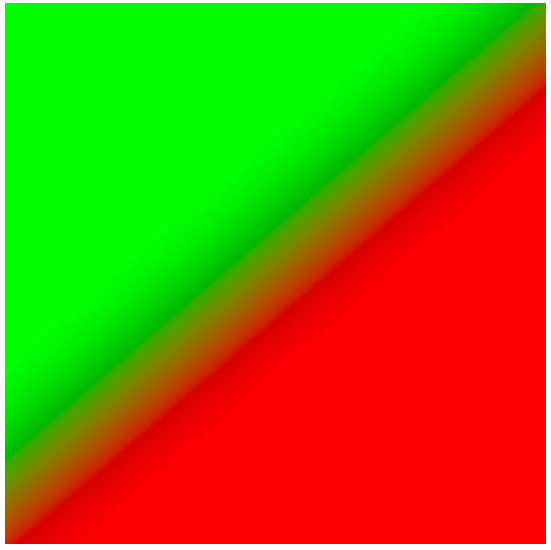
data



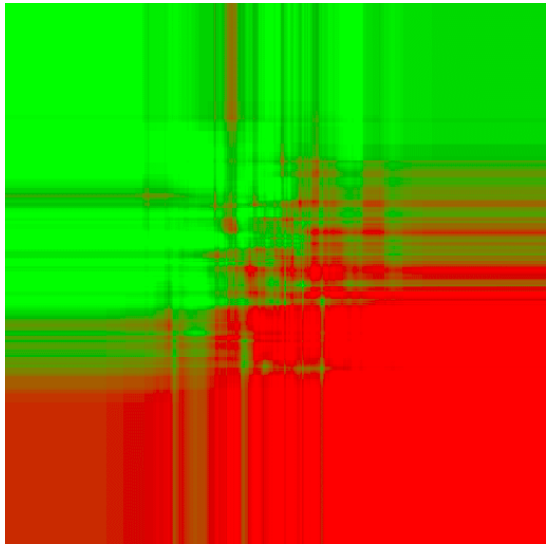
optimal boundary



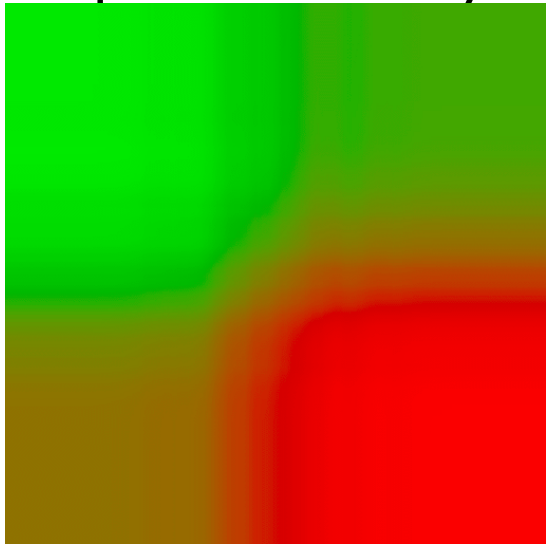
data



optimal boundary

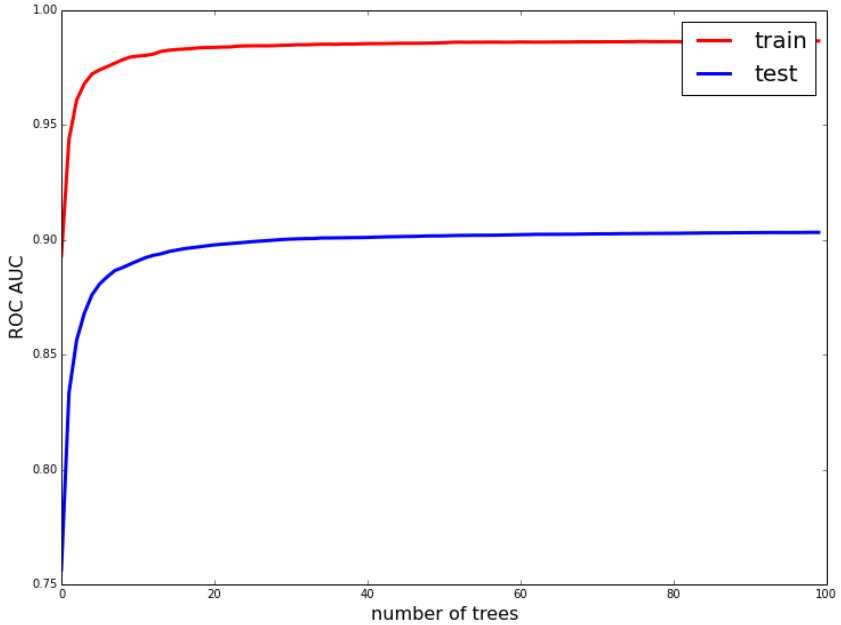


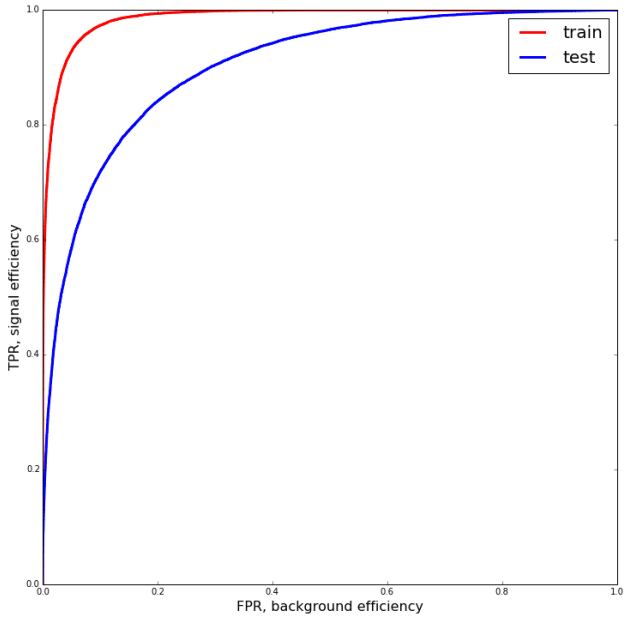
50 trees



2000 trees

Overfitting





- doesn't overfit: increasing complexity (adding more trees) doesn't spoil a classifier

- Works with features of different nature
- Stable to outliers in data.

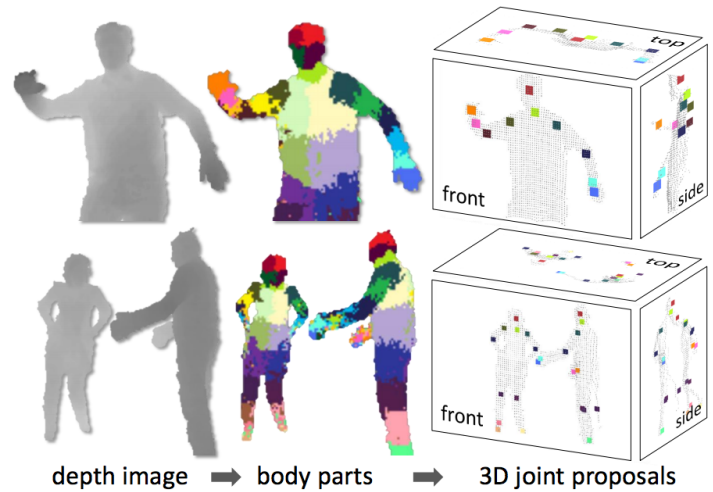
- Works with features of different nature
- Stable to outliers in data.

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.

Applications:

> **Random forests**
> L Breiman - Machine learning, 2001 - Springer
> Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. The generalization error for forests converges as to a limit as the number of
> Cited by 27256 Related articles All 71 versions Web of Science: 12719 Cite Save



- a good approach for tabular data
- Random forest is used to predict movements of individual body parts in Kinect

Random Forest overview

- Impressively simple
- Trees can be trained in parallel
- Doesn't overfit

Random Forest overview

- 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: $N_{\text{used}} = \sqrt{N_{\text{features}}}$

Random Forest overview

- 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: $N_{\text{used}} = \sqrt{N_{\text{features}}}$

But

- Hardly interpretable
- Trained trees take much space, some kind of pre-stopping is required in practice
- Doesn't fix mistakes done by previous trees

Sample weights in ML

Can be used with many estimators. We now have triples

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

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

Sample weights in ML

Can be used with many estimators. We now have triples

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

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

Example for logistic regression:

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

Sample weights in ML

Weights (parameters) of a 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 observations.

Only sample weights when talking about AdaBoost.

AdaBoost [Freund, Shapire, 1995]

Bagging: information from previous trees not taken into account.

Adaptive Boosting is a 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 observations iff $y_i d_j(x_i) = -1$

AdaBoost

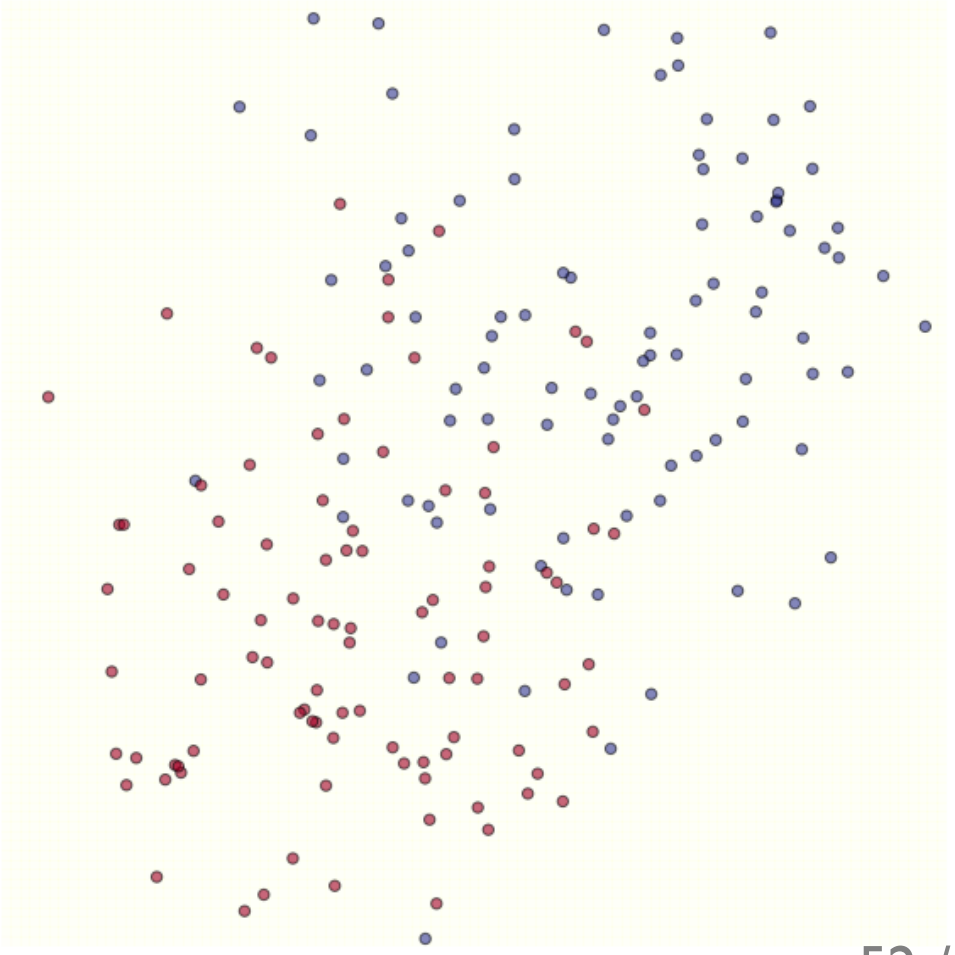
$$D(x) = \sum_j \alpha_j d_j(x)$$

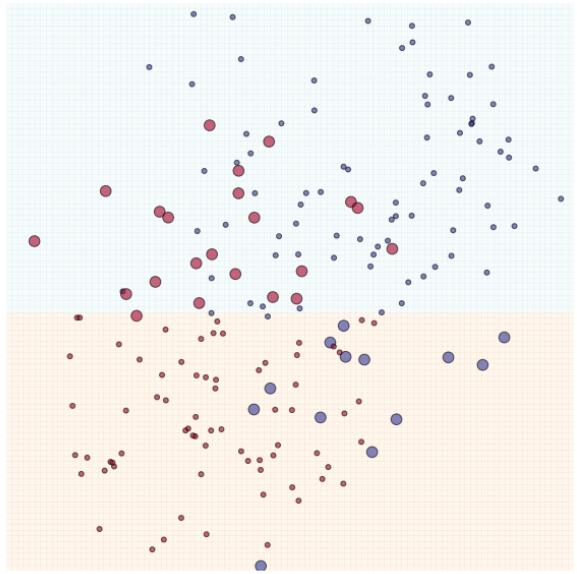
Weak learners are built in sequence, each classifier is trained using different weights

