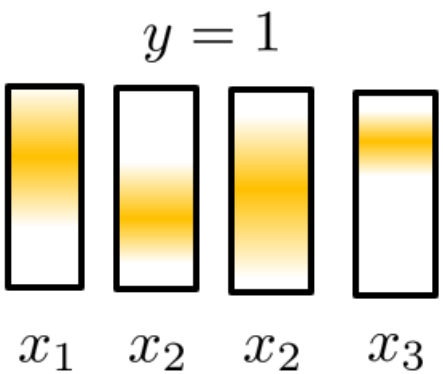
Photogrammetry & Robotics Lab

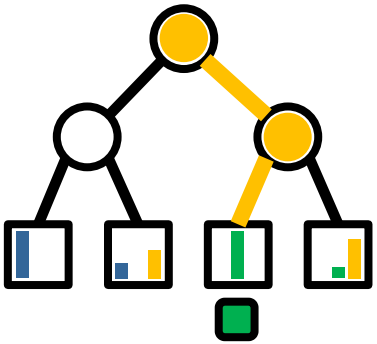
Machine Learning for Robotics and Computer Vision

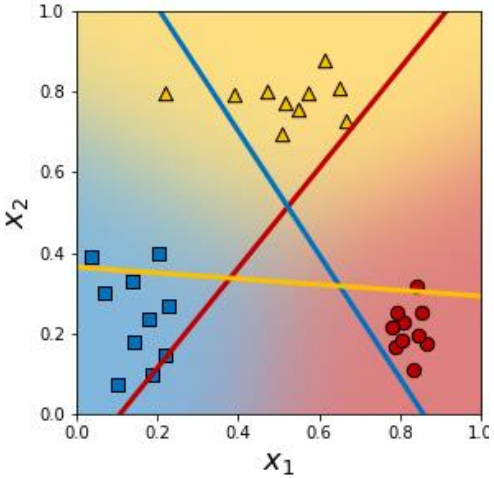
Ensemble Learning

Jens Behley

Recap: Last Lecture







- Classification models
  - Naïve Bayes (Generative Model)
  - Decision Tree (Discriminative Model)
  - Logistic/Softmax Regression (Discriminative Model)

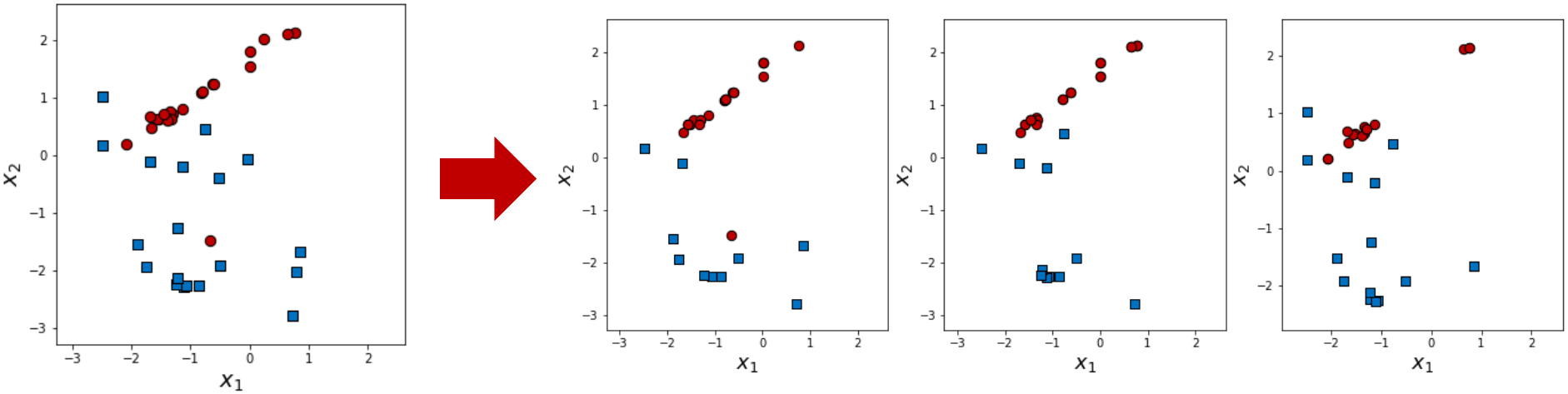
- Optimization with Gradient Descent

Classification with Ensembles

- **Idea:** "Wisdom of the Crowd" approach
- Combine many **weak learners** into a stronger model
- Each individual weak learner inaccurate, but "votes" give better prediction
- Simple ensemble as weighted sum:

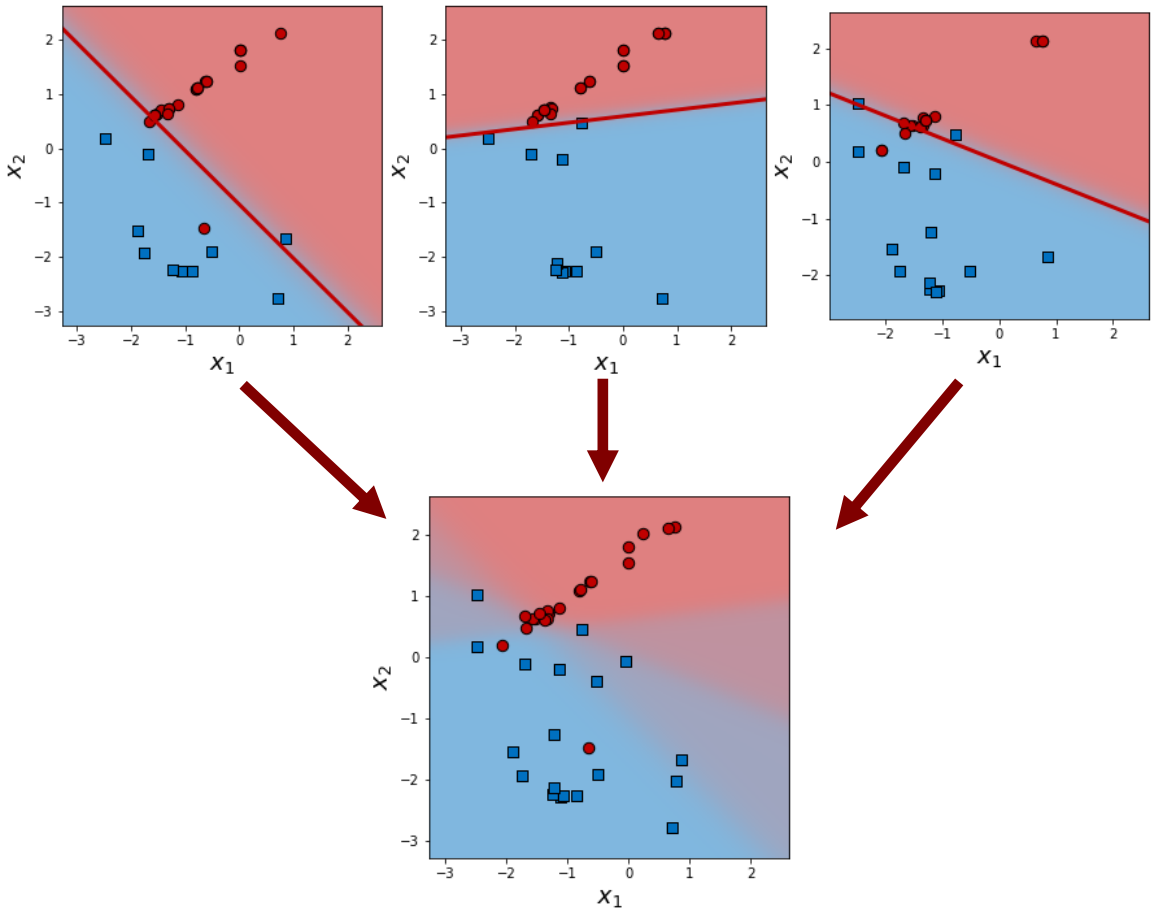
$$P(y|\mathbf{x}) = \sum_j \alpha_j P_j(y|\mathbf{x})$$

Bagging (Boostrap aggregating)



- Sample **with replacement** new datasets
- Train classifiers on new datasets
- Use multiple classifiers $P_j(y|\mathbf{x})$, e.g., Decision Tree, Logistic Regression, …

