Mathematical techniques in data science

Lecture 10: Boosting

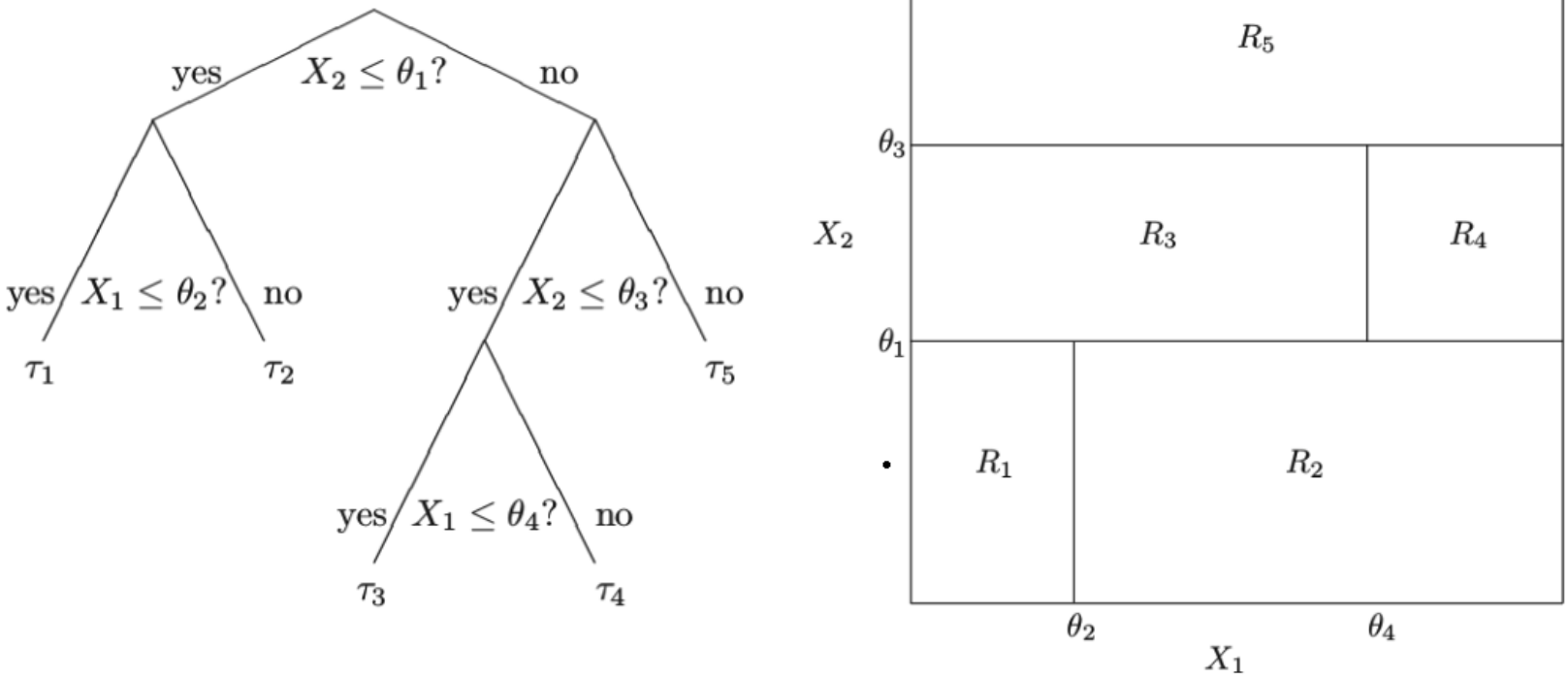
Mathematical techniques in data sciences

- A short introduction to statistical learning theory
- Tree-based methods — boosting and bootstrapping
- SVM – the kernel trick
- Linear regression – regularization and feature selection

Decision trees

Tree-based methods

- Partition the feature space into a set of rectangles
- Fit a simple model (e.g. a constant) in each rectangle
- Conceptually simple yet powerful



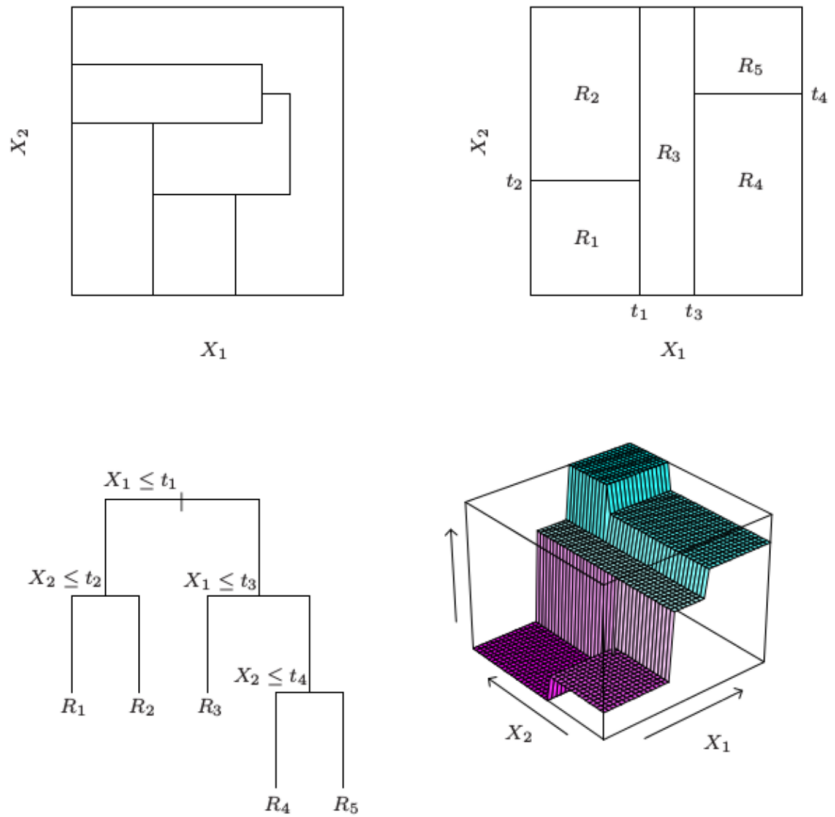
Izenman, 2013, Figure 9.1.

Tree-based methods

- Advantages:
  - Often mimics human decision-making process (e.g. doctor examining patient).
  - Very easy to explain and interpret.
  - Can handle both regression and classification problems.
- Disadvantage: Basic implementation is generally not competitive compared to other methods.
- However, by aggregating many decision trees and using other variants, one can improve the performance significantly.
- Such techniques may lead to state-of-the-art models. However, in doing so, one loses the easy interpretability of decision trees.

Decision trees

To simplify, we will only consider **binary** decision trees.



ESL, Figure 9.2.

Top Left: Not binary. Top Right: binary.

Bottom Left: Tree corresponding to Top Right partition. Bottom Right: Prediction surface.

How to grow a decision tree?