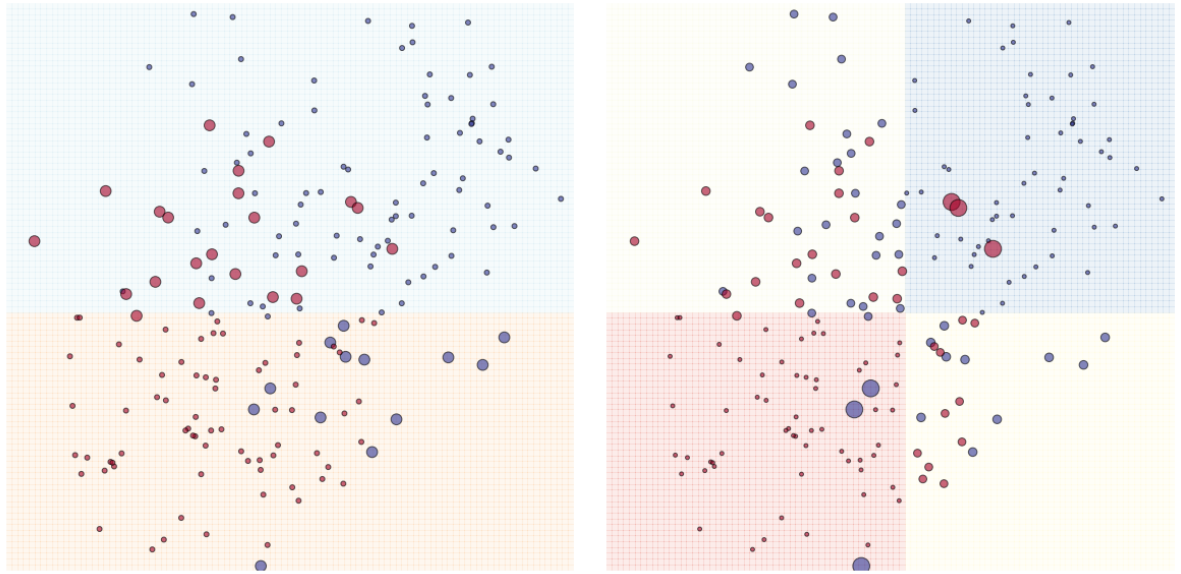
- initially w_i = 1 for each training sample
- After building jth base classifier:
  1. compute the total weight of correctly and wrongly classified observations
     $\alpha_j = \frac{1}{2} \ln \left(\frac{w_{\text{correct}}}{w_{\text{wrong}}} \right)$
  2. increase weight of misclassified samples
     $w_i \leftarrow w_i \times e^{-\alpha_j y_i d_j(x_i)}$

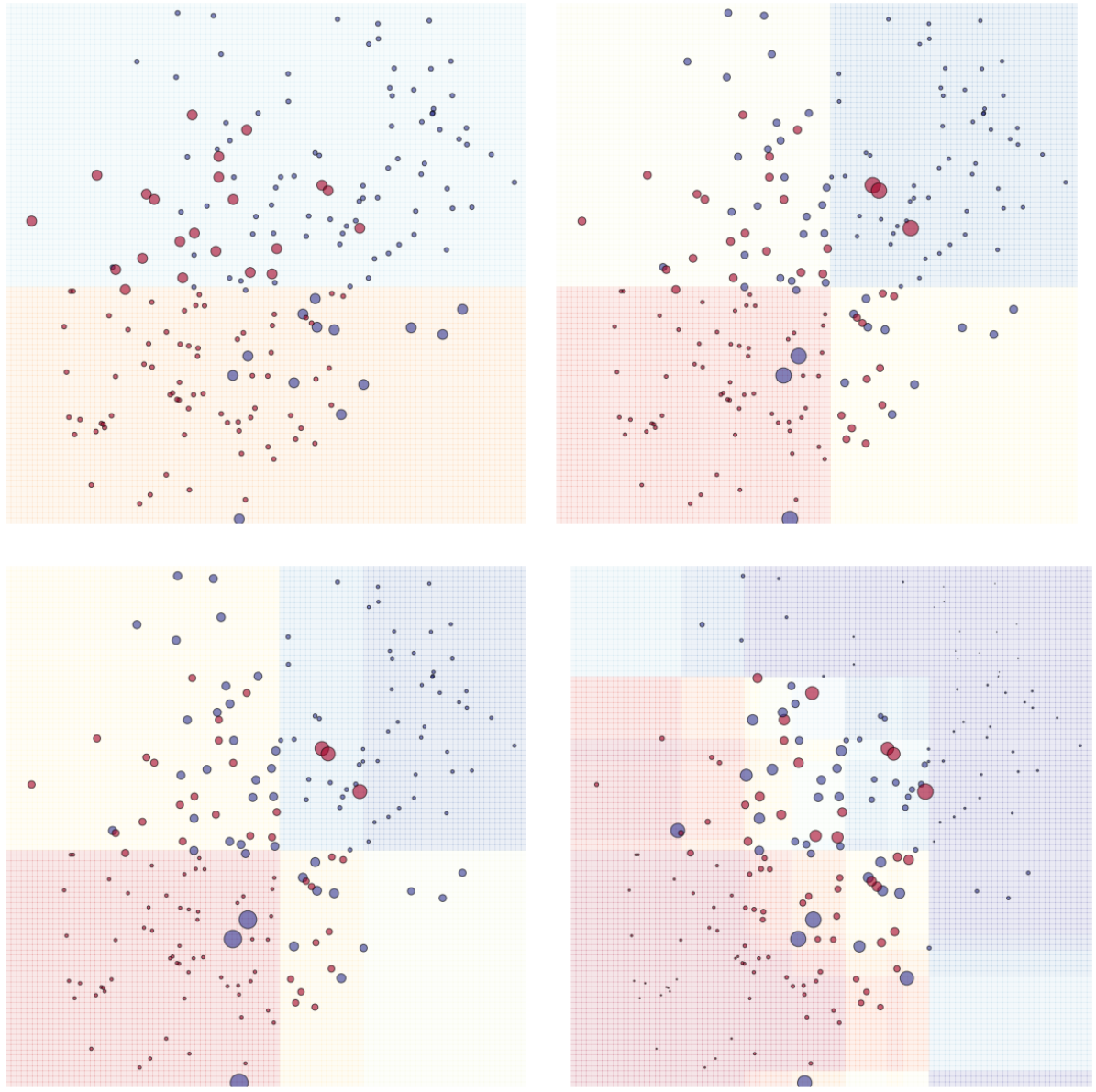
AdaBoost example

Decision trees of depth 1 will be used.









(1, 2, 3, 100 trees)

AdaBoost secret

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

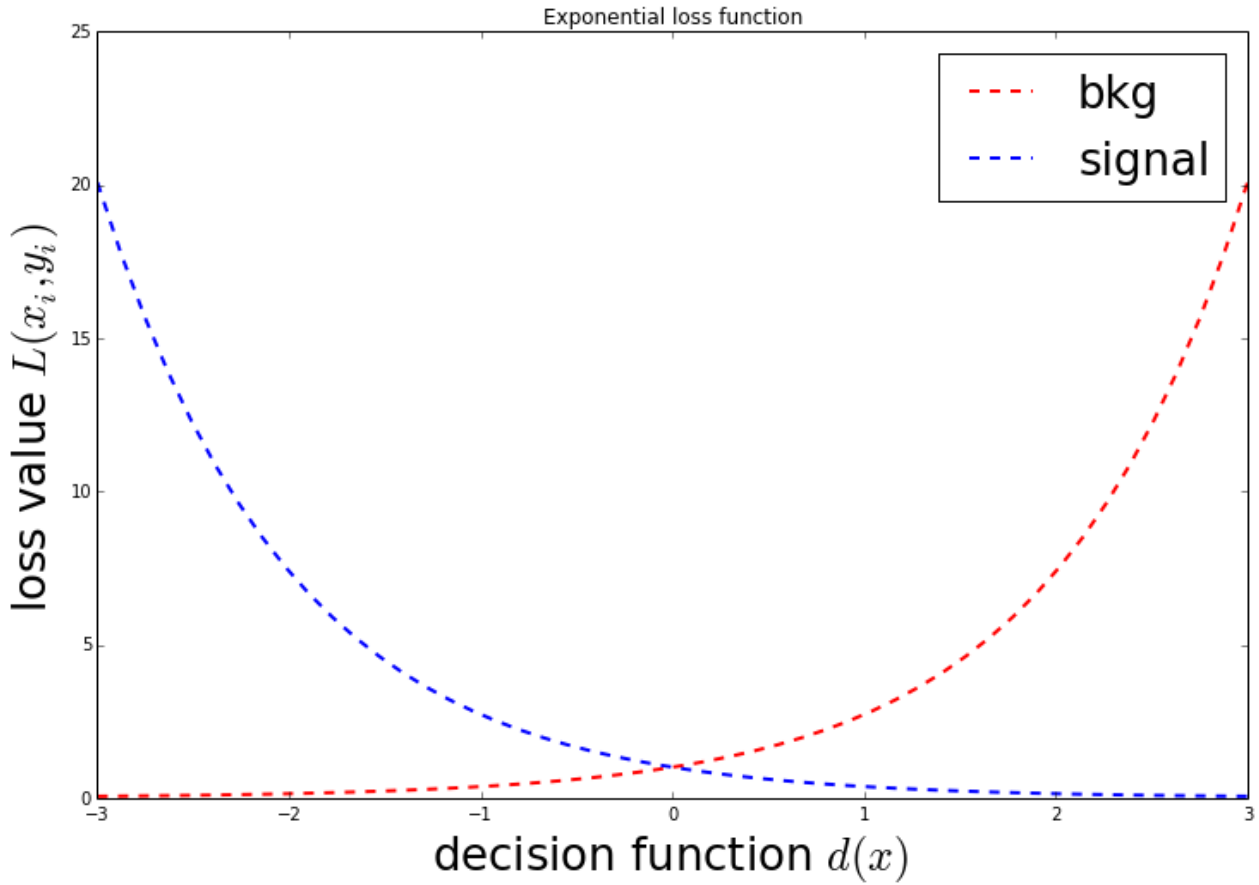
$$\mathcal{L} = \sum_i L(x_i, y_i) = \sum_i \exp(-y_i D(x_i)) \to \min$$

- sample weight is equal to penalty for an observation

$$w_i = L(x_i, y_i) = \exp(-y_i D(x_i))$$

- α_j is obtained as a result of analytical optimization
  Exercise: prove formula for α_j

Loss function of AdaBoost



AdaBoost summary

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

In scikit-learn, one can run AdaBoost over other algorithms.

n-minutes break

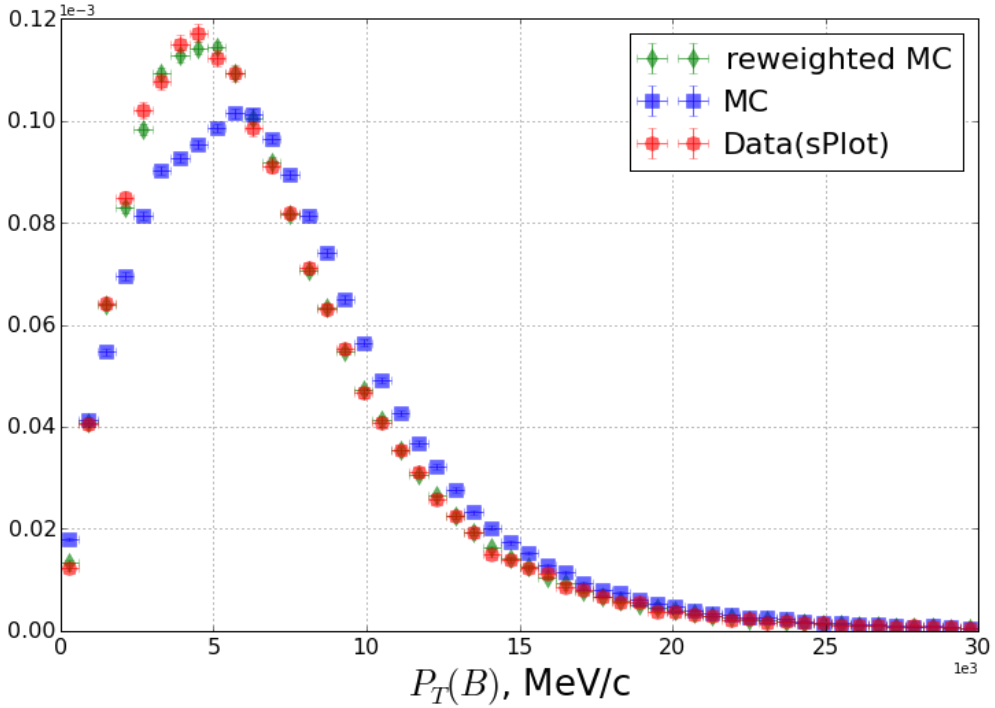
Recapitulation

1. Decision tree:
    - splitting criteria
    - pre-stopping
2. Random Forest: averaging over trees
3. AdaBoost: exponential loss, takes mistakes into account

Reweighting

Reweighting in HEP

- Reweighting in HEP is used to minimize the difference between real samples (target) and Monte Carlo (MC) simulated samples (original)
- The goal is to assign weights to original s.t. original and target distributions coincide



Reweighting in HEP: typical approach

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

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

where $w_{\text{bin, target}}$ and $w_{\text{bin, original}}$ are total weights of samples in a bin for target and original distributions.