Aggregated Results

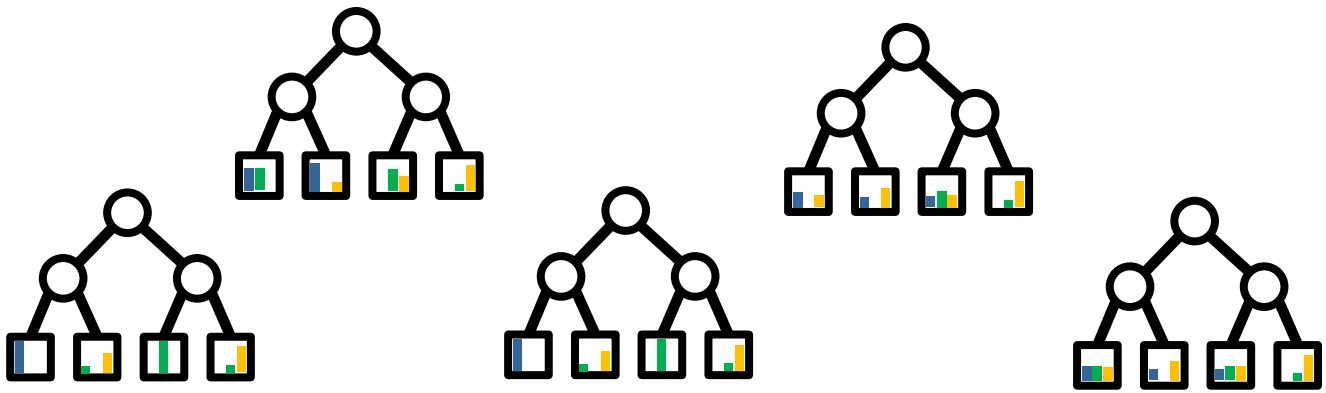


- Ensemble Prediction: $P(y|\mathbf{x}) = T^{-1} \sum_{j=1}^{T} P_j(y|\mathbf{x})$
- Combined predictions can be more accurate

Problems with Bagging

- Correlation between weak learners if sampled subsets are too similar
- Works only if weak learner are “unstable”, e.g., different subsets lead to different training results
- Reduces number of training examples for each weak learner

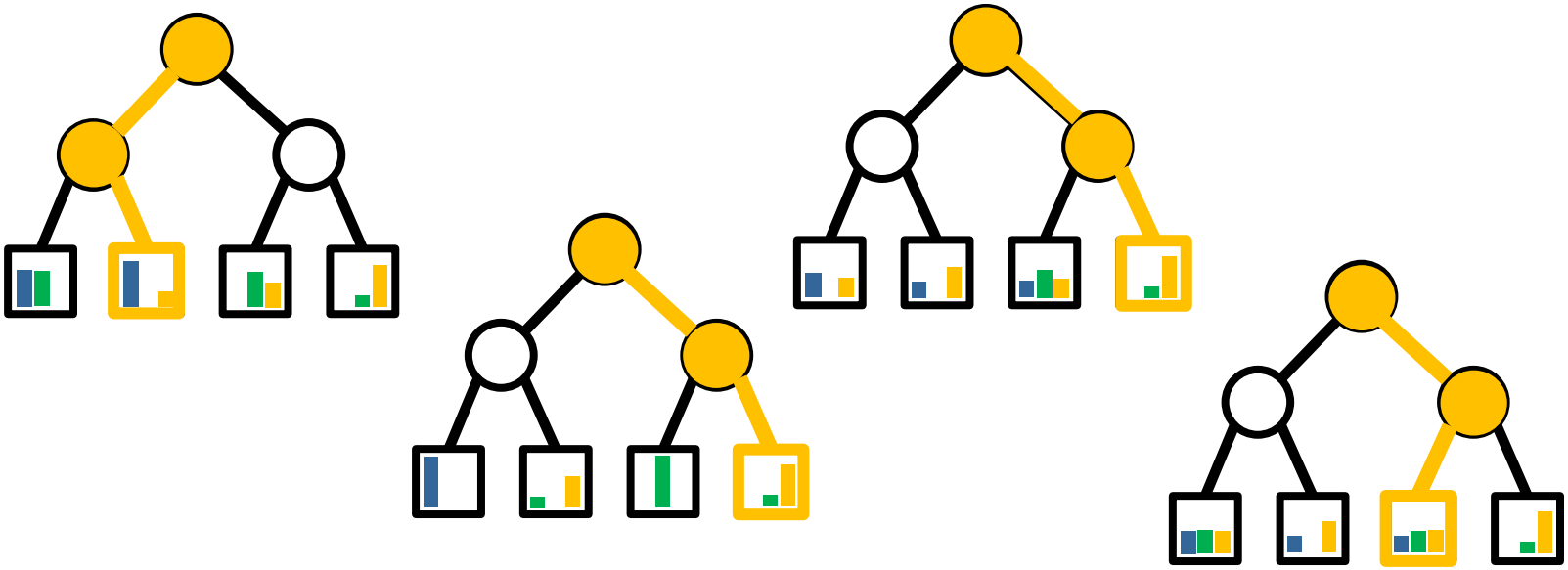
Random Forest



- Use T Decision Trees as weak learners
- Each Decision Tree is randomized by:
  1. Selecting subset of features and split functions
  2. Bagging for each Decision Tree
- Randomization of split functions reduces correlation of individual trees

[Criminisi et al., 2011]

Inference with Random Forest



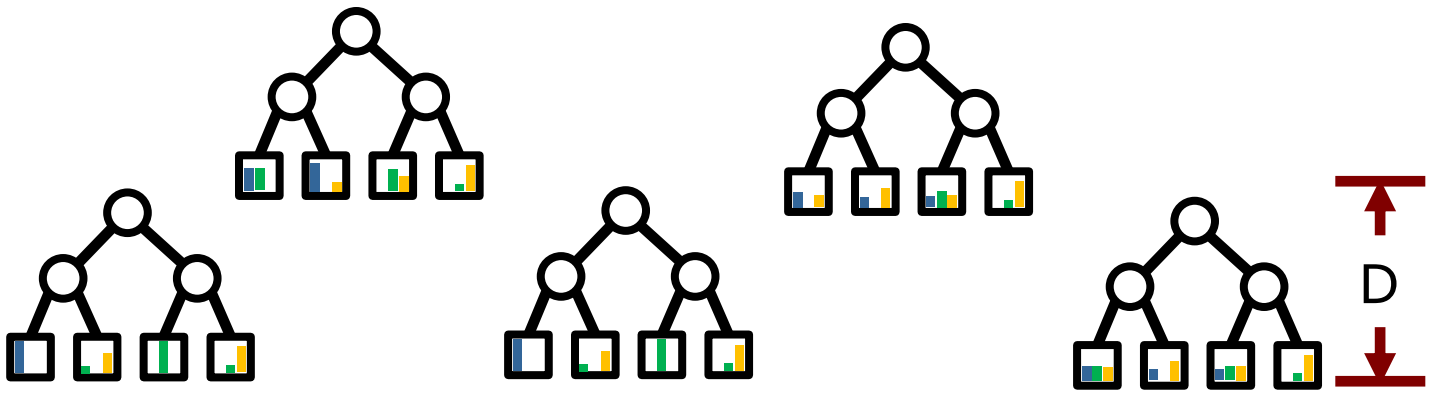
- **Inference**: Evaluate all Decision Tree
- Output:

$$P(y|\mathbf{x}) = \frac{1}{T}\sum_j P_j(y|\mathbf{x})$$

Randomized Decision Trees

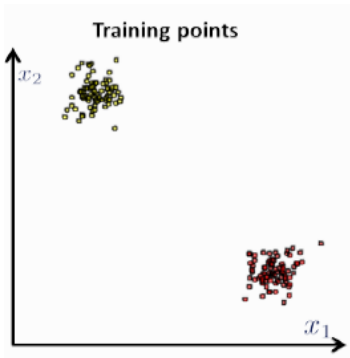
- **CreateNode**($\mathcal{S}$, depth, ρ) ← Randomness
  - Reached max depth: return Leaf($P(y|\mathcal{S})$)
  - Repeat ρ times:
    - Select feature d and example $(\mathbf{x}, y) \in \mathcal{S}$ at random
    - Determine $\mathcal{S}_L, \mathcal{S}_R$ for split function $h(\mathbf{x}|d, x_d)$
    - If $I(\mathcal{S}, \mathcal{S}_L, \mathcal{S}_R) > I^*$:
      - Set $h^*(\mathbf{x}|d^*, x_d^*) = h(\mathbf{x}|d, x_d)$
      - Set $I^* = I(\mathcal{S}, \mathcal{S}_L, \mathcal{S}_R)$
  - node = InnerNode($h^*(\mathbf{x}|d^*, x_d^*)$)
  - node.LeftChild = **CreateNode**($\mathcal{S}_L$, depth + 1)
  - node.RightChild= **CreateNode**($\mathcal{S}_R$, depth + 1)
  - **return** node