Regression tree:

- Data: $y \in \mathbb{R}^n$, $X \in \mathbb{R}^{n \times p}$.
- Each observation: $(y_i, x_i) \in \mathbb{R}^{p+1}$, $i = 1, \ldots, n$.

Suppose we have a partition of $\mathbb{R}^p$ into M regions $R_1, \ldots, R_m$.

We predict the response using a constant on each R_i:

$$f(x) = \sum_{i=1}^{m} c_i \cdot \mathbf{1}_{x \in R_i}.$$

In order to minimize $\sum_{i=1}^{n} (y_i - f(x_i))^2$, one needs to choose:

$$\hat{c}_i = \text{ave}(y_j : x_j \in R_i).$$

How do we determine the regions R_i, i.e., how do we "grow" the tree?

We need to decide:

1. Which variable to split.
2. Where to split that variable.

How to grow a decision tree?

- Finding a (globally) optimal tree is generally computationally infeasible.
- We use a greedy algorithm.

Consider a splitting variable $j \in \{1, \ldots, p\}$ and splitting point $s \in \mathbb{R}$.

Define the two half-planes:

$$R_1(j, s) := \{x \in \mathbb{R}^p : x_j \leq s\}, \qquad R_2(j, s) := \{x \in \mathbb{R}^p : x_j > s\}.$$

We choose j, s to minimize

$$\min_{j,s} \left[\min_{c_1 \in \mathbb{R}} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2 \in \mathbb{R}} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right].$$

- The determination of the splitting point s can be done very quickly.
- Hence, determining the best pair (j, s) is **feasible**.

Repeat the same process to each block.

Stoping and pruning

- Generally, the process is stopped for a given region when there are **less than** 5 observations in that region.

Problem with previous methodology:

- Likely to **overfit** the data.
- Can lead to poor prediction error.

Pruning the tree. Strategy: Grow a large tree (overfits), and the prune it (better).

- **Weakest link pruning:**

(a.k.a cost complexity pruning)
Let $T \subset T_0$ be a **subtree** of T_0 with $|T|$ **terminal nodes**. For $\alpha > 0$, define:

$$C_\alpha(T) := \sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha \cdot |T|.$$

Pick a subtree minimizing $C_\alpha(T)$.

Stoping and pruning

Pick a subtree $T \subset T_0$ minimizing:

$$C_\alpha(T) := \sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha \cdot |T|.$$

(Here, $\hat{y}_{R_m}$ =average response for observations in R_m.)

- α is a **tuning parameter**.
- Trade-off between fit of the model, and tree complexity.
- Choose α using cross-validation.

Once α has been chosen by CV, use whole dataset to find the tree corresponding to that value.

Classification trees

- So far, we discussed **regression** trees (continuous output).
- We can easily modify the methodology to predict a *categorical* output.
- We only need to modify our *splitting and pruning criteria*.

For continuous variables, we picked a constant in each box R_i to minimize the sum of squares in that region:

$$\min_{c \in \mathbb{R}} \sum_{x_i \in R_i} (y_i - c)^2.$$

As a result, we choose:

$$\hat{c}_i = \frac{1}{N_i} \sum_{x_k \in R_i} y_k,$$

where N_i denotes the number of observations in R_i.

Classification trees

Similarly, when the output is categorical, we can count the proportion of class k observations in node i:

$$\hat{p}_{ik} = \frac{1}{N_i} \sum_{x_l \in R_i} \mathbf{1}_{y_l \in R_i}.$$

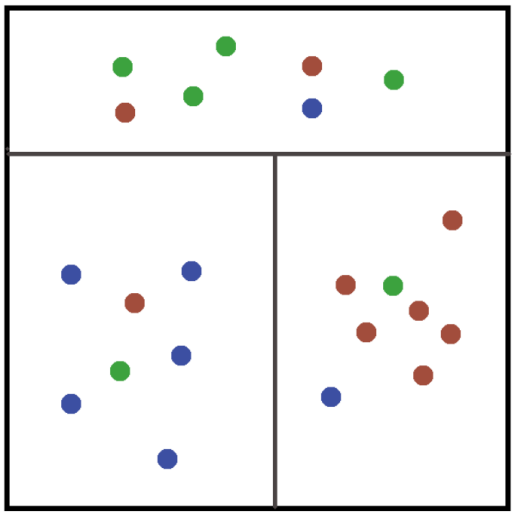
We then classify the observations in node i using a **majority vote**:

$$k(i) := \operatorname*{argmax}_k \hat{p}_{ik}.$$

Different measures are commonly used to determine how good a given partition is (and how to split a given partition):

1. **Misclassification error:** $\frac{1}{N_i} \sum_{x_l \in R_i} \mathbf{1}_{y_l \neq k(i)} = 1 - \hat{p}_{i,k(i)}$.
2. **Gini index:** $\sum_{k=1}^{K} \hat{p}_{ik}(1 - \hat{p}_{ik}) = 1 - \sum_{k=1}^{K} \hat{p}_{ik}^2$.
   (Probability that a randomly chosen point is incorrectly classified.)
3. **Entropy:** $-\sum_{k=1}^{K} \hat{p}_{ik} \log \hat{p}_{ik}$.
   (Measure of "disorder" in a given category.)

Examples



Let us focus on the **top** box.

- (Gini index) Error from classifying according to proportions:

$$P(\text{error}) = P(\text{error}|\text{green})P(\text{green}) + P(\text{error}|\text{blue})P(\text{blue}) + P(\text{error}|\text{red})P(\text{red})$$
$$= 3/7 \cdot 4/7 + 6/7 \cdot 1/7 + 5/7 \cdot 2/7 = 4/7.$$

- (Entropy) The probability distribution associated to the top box: $(4/7, 2/7, 1/7)$.

$$\text{Entropy} = -(4/7)\log_2(4/7) - (2/7)\log_2(2/7) - (1/7)\log_2(1/7) \approx 1.38.$$

Best case possible: $(1,0,0), (0,1,0), (0,0,1)$. Entropy $= 0$.