Reweighting in HEP: typical approach

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

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

where $w_{\text{bin, target}}$ and $w_{\text{bin, original}}$ are total weights of samples in a bin for target and original distributions.

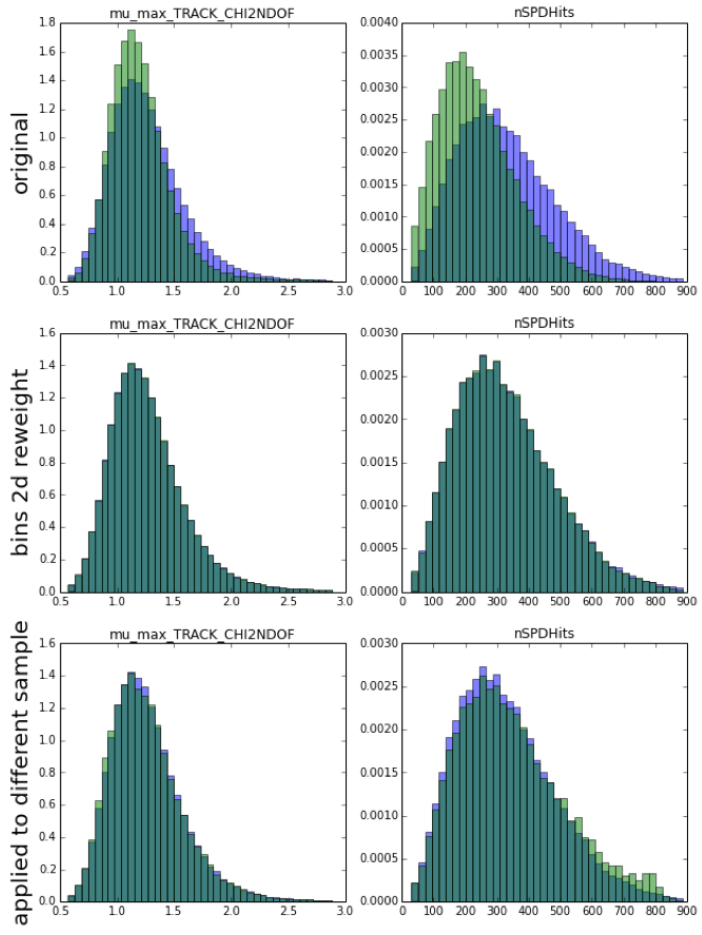
1. simple and fast
2. number of variables is very limited by statistics (typically only one, two)
3. reweighting in one variable may bring disagreement in others
4. which variable is preferable for reweighting?

Reweighting in HEP: typical approach

- Problems arise when there are too few observations in a bin
- This can be detected on a holdout (see the latest row)

Issues:

- few bins - rule is rough
- many bins - rule is not reliable



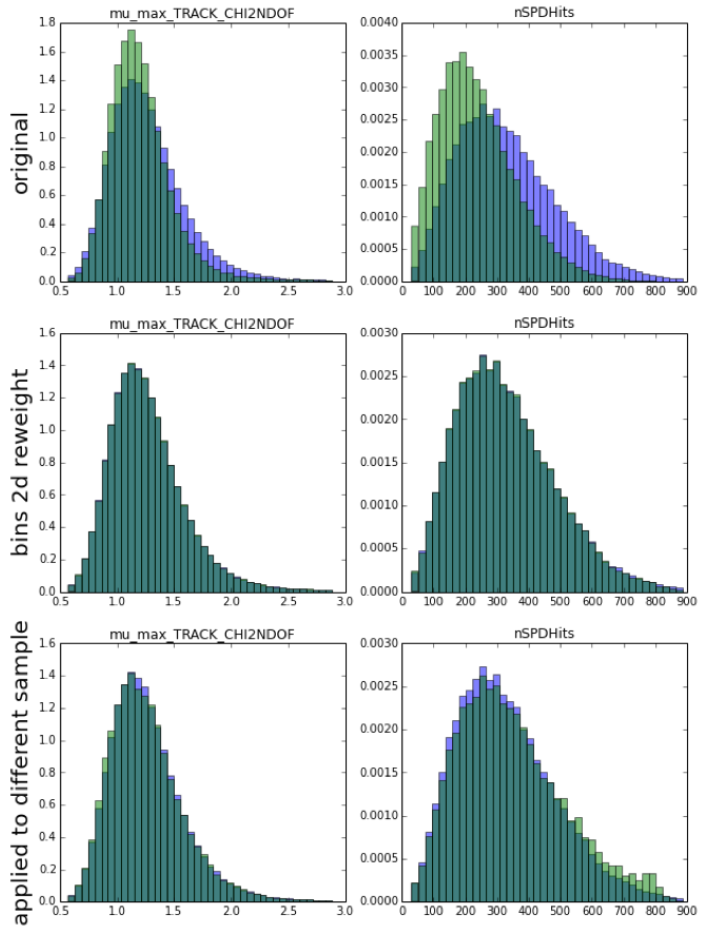
Reweighting in HEP: typical approach

- Problems arise when there are too few observations in a bin
- This can be detected on a holdout (see the latest row)

Issues:

- few bins - rule is rough
- many bins - rule is not reliable

Reweighting rule must be checked on a holdout!

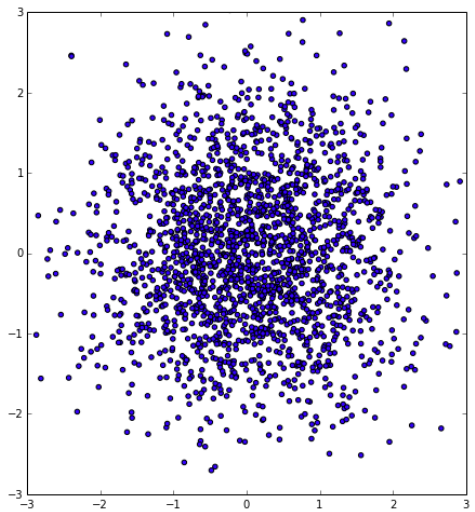


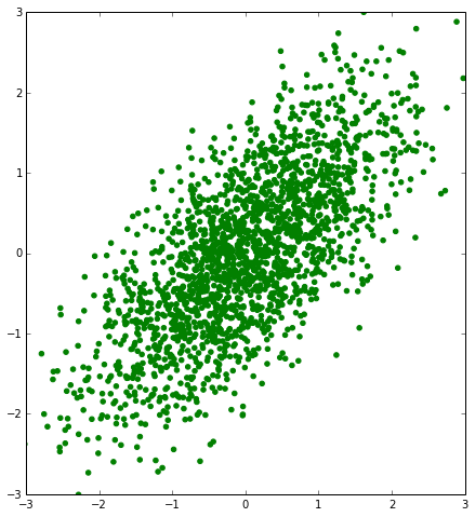
Reweighting quality

- How to check the quality of reweighting?
- One dimensional case: two samples tests (Kolmogorov-Smirnov test, Mann-Whitney test, …)
- Two or more dimensions?
- Comparing 1d projections is not a way

Reweighting quality

- How to check the quality of reweighting?
- One dimensional case: two samples tests (Kolmogorov-Smirnov test, Mann-Whitney test, …)
- Two or more dimensions?
- Comparing 1d projections is not a way

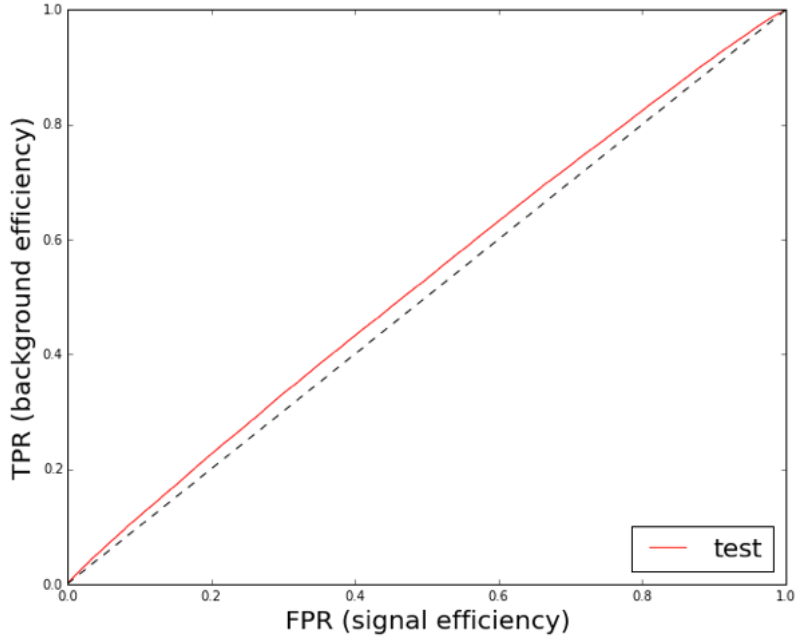




Reweighting quality with ML

- Final goal: classifier doesn't use orignal/target disagreement information = classifier cannot discriminate original and target samples

- Comparison of distributions shall be done using ML:
  - train a classifier to discriminate original and target samples
  - output of the classifier is one-dimensional variable
  - looking at the ROC curve (alternative of two sample test) on a holdout
  - AUC score should be 0.5 if the classifier cannot discriminate original and target samples



Reweighting with density ratio estimation

- We need to estimate density ratio: $\frac{f_{\text{original}}(x)}{f_{\text{target}}(x)}$
- Classifier trained to discriminate original and target samples should reconstruct probabilities $p(\text{original}|x)$ and $p(\text{target}|x)$
- For reweighting we can use

$$\frac{f_{\text{original}}(x)}{f_{\text{target}}(x)} \sim \frac{p(\text{original}|x)}{p(\text{target}|x)}$$

Reweighting with density ratio estimation

- We need to estimate density ratio: $\frac{f_{\text{original}}(x)}{f_{\text{target}}(x)}$
- Classifier trained to discriminate original and target samples should reconstruct probabilities $p(\text{original}|x)$ and $p(\text{target}|x)$
- For reweighting we can use

$$\frac{f_{\text{original}}(x)}{f_{\text{target}}(x)} \sim \frac{p(\text{original}|x)}{p(\text{target}|x)}$$

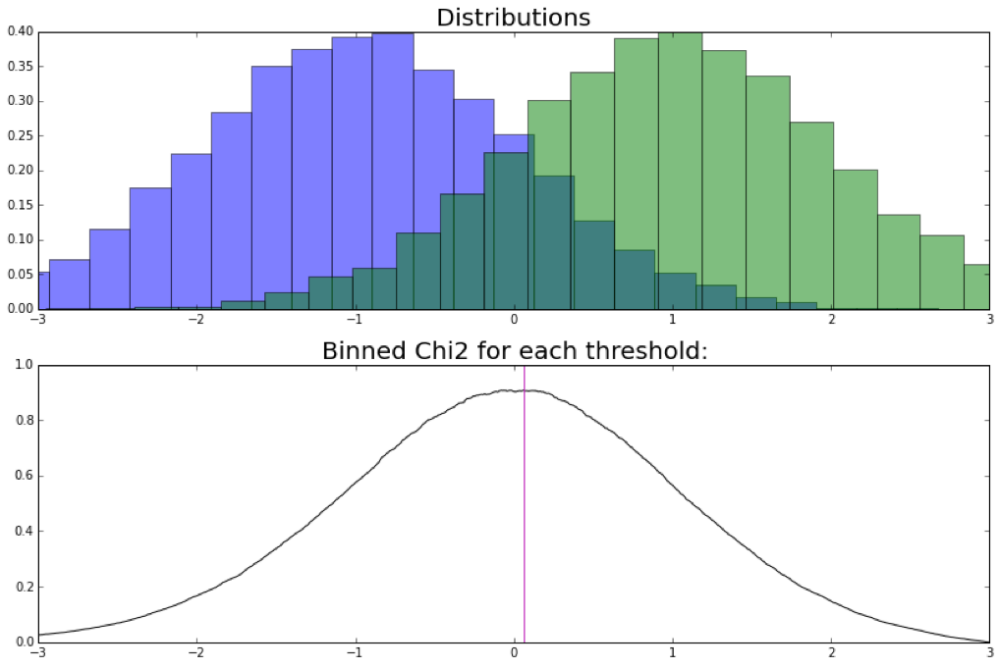
1. Approach is able to reweight in many variables
2. It is successfully tried in HEP, see D. Martschei et al, "Advanced event reweighting using multivariate analysis", 2012
3. There is poor reconstruction when ratio is too small / high
4. It is slower than histogram approach