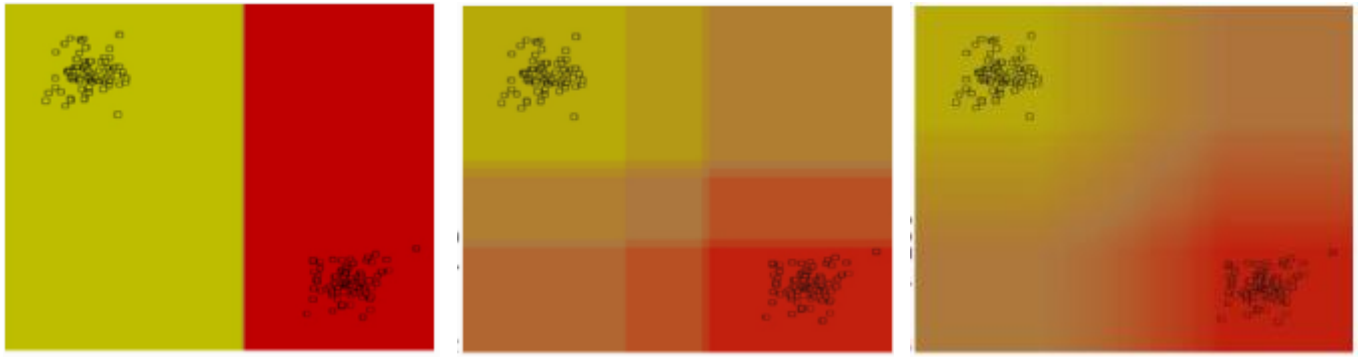
Hyperparameters



- Main hyperparameters of Random Forest
  - Number of Decision Trees T
  - Maximum Depth of Decision Trees D
  - Randomness ρ

Number of Decision Trees T



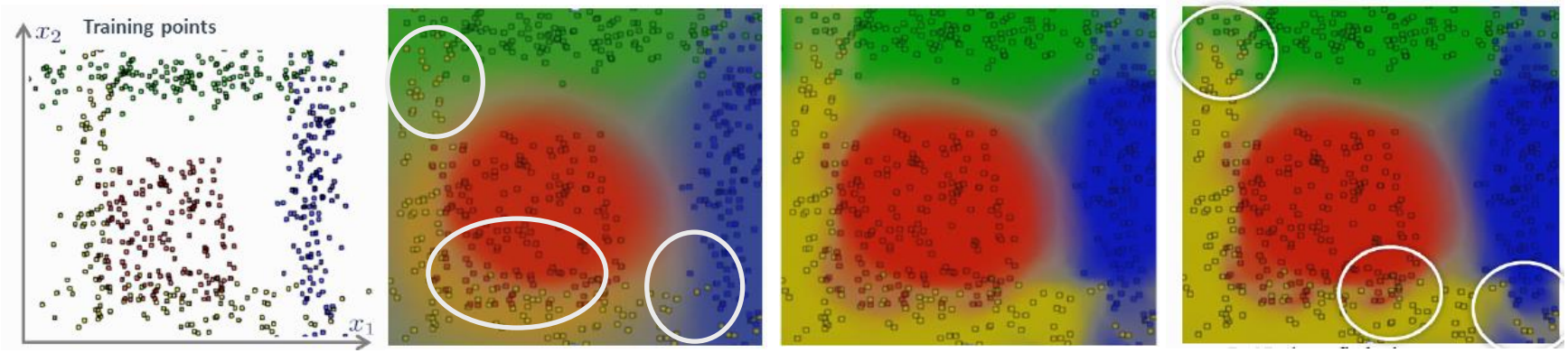


1 Tree 8 Trees 200 Trees

- Increasing number of Decision Trees leads to smoother class posteriors

[Images from Criminisi et al., 2011]

Depth of Decision Trees D

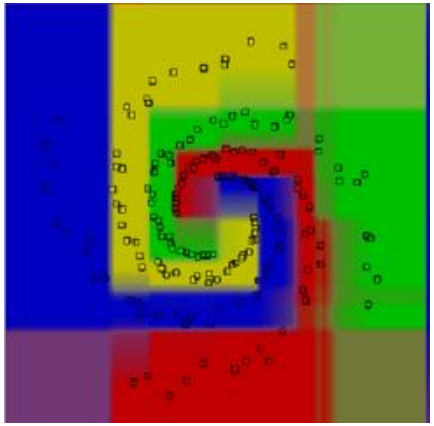


D = 3 (underfitting) D = 6 D = 15 (overfitting)

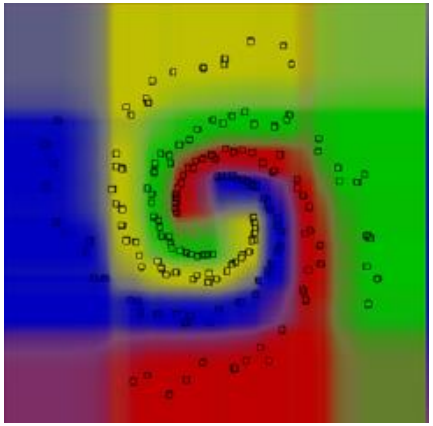
- Depth of Trees influences how well each tree fits the data

[Images from Criminisi et al., 2011]

Randomness



ρ is large,
small randomness



ρ is small,
large randomness

- Larger randomness de-correlates the trees and therefore reduces the overall "agreement" of trees
  → smoother boundaries

[Images from Criminisi et al., 2011]

Decision Forest

- Random Forest also known as **Decision Forest** in Computer Vision community
- Here only randomization via amount of split functions and no bagging
  → each Decision Tree sees all training data

- See [Criminisi et al., 2011] for an excellent introduction to Decision Forests!

Pros & Cons of Random Forests

Advantages

- Highly concurrent training possible
- Decision trees in RF can evaluate features "just-in-time"
- Random forests work well in practice
- Hyperparameters are quite robust

Limitations

- Bagging and Random Forests use same weight for each weak classifier
- Individual models are independent

Boosting

- **General Idea:** Find **weak learner** in stage-wise manner to improve ensemble in each stage ("boost" the performance)

- In boosting, we want to optimize weak learner in ensemble such that the training error is reduced

- **Assumption:** Weak learner must be better then random guess

A Note on Notation

- First, we look at **binary classification**
- **Functional view** of a classification model: We assume that classifier returns {-1, 1}

- Strong classifier $H(\mathbf{x})$ is linear combination of weak classifier $h_i(\mathbf{x})$:

$$H(\mathbf{x}) = \text{sign}\left[\alpha_1 h_1(\mathbf{x}) + \alpha_2 h_2(\mathbf{x}) + \ldots\right]$$

with

- $$\text{sign}(a) = \begin{cases} -1 & , a < 0 \\ 1 & , \text{otherwise.} \end{cases}$$

AdaBoost

- **AdaBoost (adaptive boosting)** is the first popular variant of boosting algorithms for binary classification [Freud & Schapire, 1997]

- **Core Idea:** Use weighted samples for learning in each stage

- At each stage, ensemble gets more accurate

AdaBoost Algorithm

1. Initialize $w_i = \frac{1}{N}$ for $(\mathbf{x}_i, y_i, w_i) \in \mathcal{X}_{\text{train}}$
2. For $m = 1$ to M:
   (a) Train $h_m(\mathbf{x})$ with weighted $\mathcal{X}_{\text{train}}$.
   (b) Determine weighted error:
   $$\epsilon = \sum_{i=1}^{N} w_i \mathbf{1}\{y_i \neq h_m(\mathbf{x}_i)\}$$
   (c) $\alpha_m = \log((1-\epsilon)/\epsilon)$
   (d) $w_i = w_i \cdot \exp(\alpha_m \mathbf{1}\{y_i \neq h_m(\mathbf{x}_i)\})$
   (e) Renormalize w_i
3. Return $H(\mathbf{x}) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})\right]$

Decision Trees as weak learner

- Decision Tree now over weighted samples

1. Weighted information gain:

$$I(\mathcal{S}, \mathcal{S}_L, \mathcal{S}_R) = H(\mathcal{S}) - \sum_{i \in \{L,R\}} \boxed{\frac{\sum_{w_i \in \mathcal{S}_i} w_i}{\sum_{w_i \in \mathcal{S}} w_i}} H(\mathcal{S}_i)$$