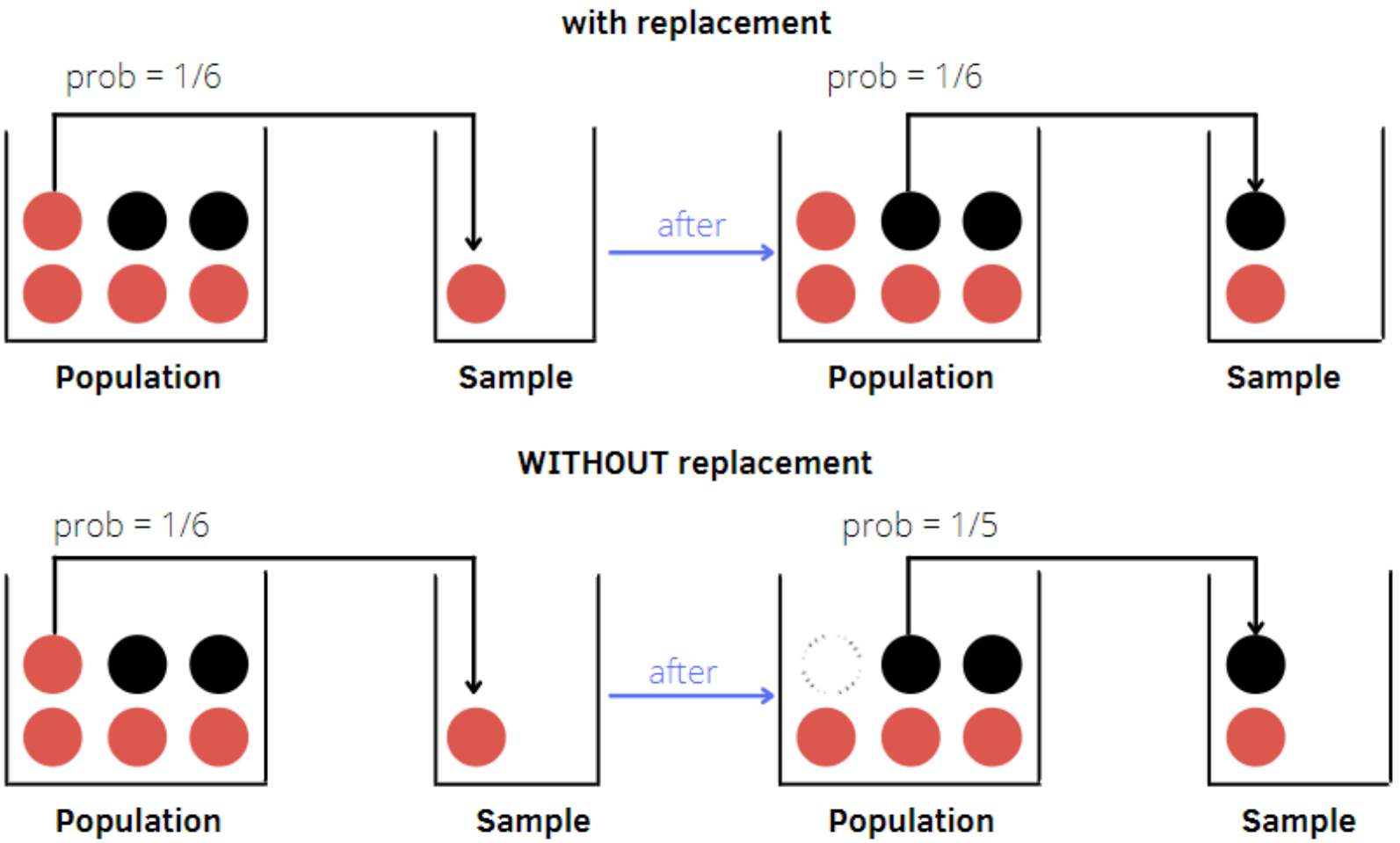
Worst case possible $(1/3, 1/3, 1/3)$. Entropy $= 1.58$.

Lab

- Build a decision tree classifer on the Iris dataset
- Question: Should we use Gini index vs Entropy for the splitting criteria?

Bootstrapping, bagging, random forests

Sampling with replacement



Bootstrap

Bootstrapping: General statistical method that relies on resampling data with replacement.

Idea: Given data (y_i, x_i), $i = 1, \ldots, n$, construct *bootstrap samples* by sampling n of the observations **with replacement** (i.e., allow repetitions):

Sample 1	Sample 2	Sample 3
(y_{i_1}, x_{i_1})	(y_{j_1}, x_{j_1})	(y_{k_1}, x_{k_1})
(y_{i_2}, x_{i_2})	(y_{j_2}, x_{j_2})	(y_{k_2}, x_{k_2})
$\vdots$	$\vdots$	$\vdots$
(y_{i_n}, x_{i_n})	(y_{j_n}, x_{j_n})	(y_{k_n}, x_{k_n})

Bagging

Bagging:(bootstrap aggregation) Suppose we have a model $y \approx \hat{f}(x)$ for data $(y_i, x_i) \in \mathbb{R}^{p+1}$.

1. Construct $B \in \mathbb{N}$ bootstrap samples.
2. Train the method on the b-th bootstrap sample to get $\hat{f}^{*b}(x)$.
3. Compute the average of the estimators:

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{i=1}^{B} \hat{f}^{*b}(x).$$

- Bagging is often used with regression trees.
- Can improve estimators significantly.

Bagging

Note: Each bootstrap tree will typically involve different features than the original, and might have a different number of terminal nodes.

The bagged estimate is the average prediction at x from these B trees.

For classification: Use a majority vote from the B trees.

Random forests

- Idea of bagging: average many noisy but approximately unbiased models, and hence reduce the variance.
- However, the bootstrap trees are generally correlated.
- Random forests improve the variance reduction of bagging by reducing the correlation between the trees.
- Achieved in the tree-growing process through random selection of the input variables.
- Popular method.

Random forests

Random forests: Each time a split in a tree is considered, a random selection of m predictors is chosen as split candidates from the full set of p predictors.

- Typical value for m is $\sqrt{p}$.
- We construct $T_1, \ldots, T_B$ trees using that method on bootstrap samples. The **random forest (regression) predictor** is

$$\hat{f}_{\mathrm{rf}}^{B}(x) = \frac{1}{B} \sum_{b=1}^{B} T_b(x).$$

Advantages

- accurate and robust
- difficult to interpret compared to a decision tree
- does not suffer from the overfitting problem
- usually have built-in relative feature importance