Reminder

- In histogram approach few bins are bad, many bins are bad too.
- Tree splits the space of variables with orthogonal cuts
- Each tree leaf can be considered as a region, or a bin
- There are different criteria to construct a tree (MSE, Gini index, entropy, …)

Decision tree for reweighting

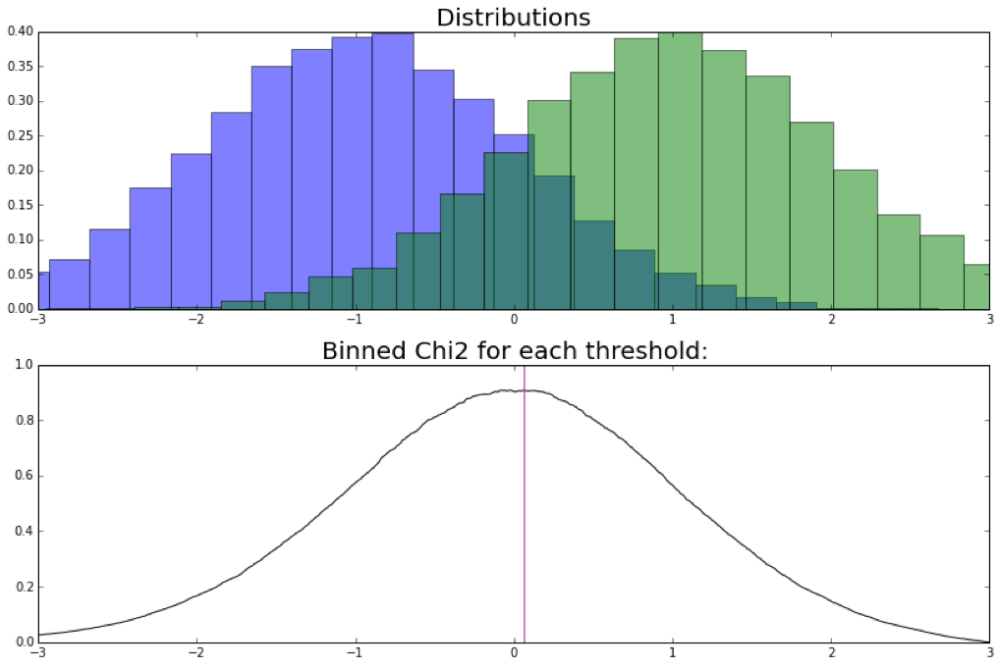
Write ML algorithm to solve directly reweighting problem:



Decision tree for reweighting

Write ML algorithm to solve directly reweighting problem:

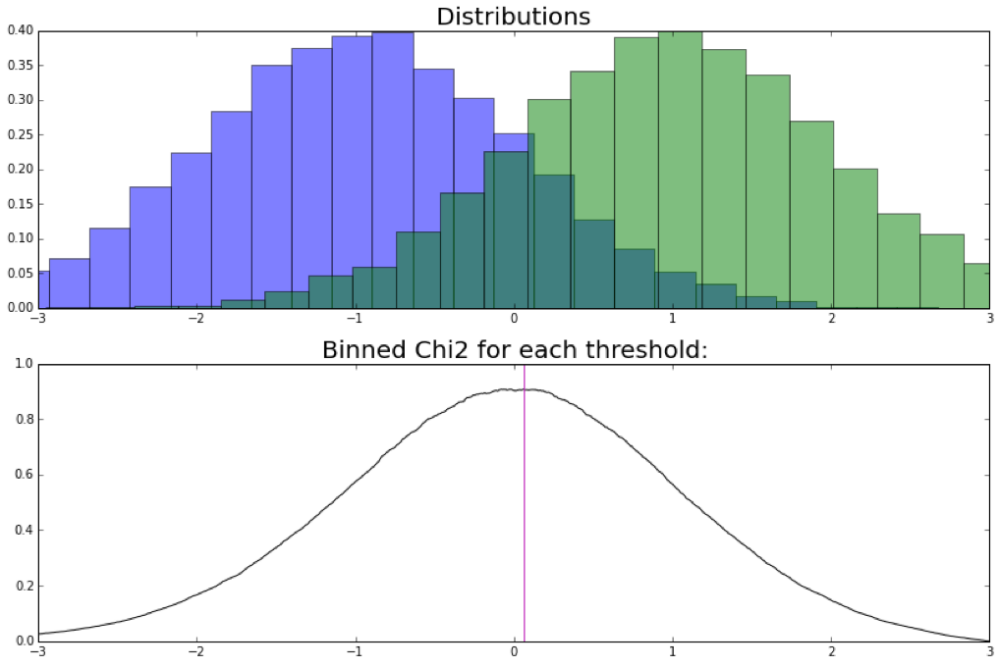
- use tree as a better approach for binning of variables space
- to find regions with the highest difference between original and target distributions use symmetrized χ^2 criteria



Decision tree for reweighting

Write ML algorithm to solve directly reweighting problem:

- use tree as a better approach for binning of variables space
- to find regions with the highest difference between original and target distributions use symmetrized χ^2 criteria



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

where $w_{\text{leaf, original}}$ and $w_{\text{leaf, target}}$ are total weights of samples in a leaf for target and original distributions.

Reweighting with boosted decision trees

Many times repeat the following steps:

Reweighting with boosted decision trees

Many times repeat the following steps:

- build a shallow tree to maximize symmetrized χ^2

Reweighting with boosted decision trees

Many times repeat the following steps:

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

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

Reweighting with boosted decision trees

Many times repeat the following steps:

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

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

- reweight distributions (compare with AdaBoost):

$$w = \begin{cases} w, & \text{if sample from target distribution} \\ w \times e^{\text{pred}}, & \text{if sample from original distribution} \end{cases}$$

Reweighting with boosted decision trees

Comparison with AdaBoost:

- different tree splitting criterion
- different boosting procedure

Reweighting with boosted decision trees

Comparison with AdaBoost:

- different tree splitting criterion
- different boosting procedure

Summary

- uses each time few large bins (construction is done intellectually)
- is able to handle many variables
- requires less data (for the same performance)
- ... but slow (being ML algorithm)

Reweighting applications

- high energy physics (HEP): introducing corrections for simulation samples applied to
  - search for rare decays
  - low-mass dielectrons study

Reweighting applications

- high energy physics (HEP): introducing corrections for simulation samples applied to
  - search for rare decays
  - low-mass dielectrons study

- astronomy: introducing corrections to fight with bias in objects observations
  - objects that are expected to show more interesting properties are preferred when it comes to costly high-quality spectroscopic follow-up observations

Reweighting applications

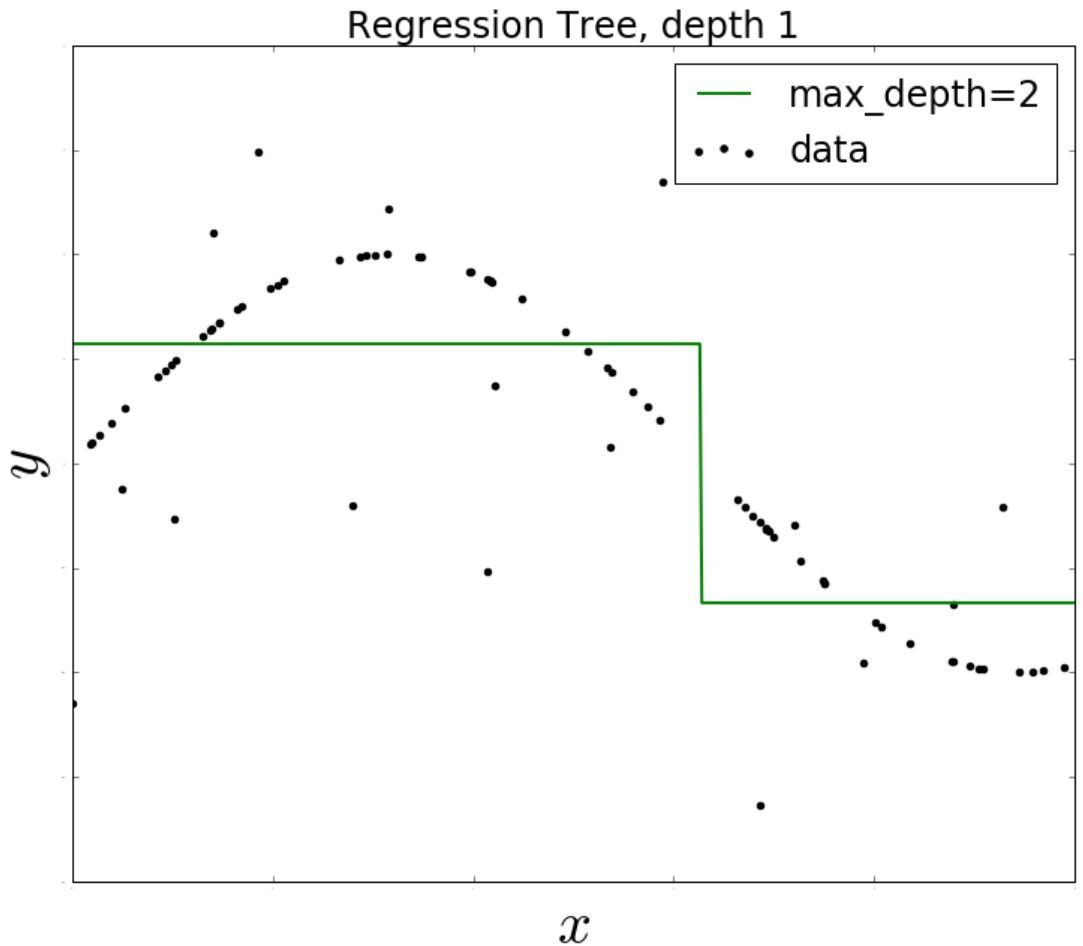
- high energy physics (HEP): introducing corrections for simulation samples applied to
  - search for rare decays
  - low-mass dielectrons study

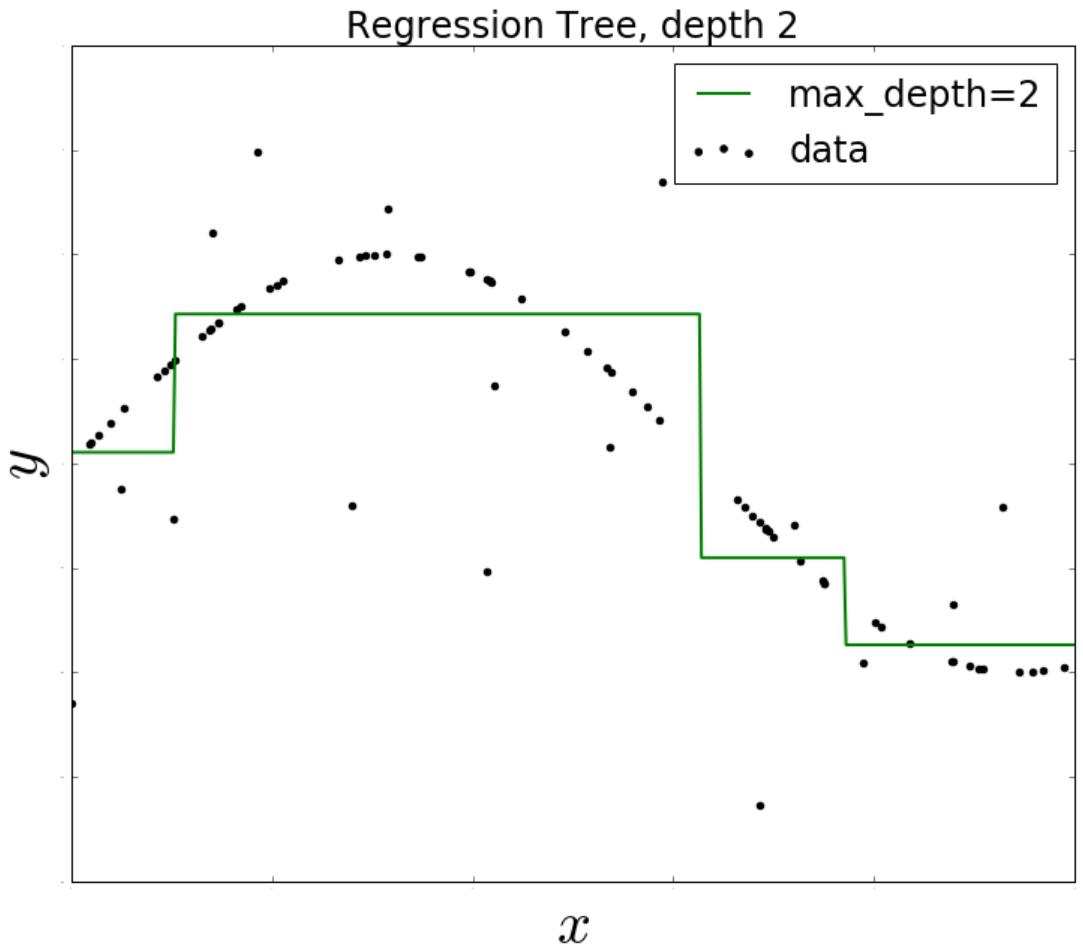
- astronomy: introducing corrections to fight with bias in objects observations
  - objects that are expected to show more interesting properties are preferred when it comes to costly high-quality spectroscopic follow-up observations

- polling: introducing corrections to fight non-response bias
  - assigning higher weight to answers from groups with low response.