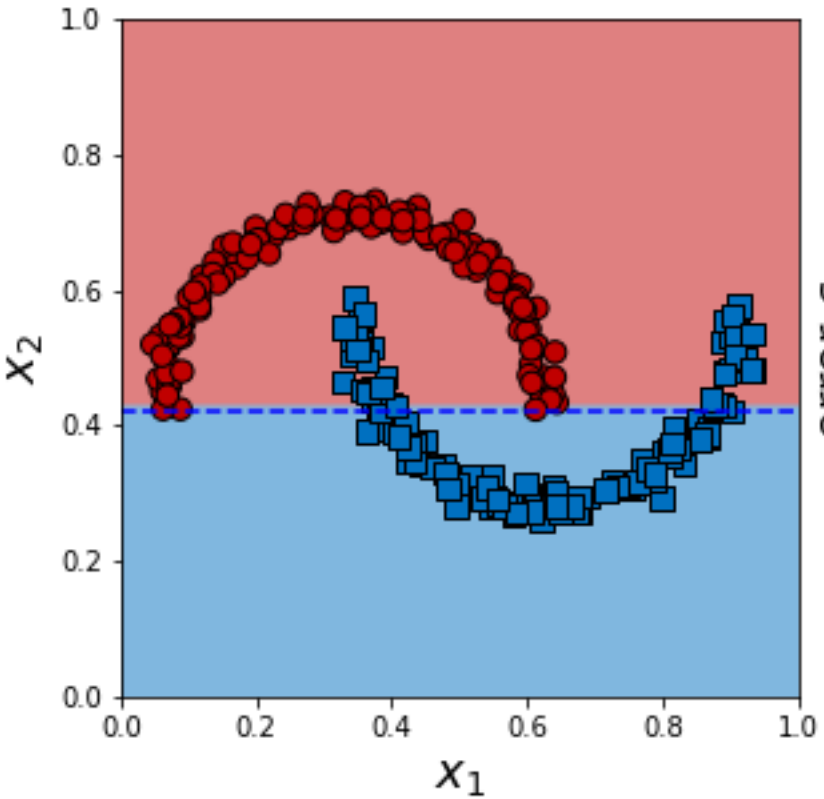
$$H(\mathcal{S}) = -\sum_{k=0}^{K-1} \tilde{P}(y=k) \log_2(\tilde{P}(y=k))$$

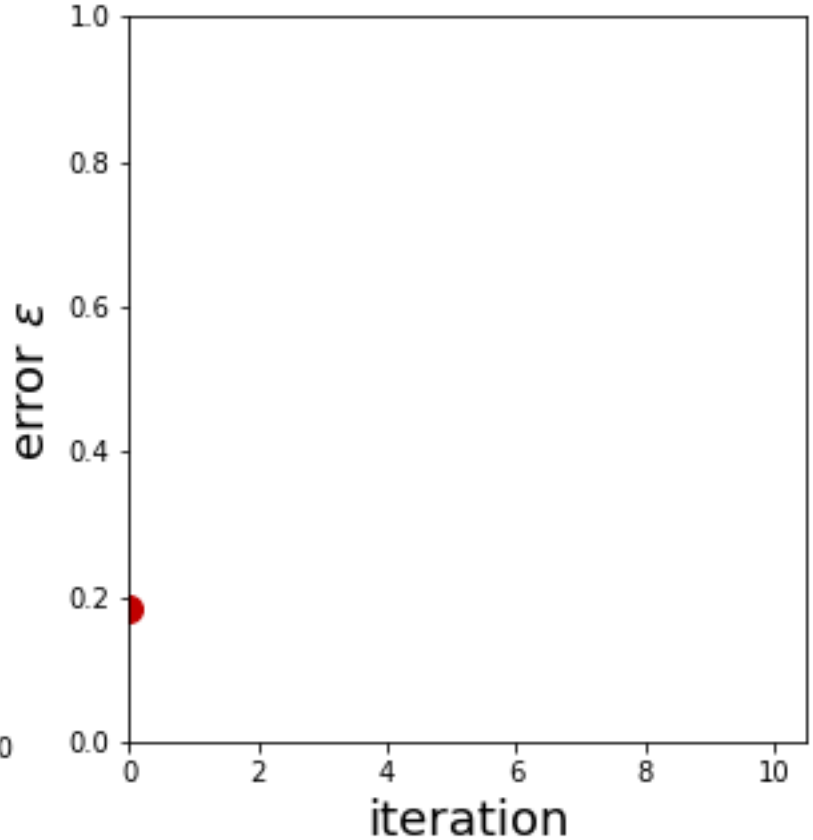
with $\boxed{\tilde{P}(y=k) = \sum_{w \in \{(\mathbf{x},y,w) \in \mathcal{S} | y=k\}} w \cdot \left(\sum_{w \in \mathcal{S}} w\right)^{-1}}$

2. Leaf output accounts for weights

$$\begin{cases} -1 & , \sum_{\{w \in \mathcal{S} | y=-1\}} w > \sum_{\{w \in \mathcal{S} | y=1\}} w \\ 1 & , \text{otherwise} \end{cases}$$

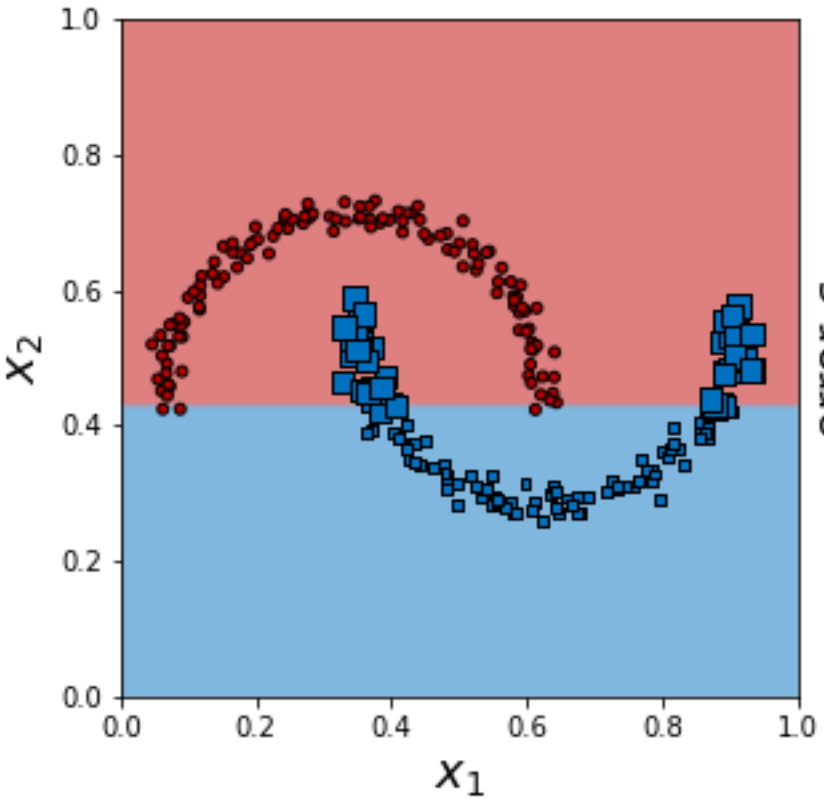
Example (m=1)

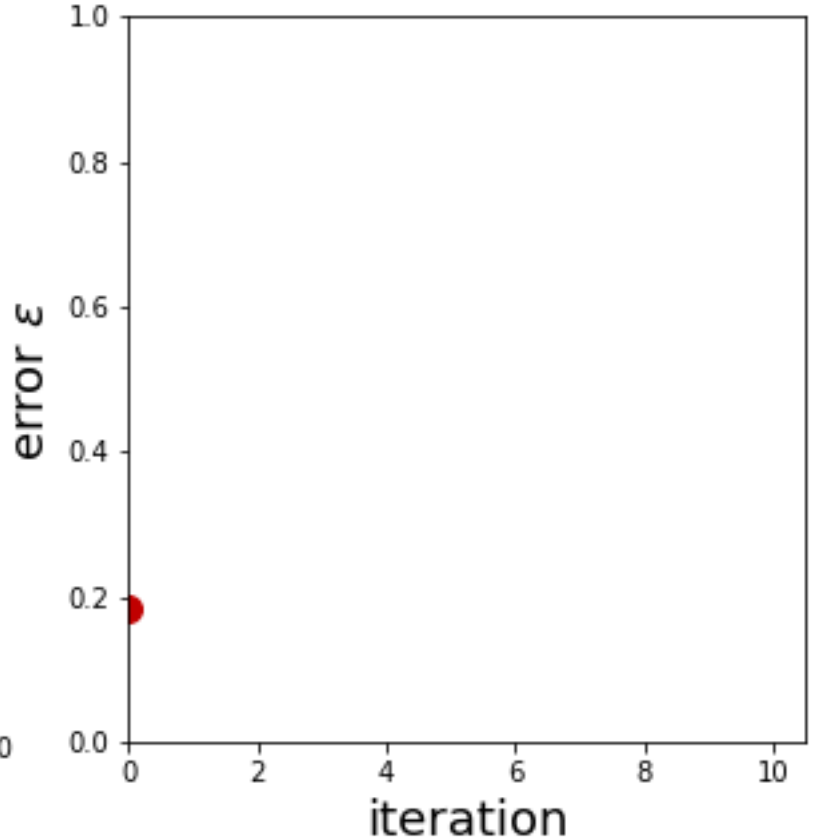




- Weight shown by size of data points
- Initially, all training examples equal weight
- Decision Trees have only **single split node**