Disadvantages

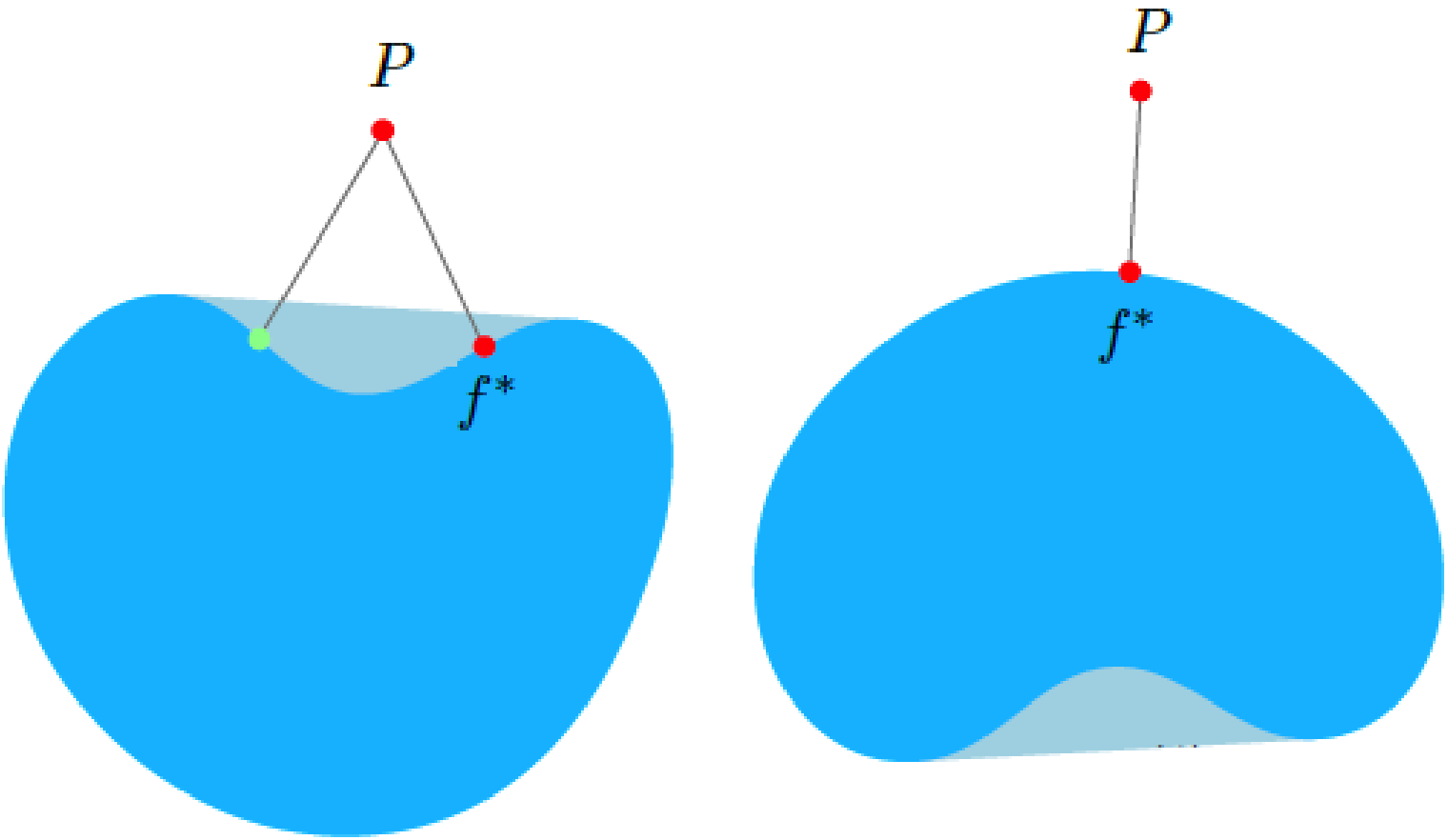
- slow in generating predictions because it has multiple decision trees
- difficult to interpret compared to a decision tree

Demo: Random forests

- use the California housing dataset
- fit a decision tree
- fit a random forest
- investigate feature importance