Gradient Boosting

Decision trees for regression





Gradient boosting to minimize MSE

Say, we're trying to build an ensemble to minimize MSE:

$$\sum_i (D(x_i) - y_i)^2 \to \min$$

and ensemble's prediction is obtained by taking sum over trees

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

How do we train this ensemble?

Introduce $D_j(x) = \sum_{j'=1}^{j} d_{j'}(x) = D_{j-1}(x) + d_j(x)$

Natural solution is to greedily minimize MSE:

$$\sum_i (D_j(x_i) - y_i)^2 = \sum_i (D_{j-1}(x_i) + d_j(x_i) - y_i)^2 \to \min$$

assuming that we already built $j - 1$ estimators.

Introduce $D_j(x) = \sum_{j'=1}^{j} d_{j'}(x) = D_{j-1}(x) + d_j(x)$

Natural solution is to greedily minimize MSE:

$$\sum_i (D_j(x_i) - y_i)^2 = \sum_i (D_{j-1}(x_i) + d_j(x_i) - y_i)^2 \to \min$$

assuming that we already built $j-1$ estimators.

Introduce *residual*: $R_j(x_i) = y_i - D_{j-1}(x_i)$, and rewrite MSE

$$\sum_i (d_j(x_i) - R_j(x_i))^2 \to \min$$

So the jth estimator (tree) is trained using the following data:

$$x_i, R_j(x_i)$$

Gradient Boosting visualization

Gradient Boosting [Friedman, 1999]

composition of weak regressors,

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

Borrow an approach to encode probabilities from logistic regression

$$p_{+1}(x) = \sigma(D(x))$$
$$p_{-1}(x) = \sigma(-D(x))$$

Optimization of log-likelihood ($y_i = \pm 1$):

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

Gradient Boosting

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

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

- Optimization problem: find all α_j and weak leaners d_j
- Mission impossible

Gradient Boosting

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

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

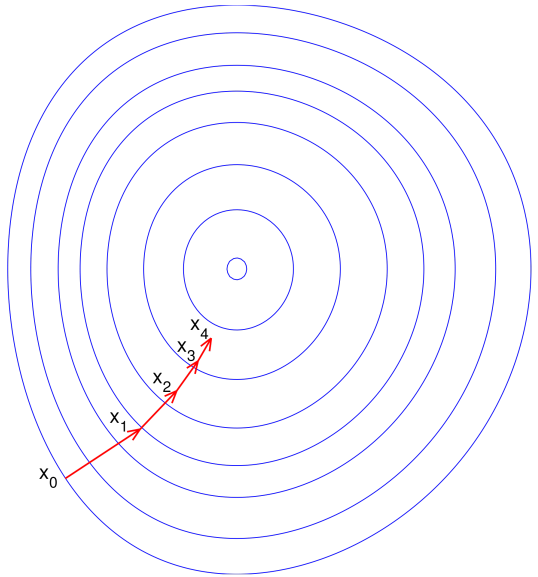
- Optimization problem: find all α_j and weak leaners d_j
- Mission impossible
- Main point: greedy optimization of loss function by training one more weak learner d_j
- Each new estimator follows the gradient of loss function

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 *j*th iteration:

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

Important exercise: compute pseudo-residuals for MSE and logistic losses.

Additional GB tricks

to make training more stable, add

- *learning rate* η:

$$D_j(x) = \sum_j \eta \alpha_j d_j(x)$$

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

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

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

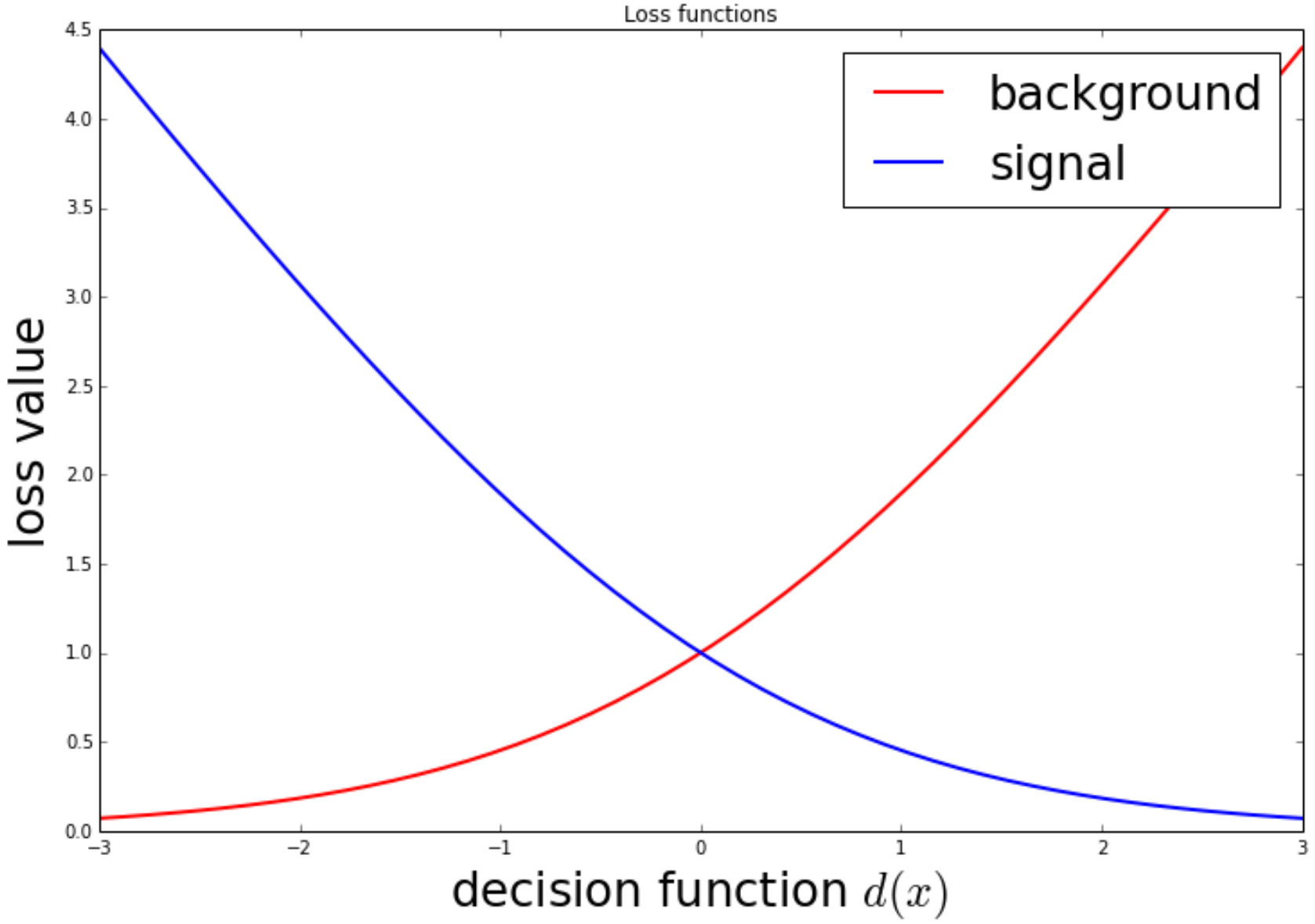
This loss function is called *ExpLoss* or *AdaLoss*.

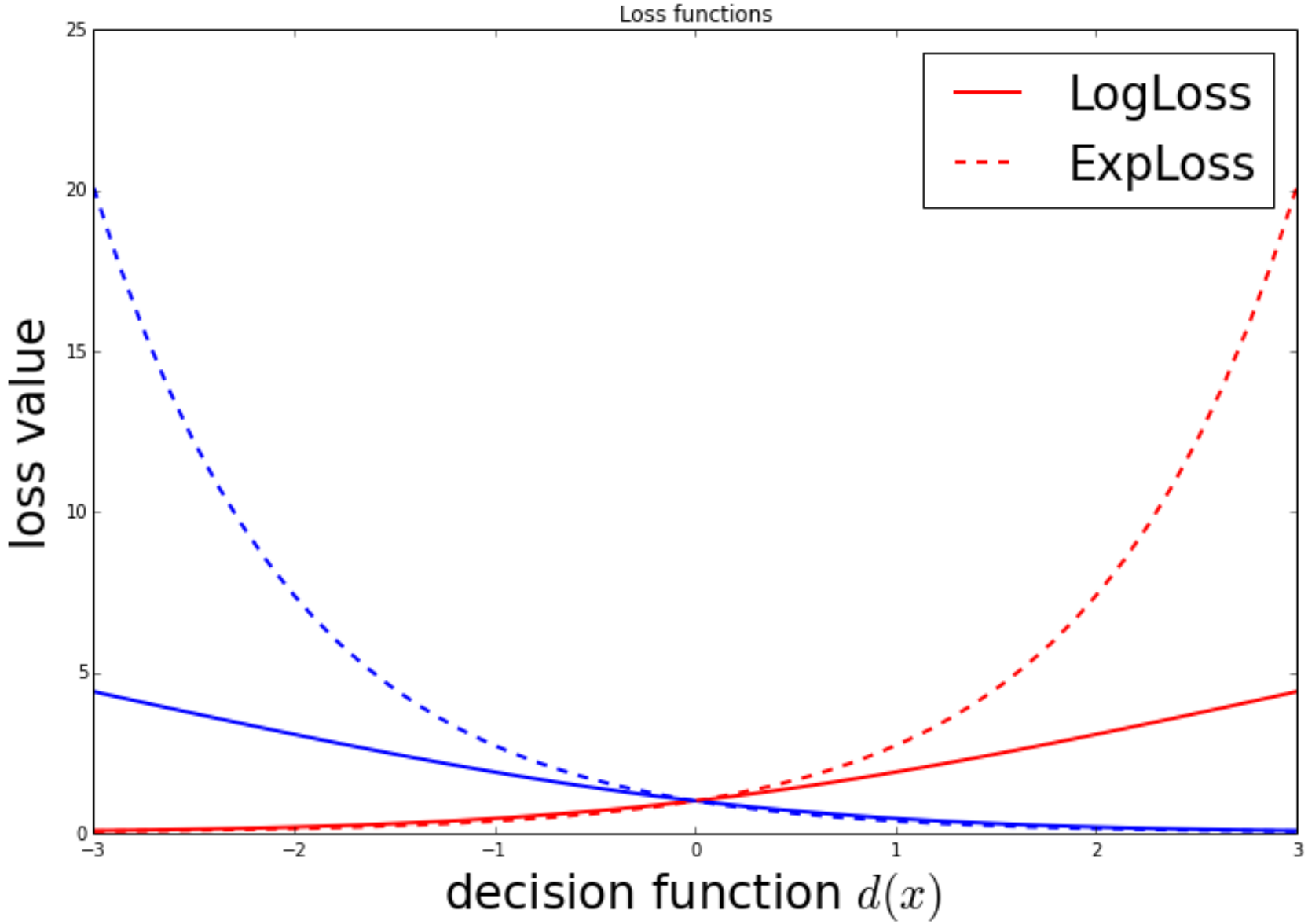
*(also AdaBoost expects that $d_j(x_i) = \pm 1$)

Loss functions

Gradient boosting can optimize different smooth loss function (apart from the gradients second-order derivatives can be used).