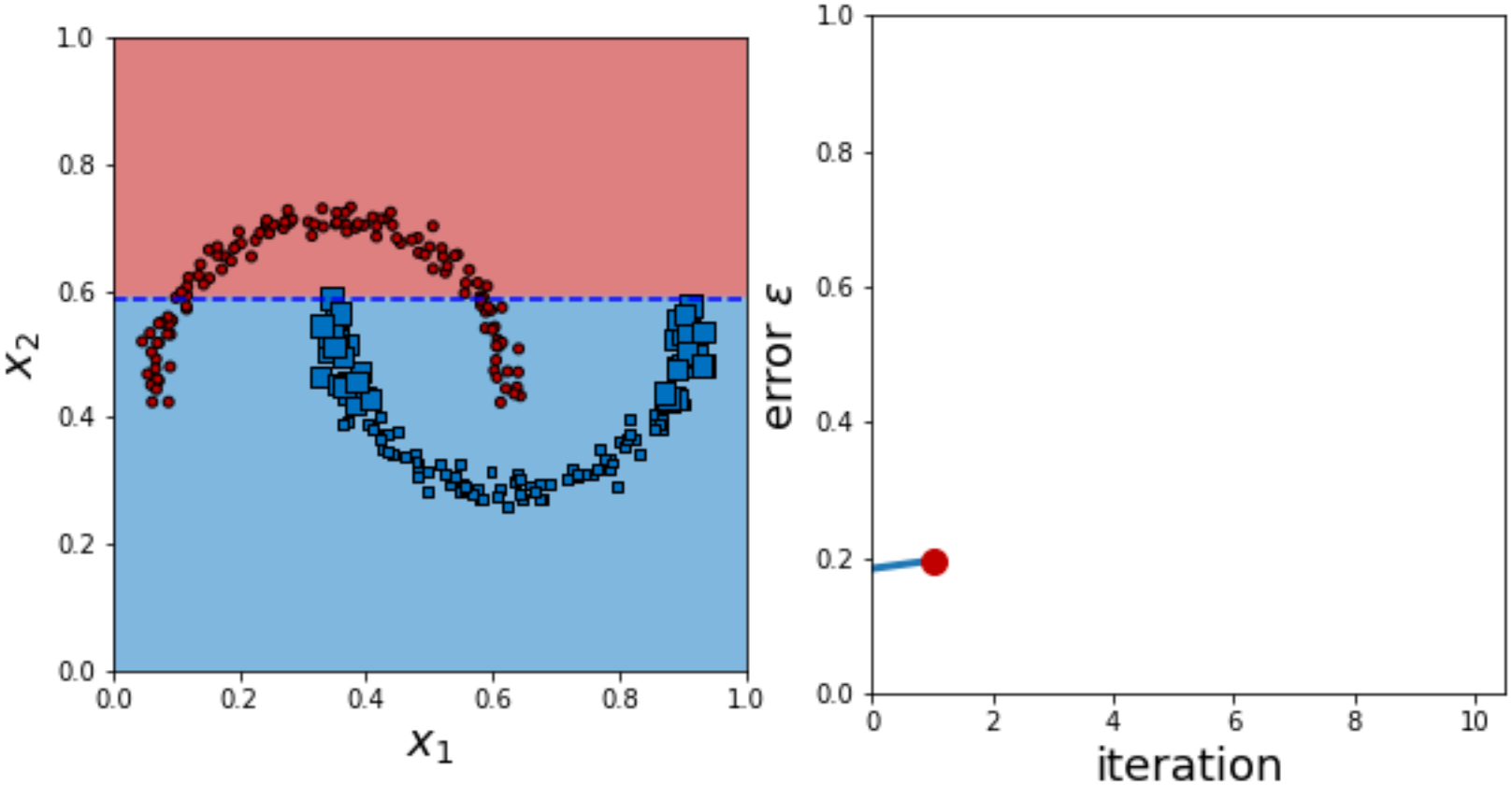
Example (m=1) - Reweighting





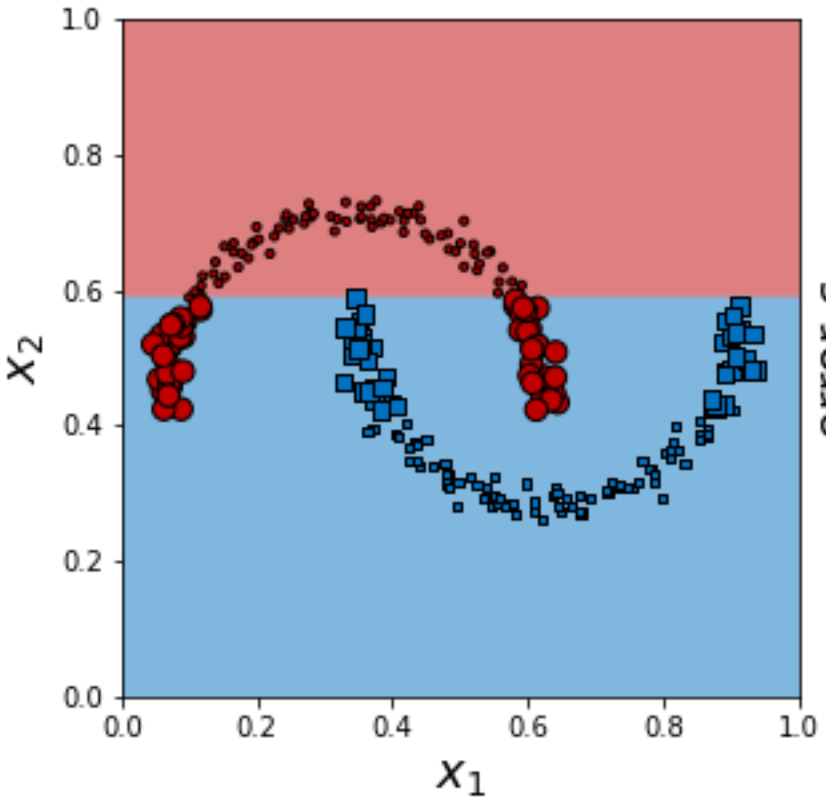
- Weight shown by size of data points
- Weights for examples with wrong class increases!

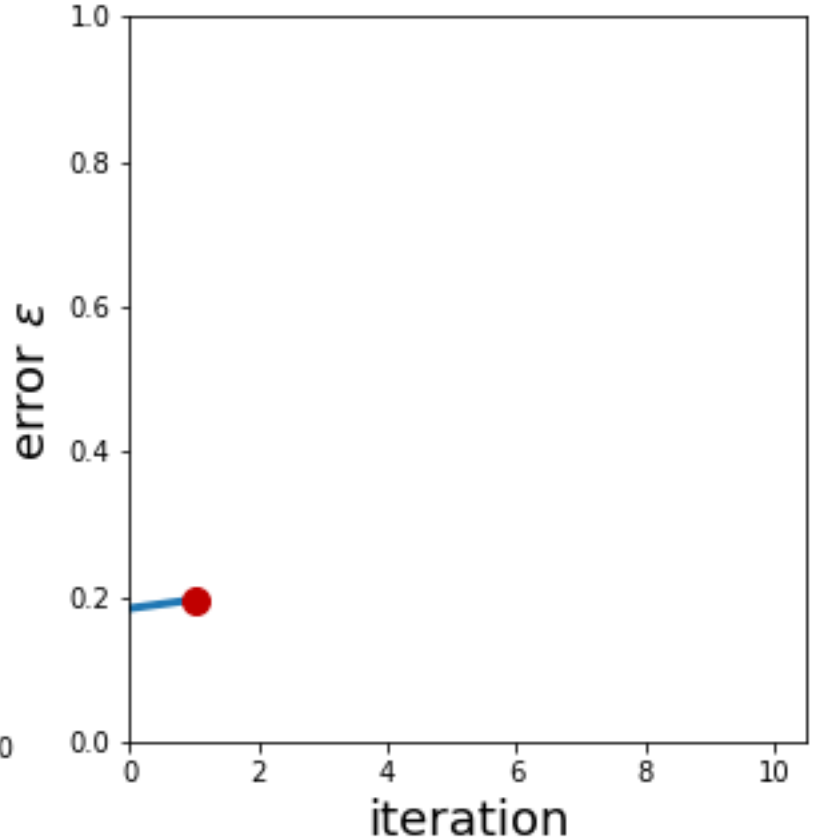
Example (m=2)