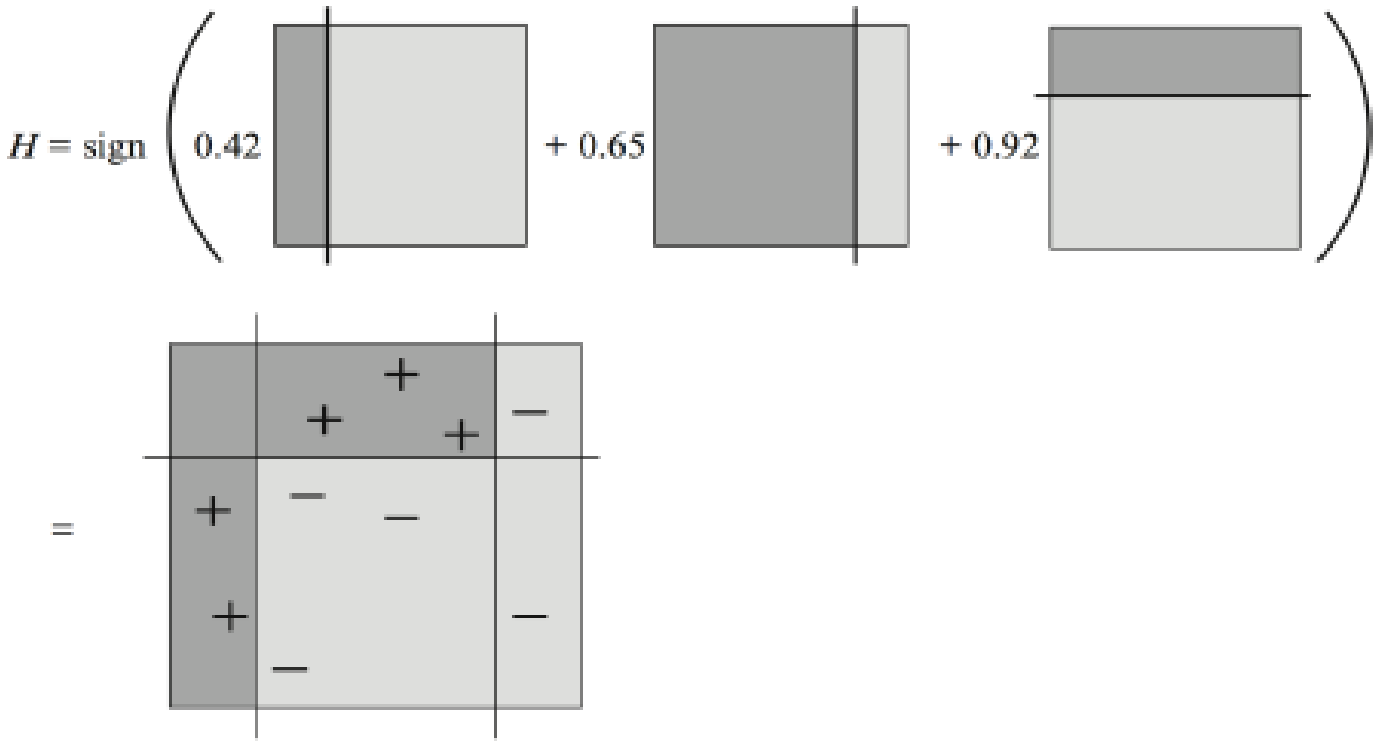
Boosting

Convexification of the hypothesis space



Moving out of the hypothesis space

$$H(x) = \sum_t \rho_t h_t(x)$$

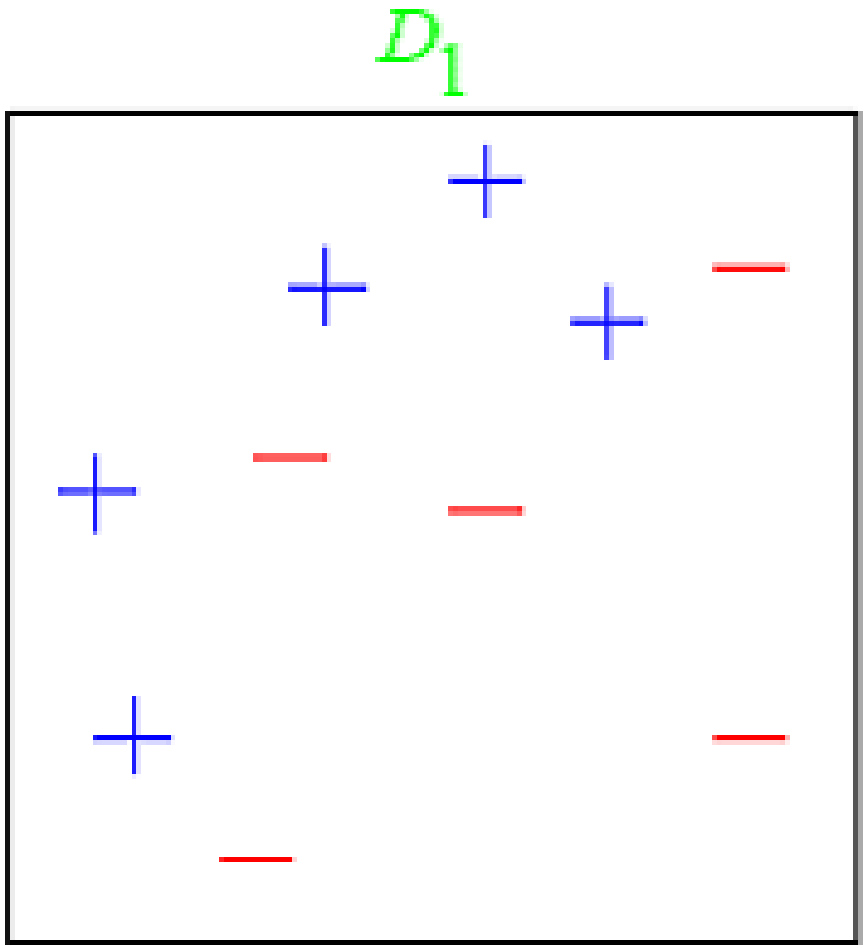


Adaboost

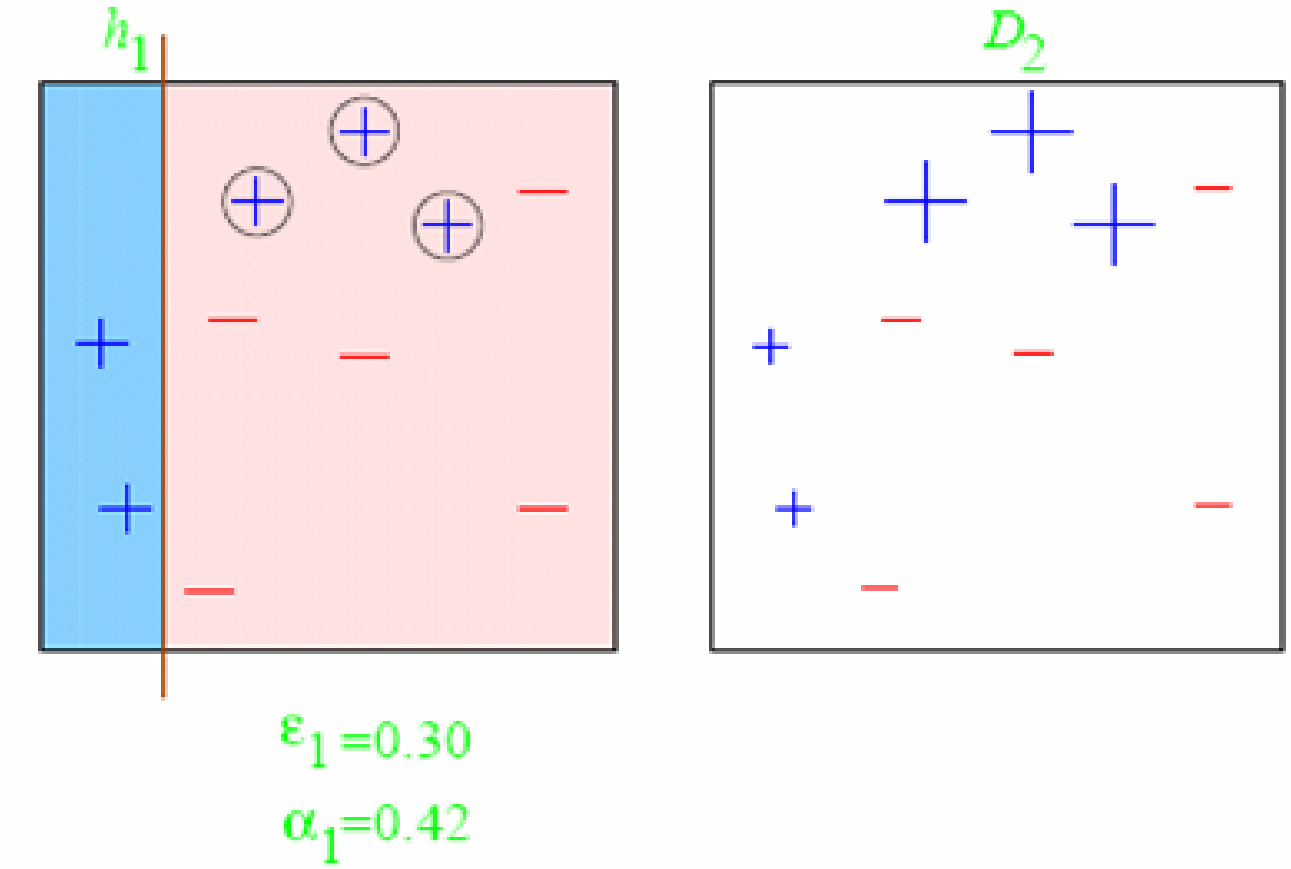
Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