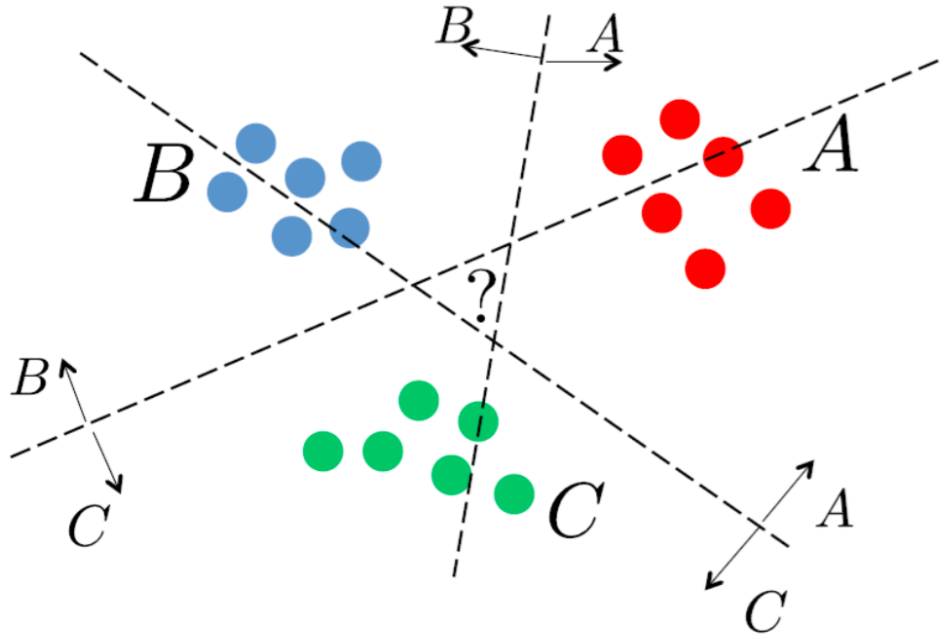
- regression, $y \in \mathbb{R}$
  - Mean Squared Error $\sum_i (d(x_i) - y_i)^2$
  - Mean Absolute Error $\sum_i |d(x_i) - y_i|$
- binary classification, $y_i = \pm 1$
  - ExpLoss (ada AdaLoss) $\sum_i e^{-y_i d(x_i)}$
  - LogLoss $\sum_i \log(1 + e^{-y_i d(x_i)})$


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

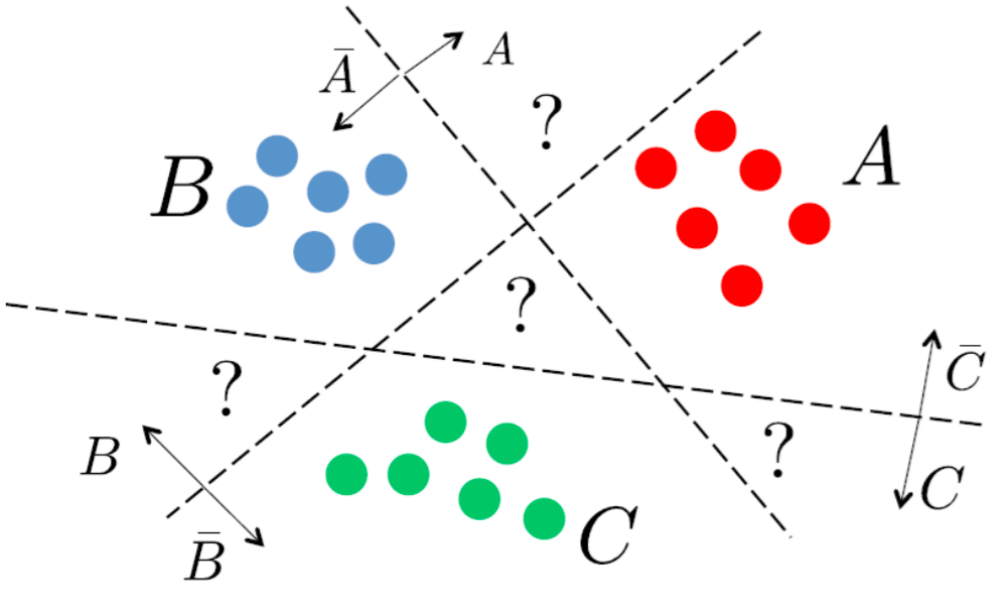

Loss functions
LogLoss
ExpLoss
loss value
decision function $d(x)$
25
20
15
10
5
0
−3
−2
−1
0
1
2
3

Gradient Boosting classification playground

Multiclass classification: ensembling



One-vs-one, $\frac{n_{\text{classes}} \times (n_{\text{classes}} - 1)}{2}$



One-vs-rest, n_{classes}

scikit-learn implements those as meta-algorithms.

Multiclass classification: modifying an algorithm

Most classifiers have natural generalizations to multiclass classification.

Example for logistic regression: introduce for each class $c \in 1, 2, ..., C$ a vector w_c.

$$d_c(x) =< w_c, x >$$

Converting to probabilities using *softmax* function:

$$p_c(x) = \frac{e^{d_c(x)}}{\sum_{\tilde{c}} e^{d_{\tilde{c}}(x)}}$$

And minimize LogLoss: $\mathcal{L} = \sum_i - \log p_{y_i}(x_i)$

Softmax function

Typical way to convert n numbers to n probabilities.

Mapping is surjective, but not injective (n dimensions to $n-1$ dimension). Invariant to global shift:

$$d_c(x) \to d_c(x) + \text{const}$$

For the case of two classes: $p_1(x) = \frac{e^{d_1(x)}}{e^{d_1(x)} + e^{d_2(x)}} = \frac{1}{1 + e^{d_2(x) - d_1(x)}}$

Coincides with logistic function for $d(x) = d_1(x) - d_2(x)$

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 penalize for misordering:

$$\mathcal{L} = \sum_{i,\tilde{i}} L(x_i, x_{\tilde{i}}, y_i, y_{\tilde{i}})$$

$$L(x, \tilde{x}, y, \tilde{y}) = \begin{cases} \sigma(d(\tilde{x}) - d(x)), & y < \tilde{y} \\ 0, & \text{otherwise} \end{cases}$$

Boosting applications

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

It is powerful for tabular data of different nature.

Among the 29 challenge winning solutions published at Kaggle's blog during 2015, 17 solutions used XGBoost. Among these solutions, eight solely used XGBoost to train the model, while most others combined XGBoost with neural nets in ensembles.

Boosting applications

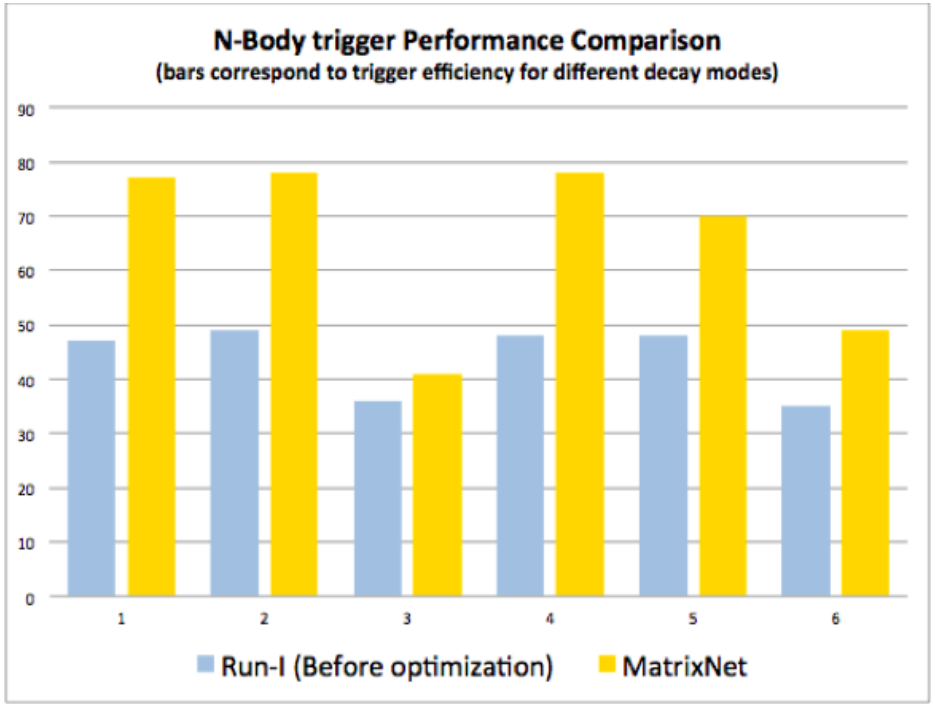
- ranking
  - search engine
    - search engine Yandex uses own variant of gradient boosting which uses oblivious decision trees
    - Bing's search uses RankNet, gradient boosting for ranking invented by Microsoft Research
  - recommendations
    - as the final ranking model combining properties of user, text, image contents

Boosting applications

- classification and regression
  - predicting clicks on ads at Facebook
  - in high energy physics to combine geometric and kinematic properties
    - B-meson flavour tagging
    - trigger system
    - particle identification
    - rare decays search
  - store sales prediction
  - motion detection
  - customer behavior prediction
  - web text classification

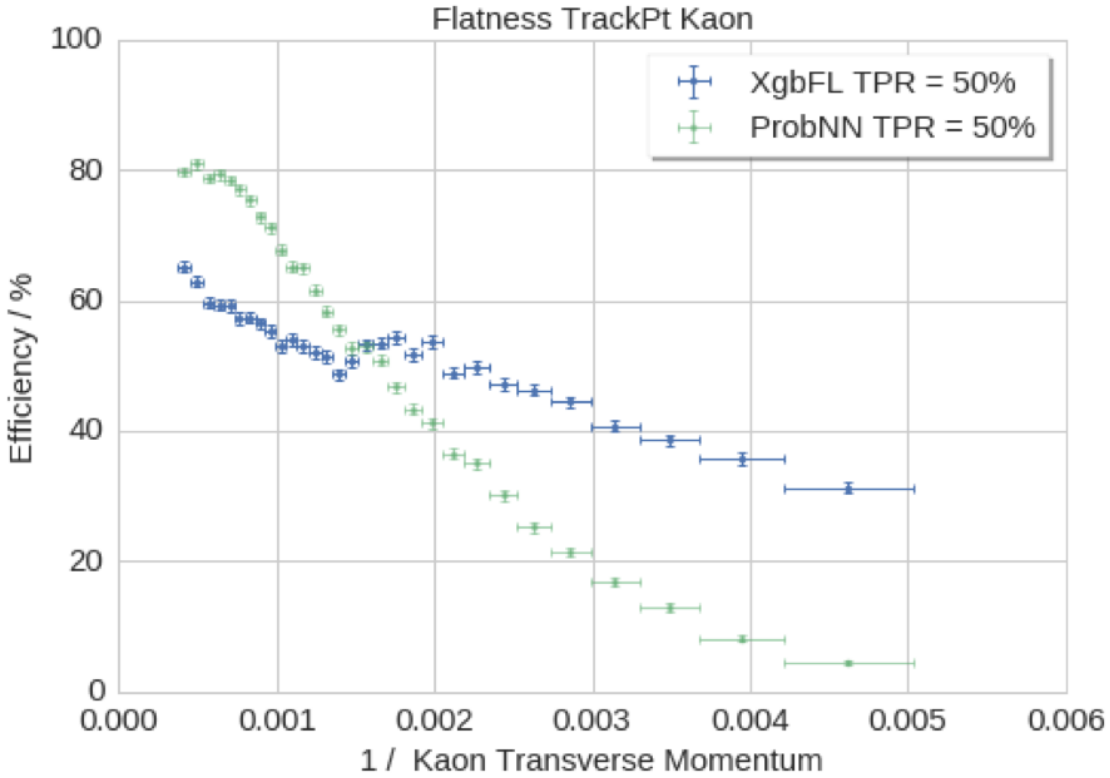
Trigger system

- The goal is to select interesting proton-proton collisions based on detailed online analysis of measured physics information
- Trigger system can consists of two parts: hardware and software
- It selects several thousands collisions among 40 millions per second $\rightarrow$ should be fast
- We have improved LHCb software topological trigger to select beauty and charm hadrons
  - Yandex gradient boosting over oblivious decision trees (MatrixNet) is used