- Now process repeats: Fit weak classifier

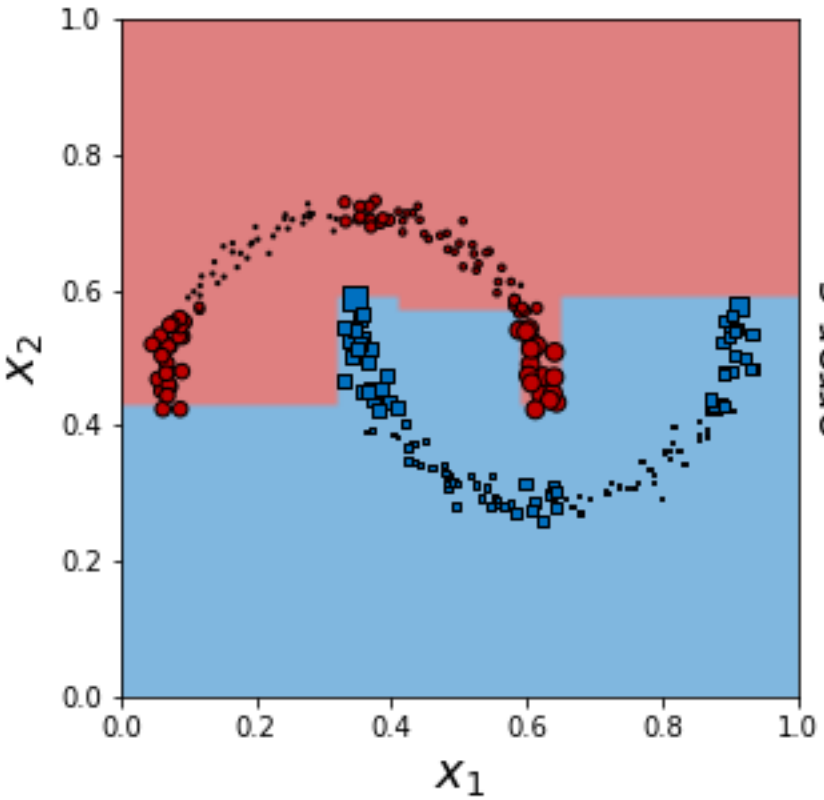
Example (m=2) - Reweighting

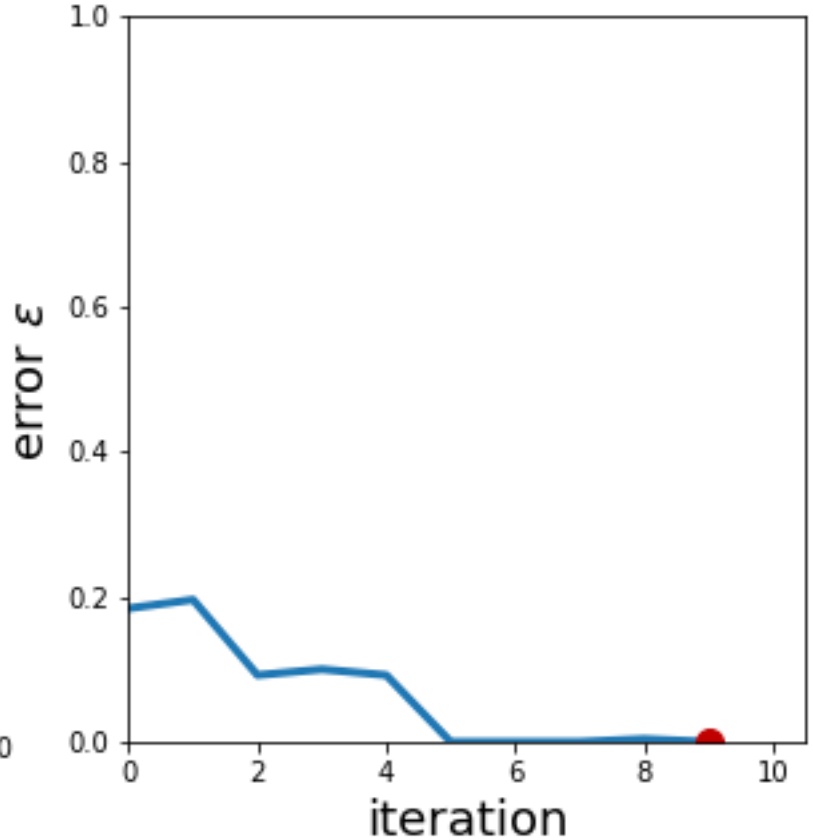




- Now process repeats: Reweight examples

AdaBoost Example (Ctd.)





- Iterate until convergence: Fit & Reweight

Properties of AdaBoost

- **Arbitrary weak learners** possible
  - Only assumption: better than random guess.
- **Cascaded** classification: Evaluate classifiers only if decision can still be changed, i.e., sum of remaining weights is larger then current output:

$$\left|\sum_{m=1}^{k-1} \alpha_m h_m(\mathbf{x})\right| < \sum_{m=k}^{M} \alpha_m$$

- Generally, fast convergence.

General Boosting

- AdaBoost is a specific kind of boosting algorithm
- Underlying assumption: exponential loss

$$\ell(y_i, H(\mathbf{x}_i)) = \exp(-y_i \cdot H(\mathbf{x}_i))$$

- AdaBoost optimizes $H(\mathbf{x})$ to minimize exponential loss on train set

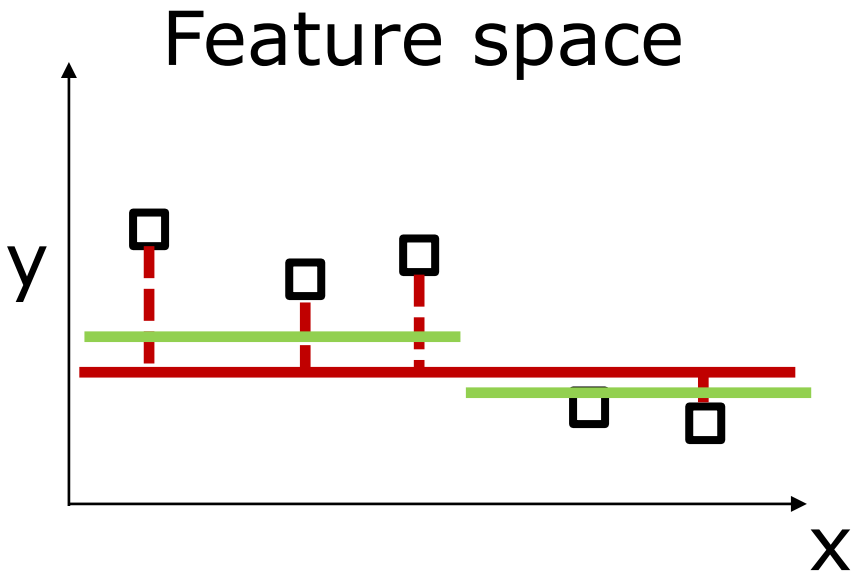
- Can we use this functional optimization also for other losses?

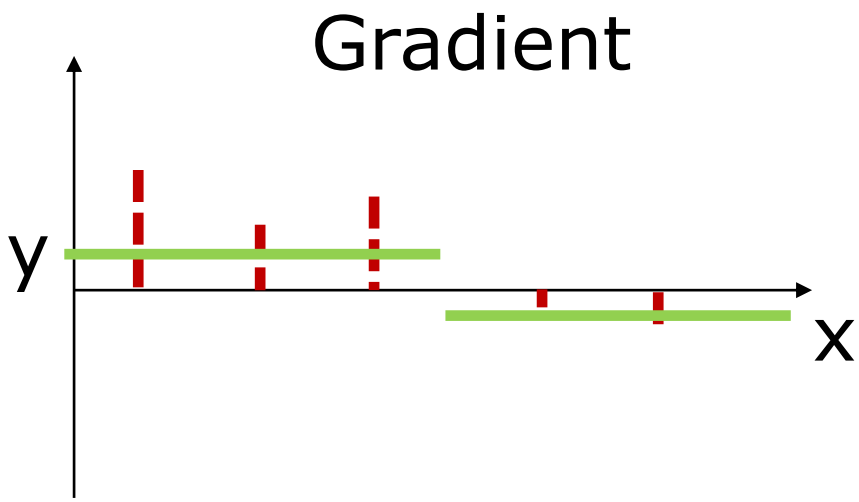
Gradient Boosting

- Gradient Boosting optimizes $H(\mathbf{x})$ in the function space
- **General idea:** Functional optimization similar to gradient descent
- Again, $h_m(\mathbf{x})$ are weak learners, but now **regression models** to approximate gradient

- Gradient Boosting improves $H(\mathbf{x})$ without knowing it's functional shape

Intuition of Gradient Boosting





- Learn approximate gradient via regression model (these are the weak learners $h_m(\mathbf{x})$)
- Gradient Boosting optimizes function via small steps with fitted gradients

General Gradient Boosting

1. $H_0(\mathbf{x}) = \arg\min_c \sum_{i=1}^{N} \ell(y_i, c)$
2. For $m = 1$ to M:

   (a) $\tilde{y}_i = -\left[\frac{\partial \ell(y_i, H(\mathbf{x}_i))}{\partial H(\mathbf{x}_i)}\right]_{H(\mathbf{x})=H_{m-1}(\mathbf{x})}, i = 1, \ldots, N$

   (b) $h_m = \arg\min_h \sum_i \left[\tilde{y}_i - h(\mathbf{x}_i)\right]^2$

   (c) $\alpha_m = \arg\min_\alpha \ell(y_i, H_{m-1}(\mathbf{x}) + \alpha h_m(\mathbf{x}))$

   (d) $H_m(\mathbf{x}) = H_{m-1}(\mathbf{x}) + \alpha_m h_m(\mathbf{x})$

- Here, scale of update α_m of weak learner are optimally selected.