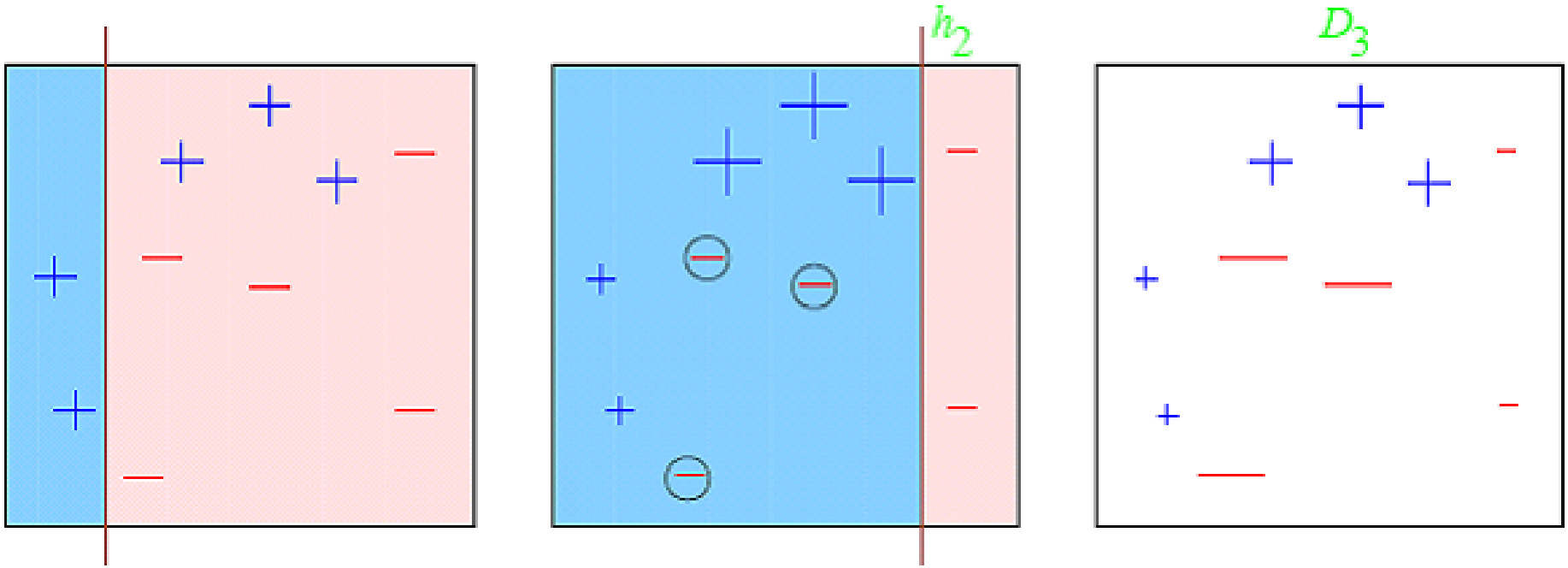
Adaboost



Adaboost



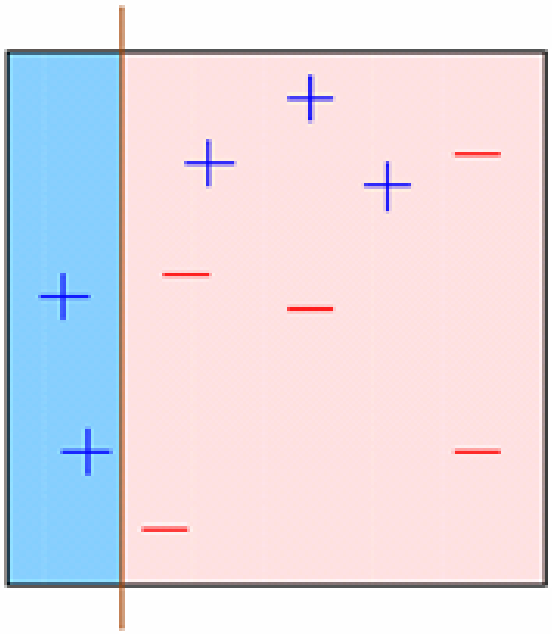
Adaboost

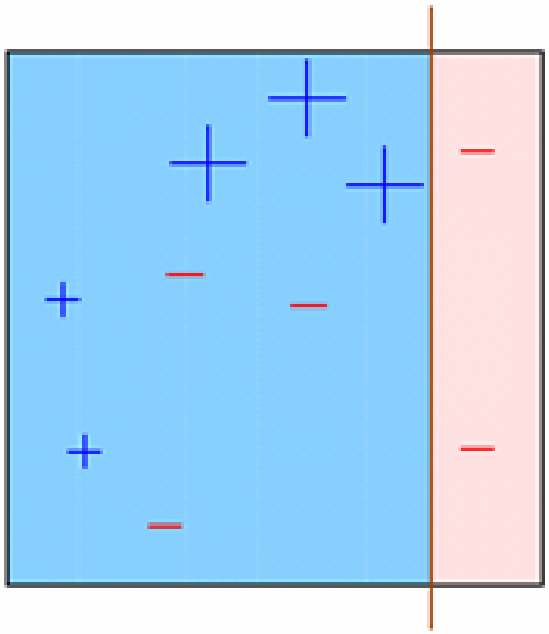


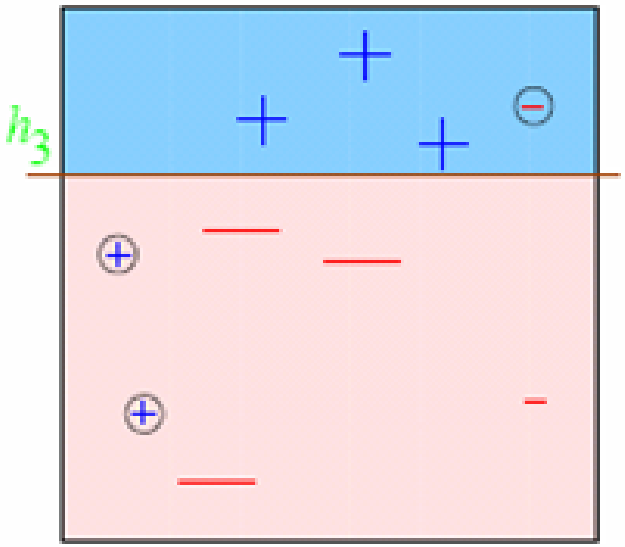
$\varepsilon_2 = 0.21$

$\alpha_2 = 0.65$

Adaboost



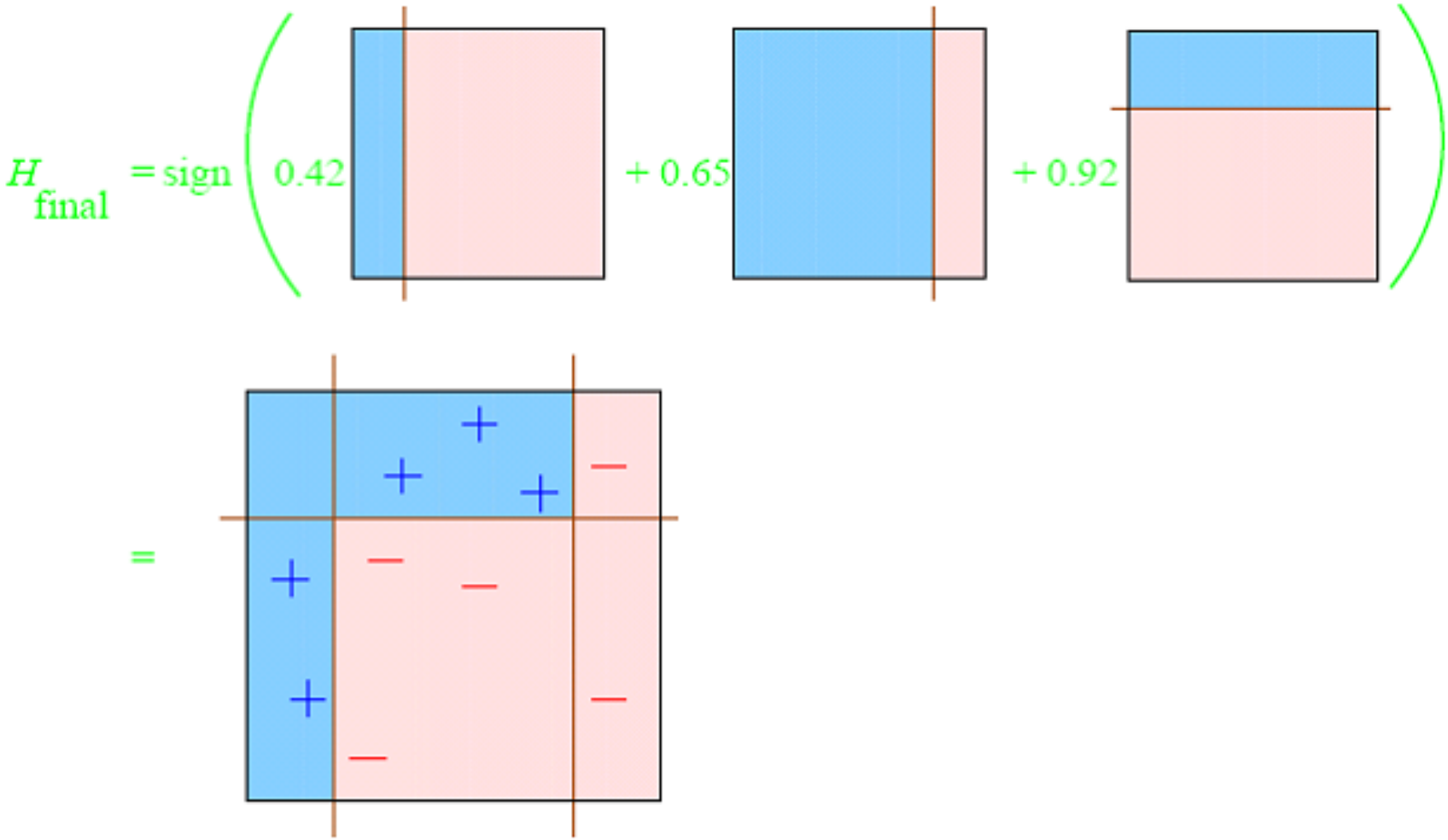




$\varepsilon_3 = 0.14$

$\alpha_3 = 0.92$

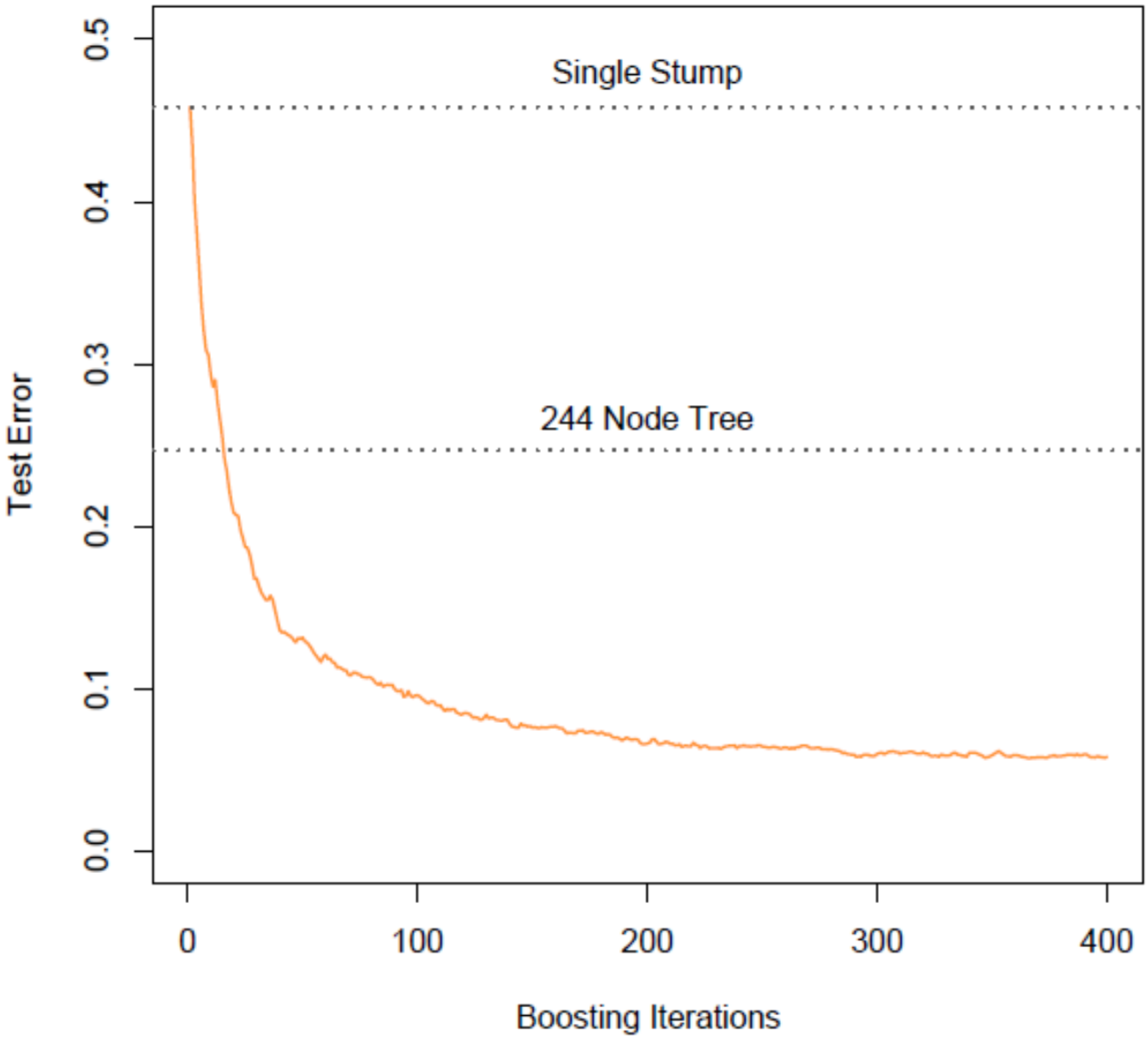
Adaboost



Gradient boosting: history

- Invent Adaboost, the first successful boosting algorithm [Freund et al., 1996, Freund and Schapire, 1997]
- Formulate Adaboost as gradient descent with a special loss function[Breiman et al., 1998, Breiman, 1999]
- Generalize Adaboost to Gradient Boosting in order to handle a variety of loss functions [Friedman et al., 2000, Friedman, 2001]

Gradient boosting



Gradient descent

Gradient Descent

Minimize a function by moving in the opposite direction of the gradient.