Boosting to uniformity

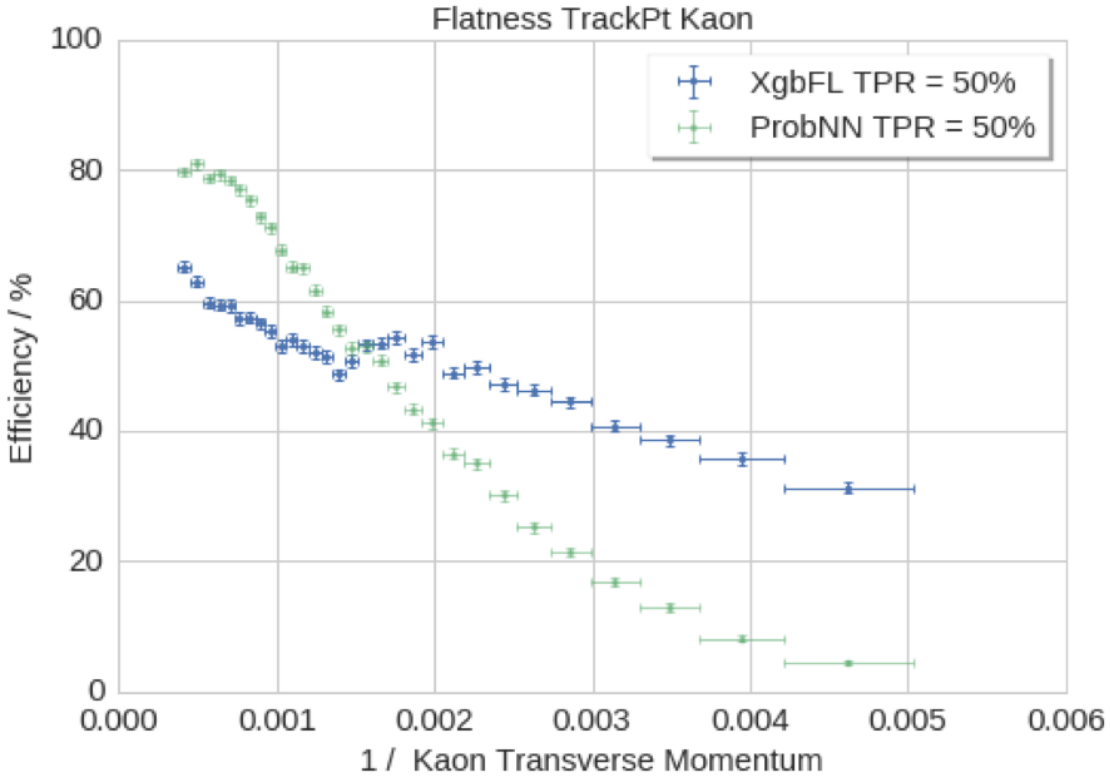
In HEP applications additional restriction can arise: classifier should have constant efficiency (FPR/TPR) against some variable.



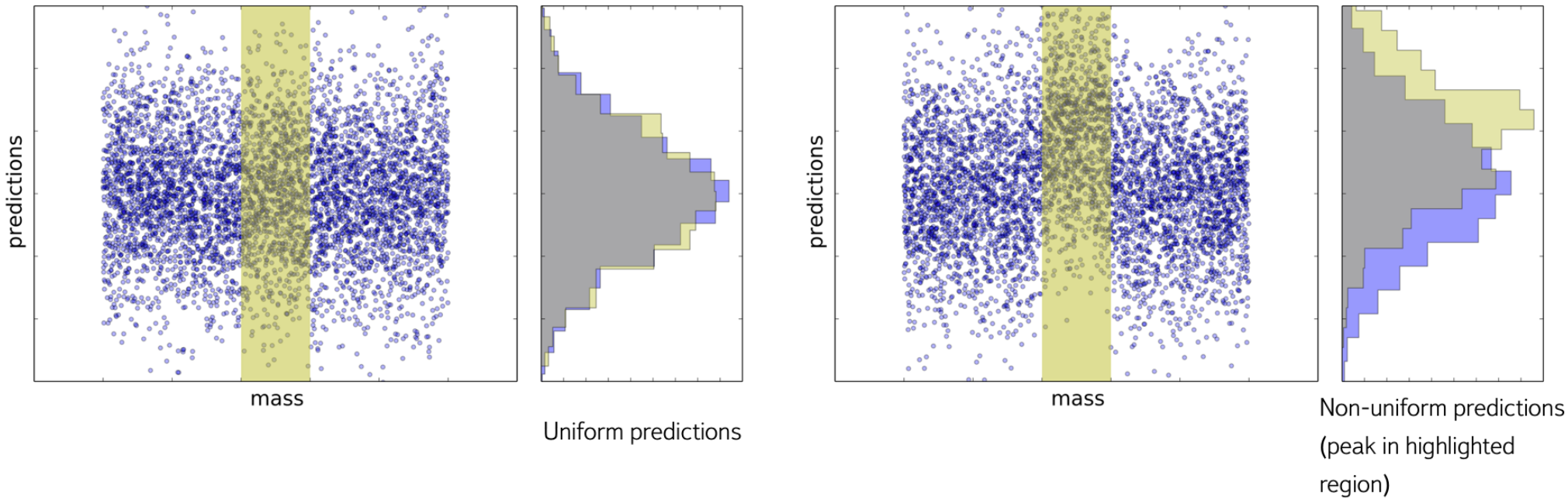
Boosting to uniformity

In HEP applications additional restriction can arise: classifier should have constant efficiency (FPR/TPR) against some variable.

- select particles independently on its life time (trigger system)
- identify particle type independently on its momentum (particle identification)
- select particles independently on its invariant mass (rare decays search)



Non-uniformity measure

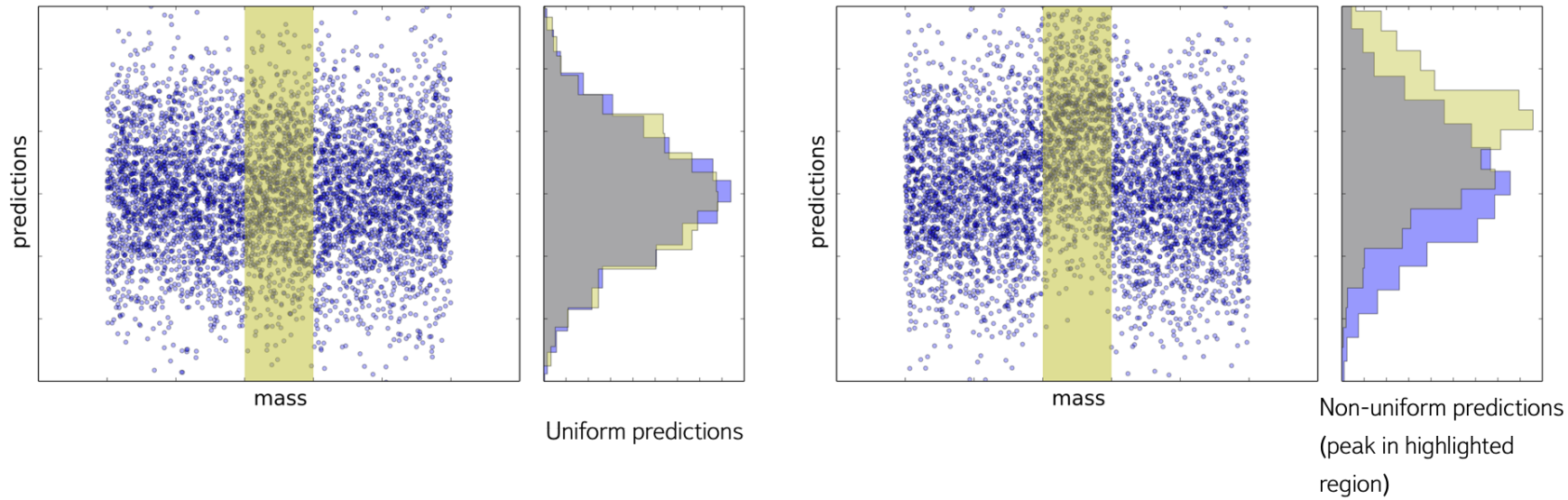


- difference in the efficiency can be detected analyzing distributions
- uniformity = no dependence between mass and predictions

Non-uniformity measure

Average contributions (difference between global and local distributions) from different regions in the variable using Cramer-von Mises measure (integral characteristic)

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



Minimizing non-uniformity measure

- why not minimizing CvM as a loss function with GB?

Minimizing non-uniformity measure

- why not minimizing CvM as a loss function with GB?
- … because we can’t compute the gradient
- and minimizing CvM doesn't encounter classification problem:
  - the minimum of CvM is achieved i.e. on a classifier with random predictions

Minimizing non-uniformity measure

- why not minimizing CvM as a loss function with GB?
- … because we can’t compute the gradient
- and minimizing CvM doesn't encounter classification problem:
  - the minimum of CvM is achieved i.e. on a classifier with random predictions
- ROC AUC, classification accuracy are not differentiable too; however, logistic loss, for example, works well

Flatness loss

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

$$\mathcal{L} = \mathcal{L}_{\text{AdaLoass}} + \alpha \mathcal{L}_{\text{FlatnessLoss}}$$

where α is a trade off classification quality and uniformity

Flatness loss

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

$$\mathcal{L} = \mathcal{L}_{\text{AdaLoass}} + \alpha \mathcal{L}_{\text{FlatnessLoss}}$$

where α is a trade off classification quality and uniformity

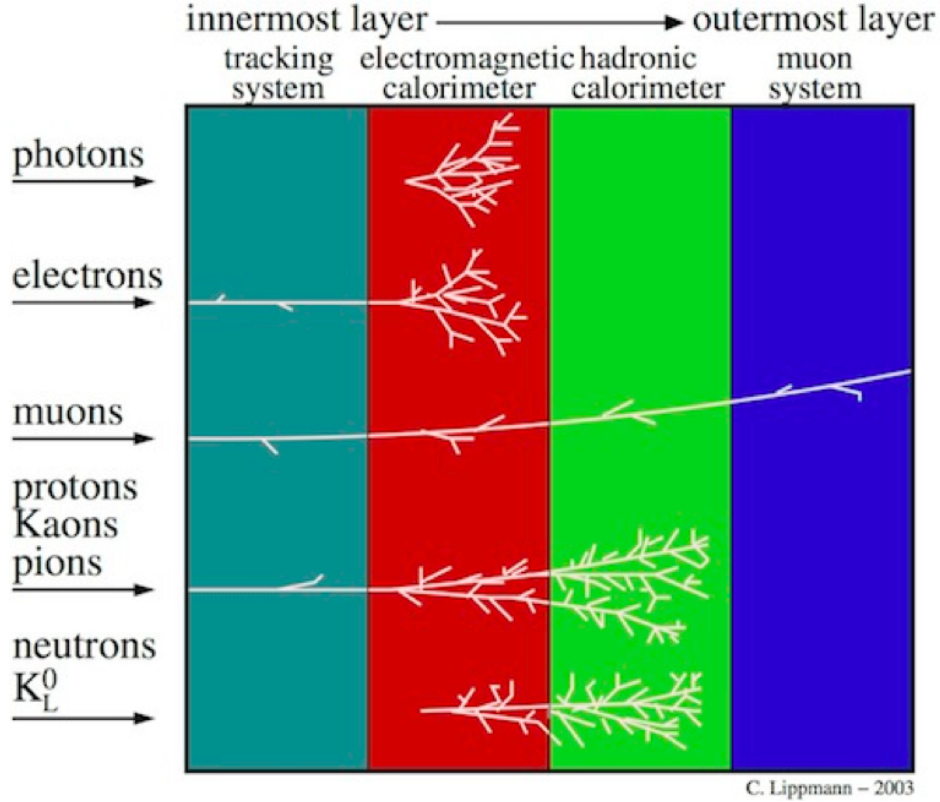
- Flatness loss approximates non-differentiable CvM measure

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

$$\frac{\partial}{\partial D(x_i)} \mathcal{L}_{\text{FlatnessLoss}} \sim 2 \left[F_{\text{region}}(s) - F_{\text{global}}(s) \right] |_{s=D(x_i)}$$