[Friedman, 2001]

Simplified Version

1. $\boxed{H_0(\mathbf{x}) = 0}$
2. For $m = 1$ to M:

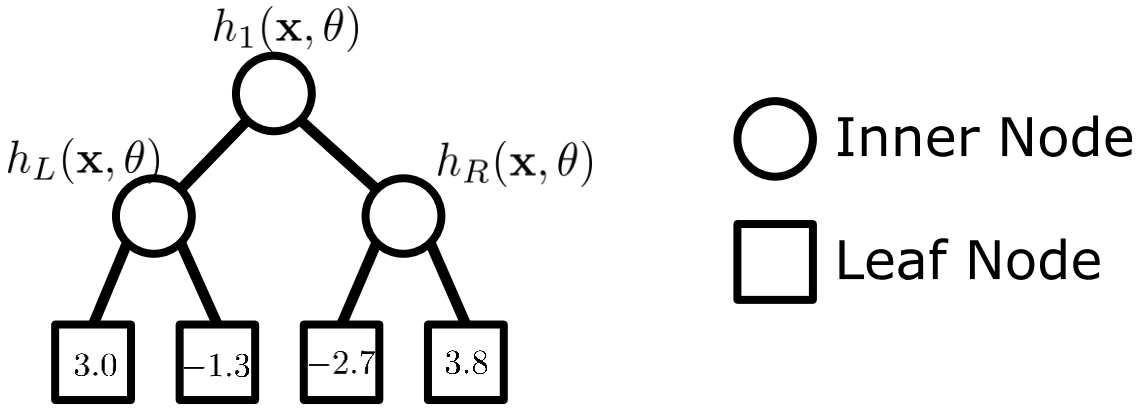
   (a) $\tilde{y}_i = -\left[\frac{\partial \ell(y_i, H(\mathbf{x}_i))}{\partial H(\mathbf{x}_i)}\right]_{H(\mathbf{x})=H_{m-1}(\mathbf{x})}, i = 1, \ldots, N$

   (b) $h_m = \arg\min_h \sum_i \left[\tilde{y}_i - h(\mathbf{x}_i)\right]^2$

   $\boxed{\text{(c) } H_m(\mathbf{x}) = H_{m-1}(\mathbf{x}) + \nu h_m(\mathbf{x})}$

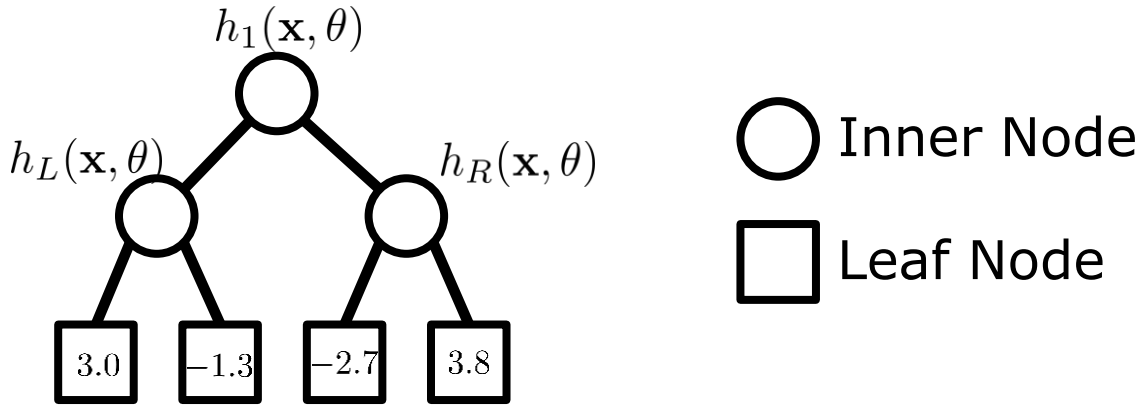
- As long as we perform a step in the right direction (given by the gradient), we are fine with doing a small step by $\nu \in \mathbb{R}$
- $\nu \in \mathbb{R}$ is the learning rate

Gradient Tree Boosting



- Popular choice for Regressor: Decision Tree for regression problems
- Therefore commonly termed "gradient boosted trees" or "Gradient Tree Boosting"

Decision Tree for Regression



- Same algorithm, but now targets $y_i \in \mathbb{R}$
- Leaves return estimate for region
- Training objective: $\sum_{(\mathbf{x}_i, y_i) \in \mathcal{X}_{\text{train}}} (y_i - H(\mathbf{x}_i))^2$
- Split functions still threshold function:

$$h(\mathbf{x}|d, \tau) = \begin{cases} x_d < \tau, & 0 \\ x_d \geq \tau, & 1 \end{cases}$$

Building a Regression Tree

- Same as before: Recursively split examples via split functions at inner nodes.
- Estimated value of examples reaching leaf:

$$\bar{y} = \frac{1}{|\mathcal{S}|} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{S}} y_i$$

- To determine quality of split $Q(\mathcal{S}_L, \mathcal{S}_R)$, we can use the square loss with respect to the average $\bar{y}$:

$$Q(\mathcal{S}_L, \mathcal{S}_R) = \sum_{(\mathbf{x}, y) \in \mathcal{S}_L} (y - \bar{y})^2 + \sum_{(\mathbf{x}, y) \in \mathcal{S}_R} (y - \bar{y})^2$$

Gradient Tree Boosting for multi-class classification

- Use k different ensembles for each class:

$$P(y|\mathbf{x}) = \mathrm{softmax}(H^{(1)}(\mathbf{x}), \ldots, H^{(K-1)}(\mathbf{x}))$$

$$\Rightarrow \quad P(y = k|\mathbf{x}) = \frac{\exp(H^{(k)}(\mathbf{x}))}{\sum_j \exp(H^{(j)}(\mathbf{x})}$$

- Negative log-likelihood on softmax results in loss: **cross-entropy loss**

$$\ell(y_i = k, H^k(x)) = -\log \frac{\exp(H^k(\mathbf{x}))}{\sum_j \exp(H^j(\mathbf{x}))}$$

$$= -H^k(\mathbf{x}) + \log \sum_j \exp(H^j(\mathbf{x}))$$

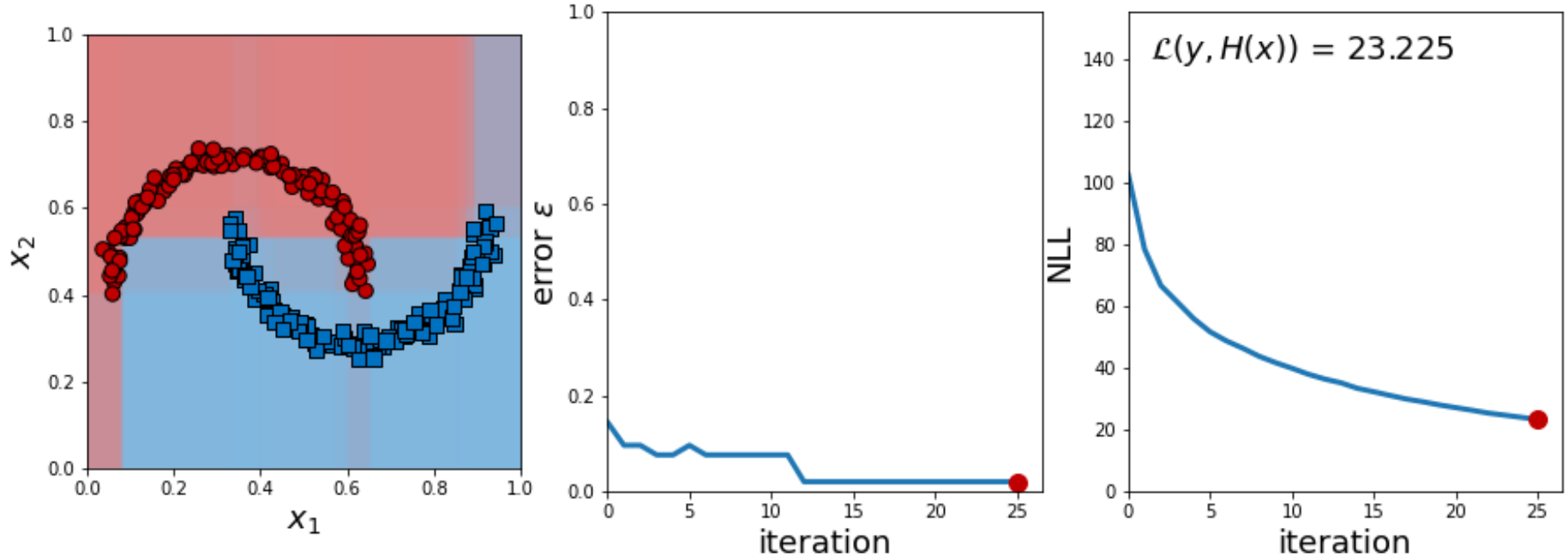
Gradient on NLL

- We already showed that gradient of NLL is given by:

$$\frac{d\ell\left(y_i, H^{(k)}(\mathbf{x})\right)}{dH(\mathbf{x})} = H^{(k)}(\mathbf{x}) - \mathbf{1}\{y_i = k\}$$