$$\theta_i := \theta_i - \rho \frac{\partial J}{\partial \theta_i}$$

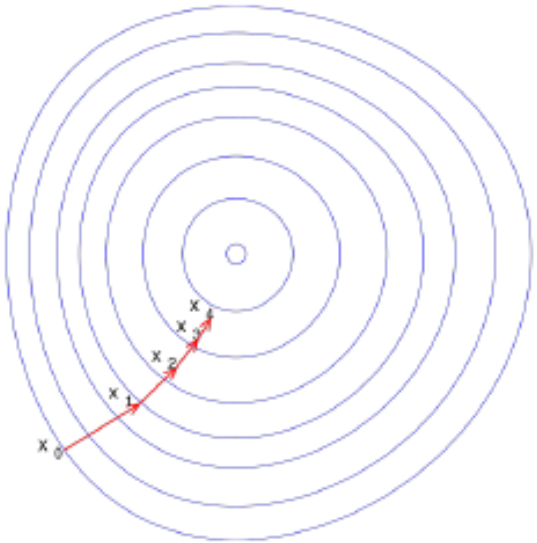
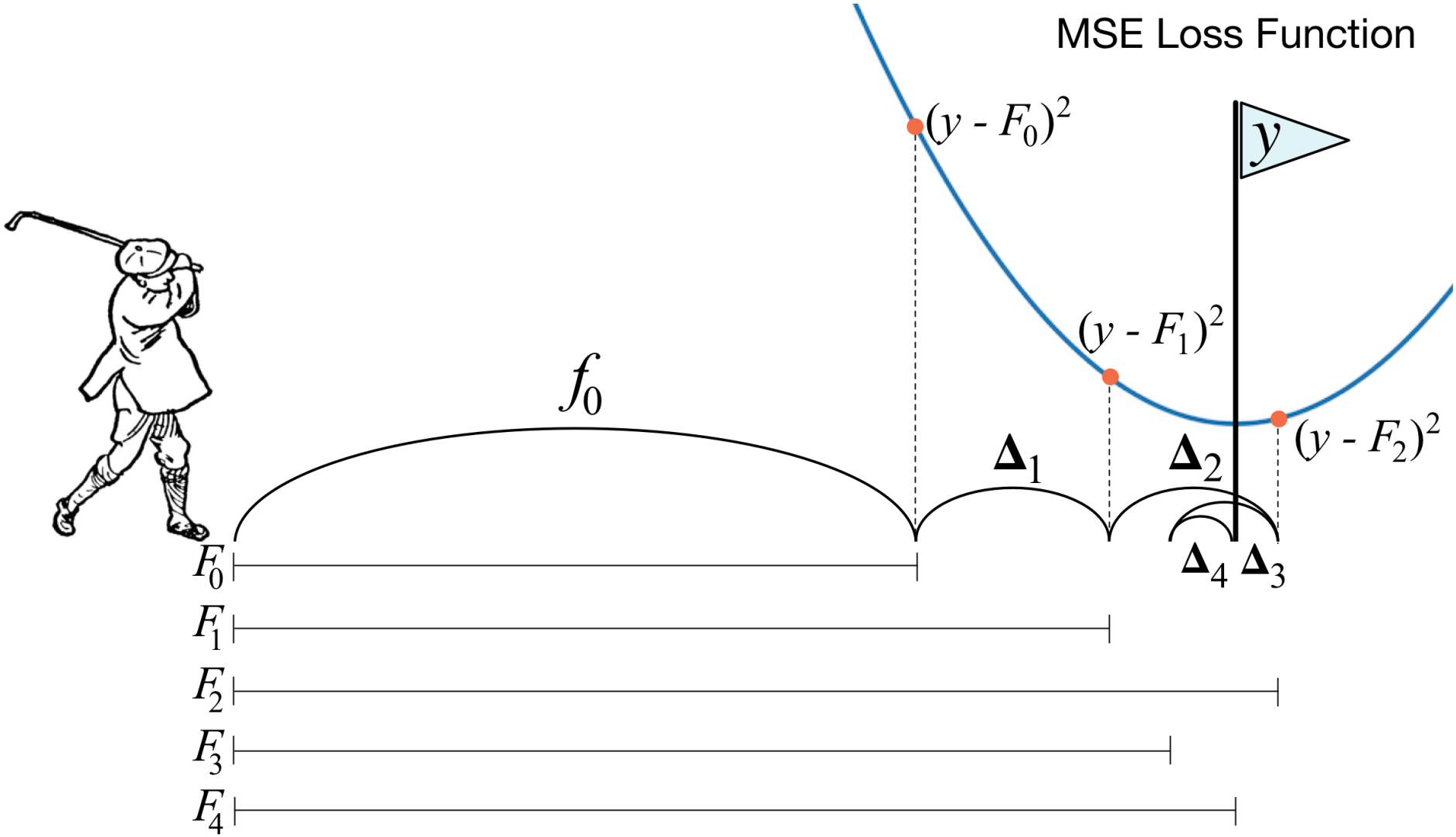


Figure: Gradient Descent. Source:

Gradient boosting



Gradient boosting

Boosting: Recursively fit trees to residuals. (Compensate the shortcoming of previous model.)

Input: $(y_i, x_i) \in \mathbb{R}^{p+1}$, $i = 1, \ldots, n$. Initialize $\hat{f}(x) = 0$, $r_i = y_i$.

For $b = 1, \ldots, B$:

1. Fit a tree estimator $\hat{f}^b$ with d splits to the training data.
2. Update the estimator using:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \cdot \hat{f}^b(x).$$

3. Update the residuals:

$$r_i \leftarrow r_i - \lambda \cdot \hat{f}^b(x_i).$$

Output: Boosted tree:

$$\hat{f}(x) = \sum_{i=1}^{B} \lambda \hat{f}^b(x).$$

Gradient boosting

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$

   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute

$$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$

   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.

Gradient boosting

`sklearn.ensemble` : Ensemble Methods

The `sklearn.ensemble` module includes ensemble-based methods for classification, regression and anomaly detecti

User guide: See the Ensemble methods section for further details.

`ensemble.AdaBoostClassifier` ([…])	An AdaBoost classifier.
`ensemble.AdaBoostRegressor` ([base_estimator, …])	An AdaBoost regressor.
`ensemble.BaggingClassifier` ([base_estimator, …])	A Bagging classifier.
`ensemble.BaggingRegressor` ([base_estimator, …])	A Bagging regressor.
`ensemble.ExtraTreesClassifier` ([…])	An extra-trees classifier.
`ensemble.ExtraTreesRegressor` ([n_estimators, …])	An extra-trees regressor.
`ensemble.GradientBoostingClassifier` ([loss, …])	Gradient Boosting for classification.
`ensemble.GradientBoostingRegressor` ([loss, …])	Gradient Boosting for regression.
`ensemble.IsolationForest` ([n_estimators, …])	Isolation Forest Algorithm
`ensemble.RandomForestClassifier` ([…])	A random forest classifier.
`ensemble.RandomForestRegressor` ([…])	A random forest regressor.
`ensemble.RandomTreesEmbedding` ([…])	An ensemble of totally random trees.
`ensemble.VotingClassifier` (estimators[, …])	Soft Voting/Majority Rule classifier for unfitted estimators.

Gradient boosting

dmlc
XGBoost eXtreme Gradient Boosting

Community | Documentation | Resources | Contributors | Release Notes

XGBoost is an optimized distributed gradient boosting library designed to be highly ***efficient***, ***flexible*** and ***portable***. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boost (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on ma distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples.

License

© Contributors, 2016. Licensed under an Apache-2 license.