Particle Identification

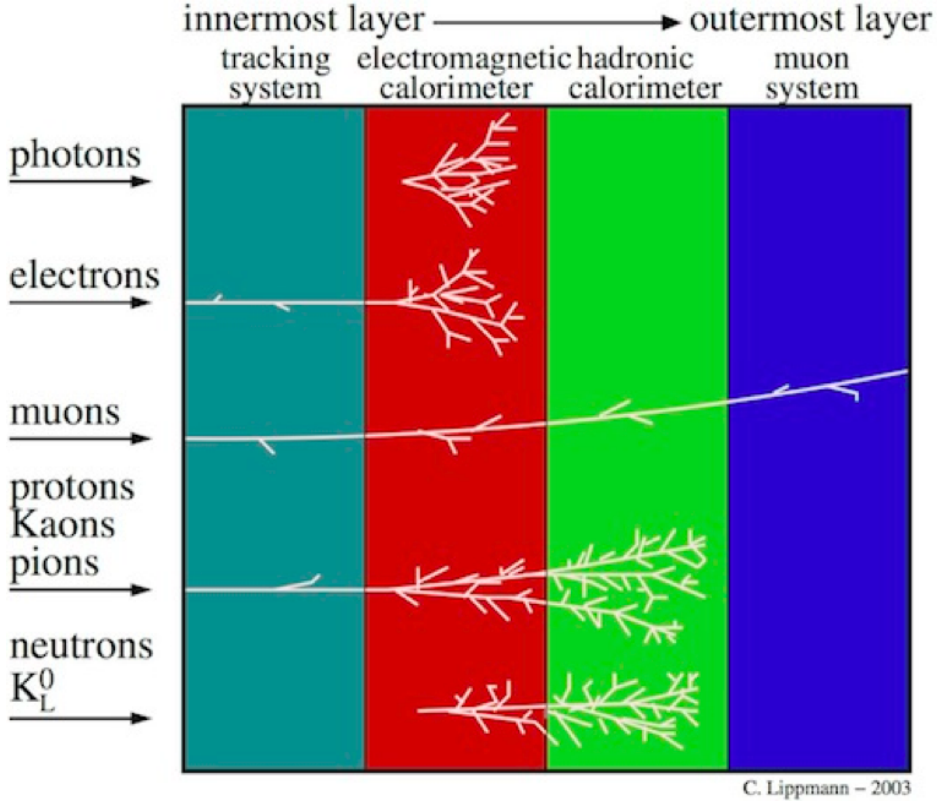
- Problem: identify charged particle associated with a track (multiclass classification problem)
  - particle "types": Ghost, Electron, Muon, Pion, Kaon, Proton.



Particle Identification

- Problem: identify charged particle associated with a track (multiclass classification problem)
  - particle "types": Ghost, Electron, Muon, Pion, Kaon, Proton.

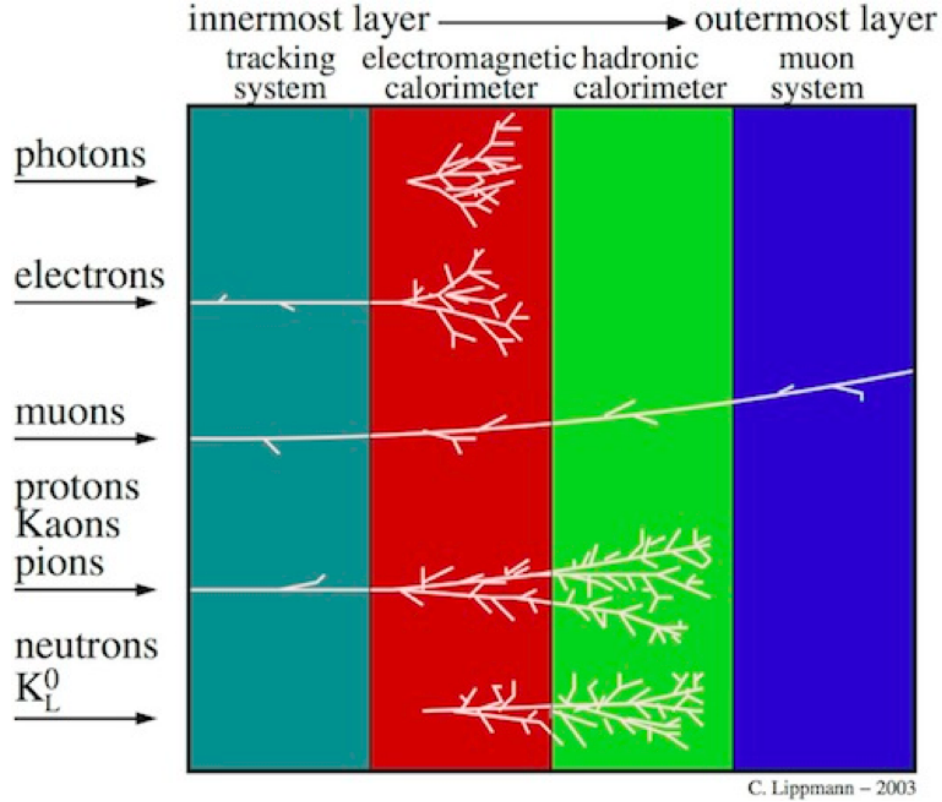
- LHCb detector provides diverse plentiful information, collected by subdetectors: calorimeters, (Ring Imaging) Cherenkov detector, muon chambers and tracker observables
  - this information can be efficiently combined using ML



Particle Identification

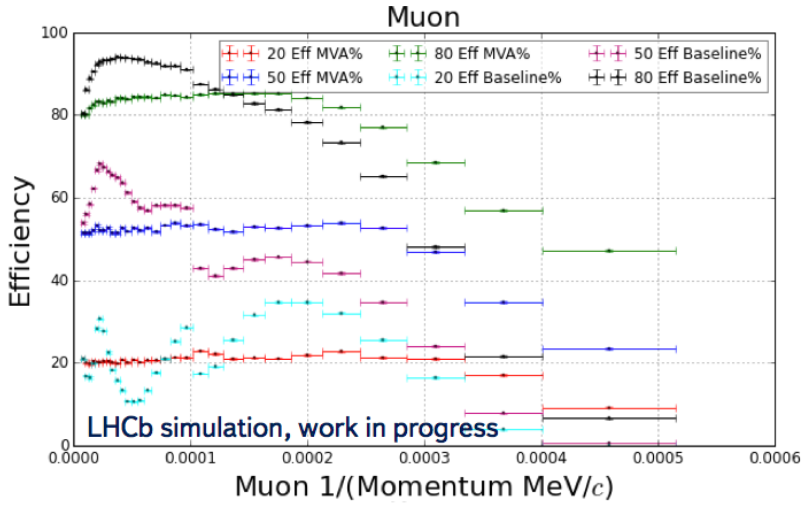
- Problem: identify charged particle associated with a track (multiclass classification problem)
  - particle "types": Ghost, Electron, Muon, Pion, Kaon, Proton.

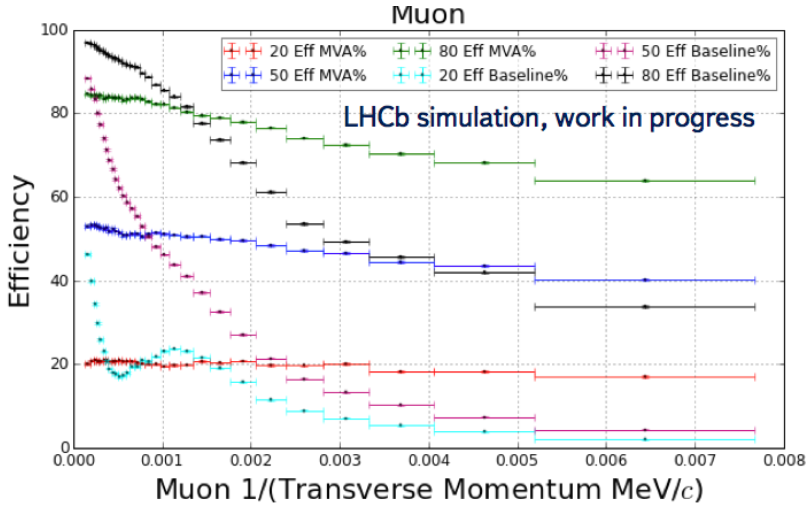
- LHCb detector provides diverse plentiful information, collected by subdetectors: calorimeters, (Ring Imaging) Cherenkov detector, muon chambers and tracker observables
  - this information can be efficiently combined using ML

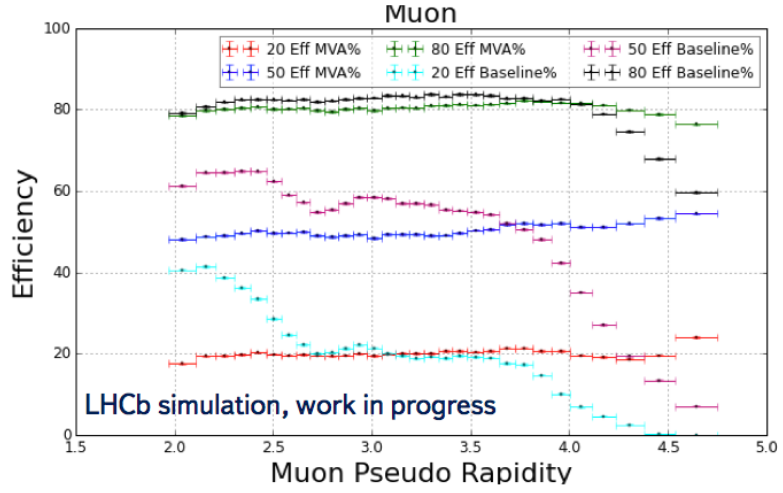


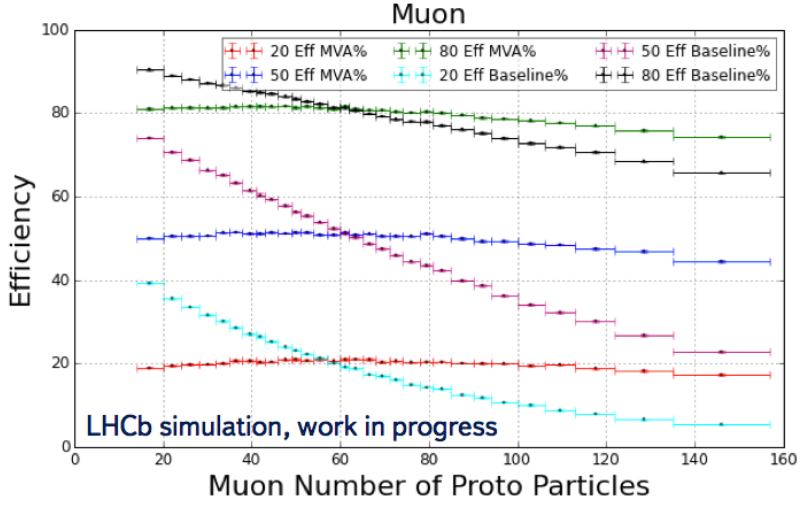
- Identification should be independent on particle momentum, pseudorapidity, etc.

Flatness loss for particle identification





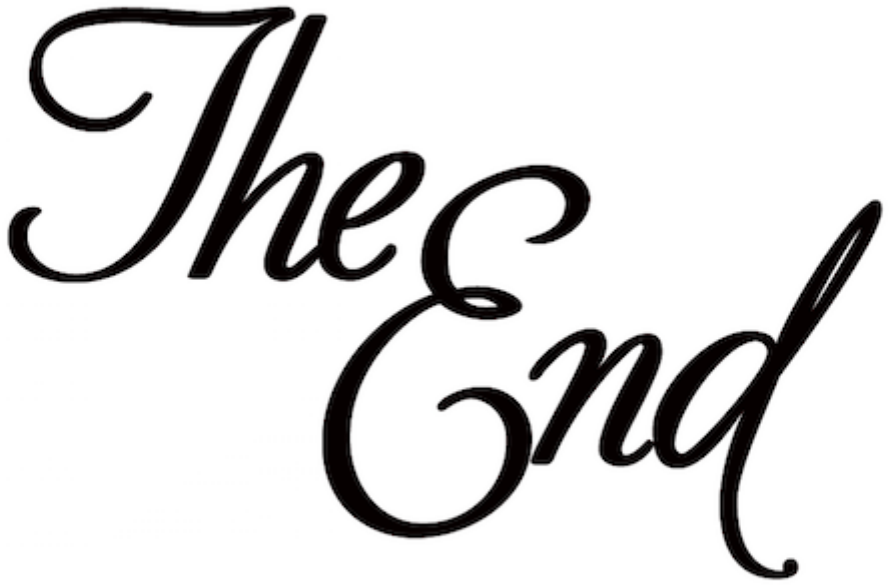




Gradient Boosting overview

A powerful ensembling technique (typically used over trees, GBDT)

- a general way to optimize differentiable losses
  - can be adapted to other problems
- 'following' the gradient of loss at each step
- making steps *in the space of functions*
- gradient of poorly-classified observations is higher
- increasing number of trees can drive to overfitting
  (= getting worse quality on new data)
- requires tuning, better when trees are not complex
- widely used in practice


The End