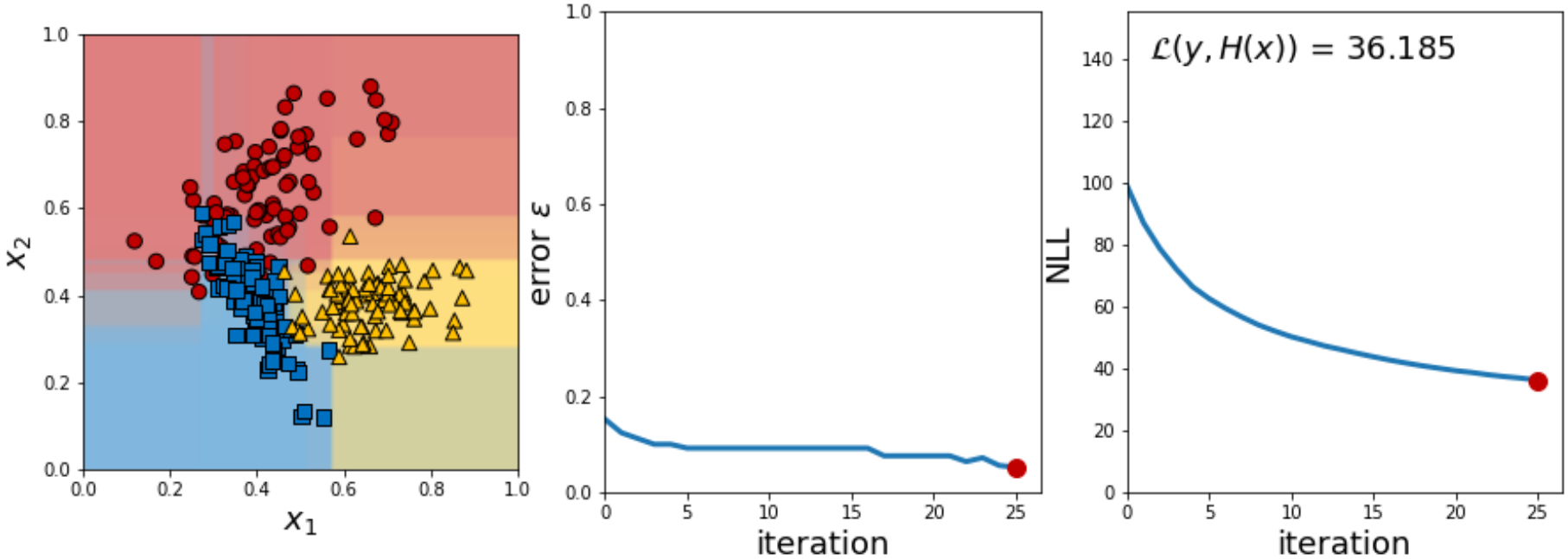
- Similarly as with softmax regression, but now change of function $H^{(k)}(\mathbf{x})$ needed to decrease NLL loss

Gradient Boosting Example



- Gradient Boosting with Regression Tree (d=1), i.e., single split node
- NLL decreases with added weak learners

Gradient Boosting Multi-class



- Example of multi-class problem
- Gradient Boosting with Regression Tree (d=1), i.e., single split node

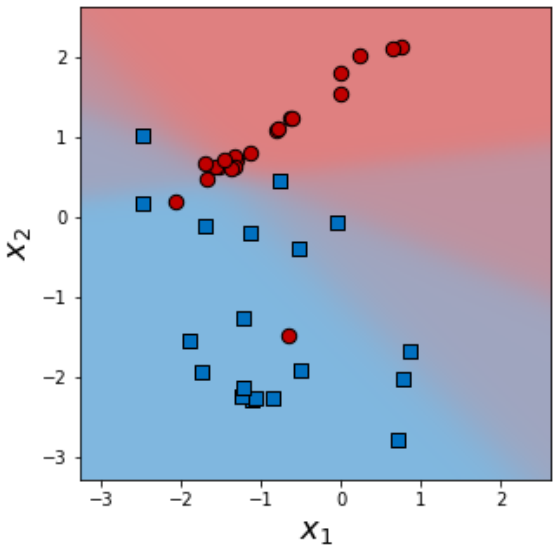
Strong Off-the-shelf models

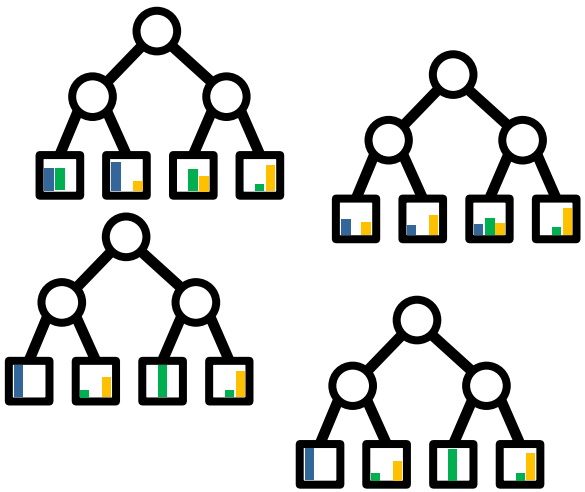
- Random Forests and Gradient Boosting are strong **off-the-shelf models** that work well in general

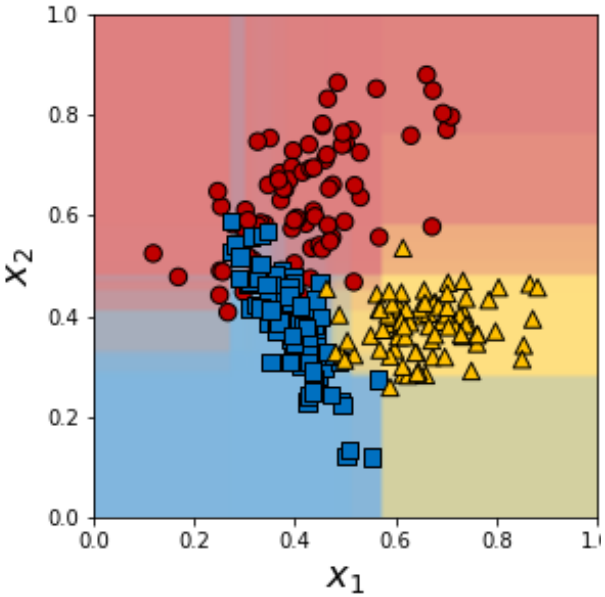
- **Gradient Boosting** is a popular choice in several Kaggle challenges
  → top performer in several competitions

- In practice (set according to validation set):
  - Decision/Regression Trees of depth 4-6
  - Several hundred rounds of boosting

Summary







- Discussed ensemble methods:
  - Bagging
  - Random Forests
  - Boosting: AdaBoost and Gradient Tree Boosting
- Powerful off-the-shelf methods

References

- Breiman, “Bagging predictors”, Machine Learning, 1994
- AdaBoost
- Criminisi et al. “Decision Forests: A Unified Framework […]”, Fond. & Trends in CG & CV, 2011
- Freund & Schapire. “A decision-theoretic generalization of on-line learning and an application to boosting”. J. of Comp. and Sys. Sci. 55: 119–139, 1997.
- Friedman, “Greedy Function Approximation: A Gradient Boosting Machine”, Ann. of Stat., 29(5): 1189-1232, 2001.

- Hastie, Tibshirani, Friedman. “The Elements of Statistical Learning” (Second Ed.), Springer, 2009. (see https://web.stanford.edu/~hastie/ElemStatLearn/)

See you next week!