Upcoming Deadlines

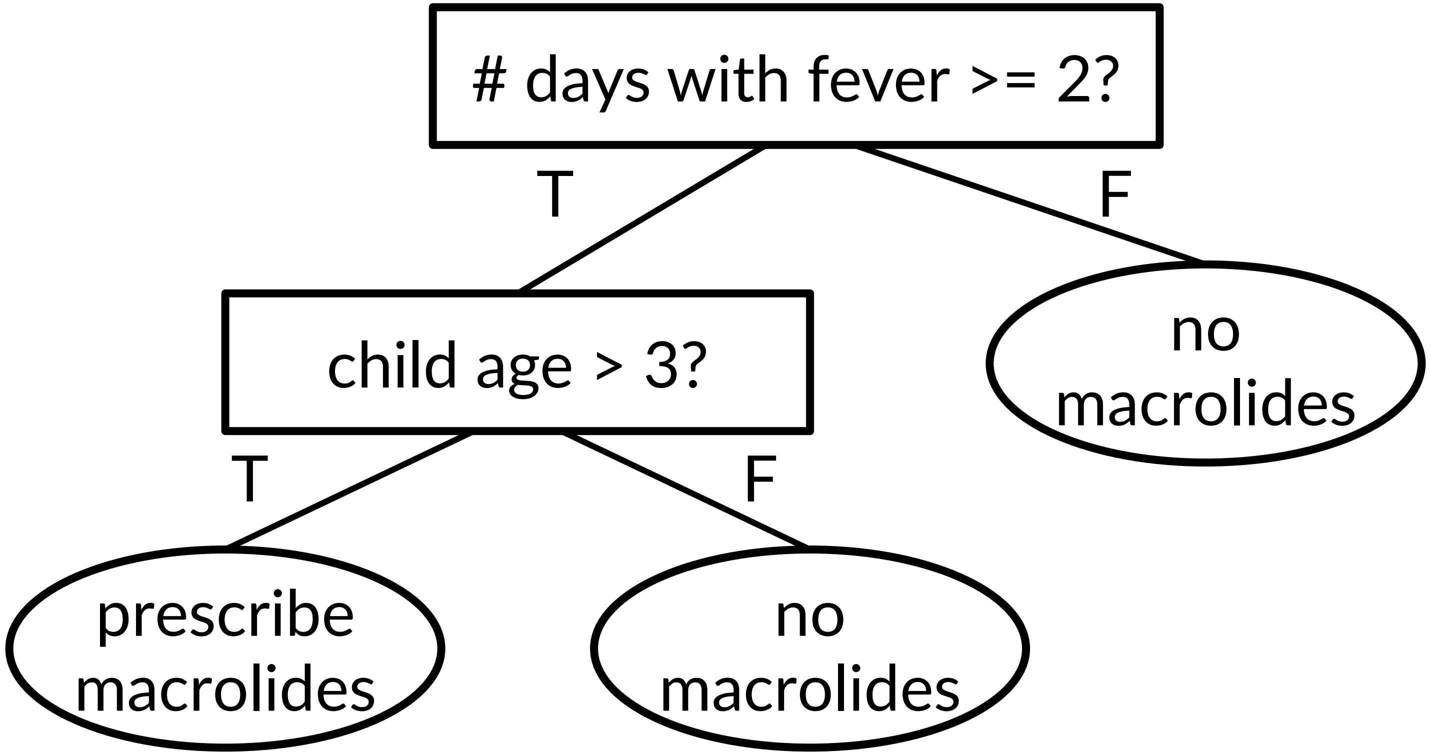
- Quiz 4 due 10/5 at 8pm
- HW 3 due 10/11 at 8pm
- Project details on Wednesday

Lecture 9: Learning Ensembles

CIS 4190/5190

Fall 2023

Decision Tree Shortcomings



Decision tree example from: Martignon and Monti. (2010). Conditions for risk assessment as a topic for probabilistic education. *Proceedings of the Eighth International Conference on Teaching Statistics* (ICOTS8).

Decision Tree Shortcomings

- Hard to manage bias-variance tradeoff
  - Small depth → High bias, low variance
  - Large depth → Small bias, high variance

Post Pruning

def PostPruneTree$(T, Z_{\text{train}}, Z_{\text{val}})$:

for each internal node N of T:

$T_N \leftarrow \text{Replace}\left(T, N, \text{LeafNode}\left(\text{Mode}(Z_{\text{train}}[N])\right)\right)$

$g_N \leftarrow \text{Loss}(T, Z_{\text{val}}) - \text{Loss}(T_N, Z_{\text{val}})$

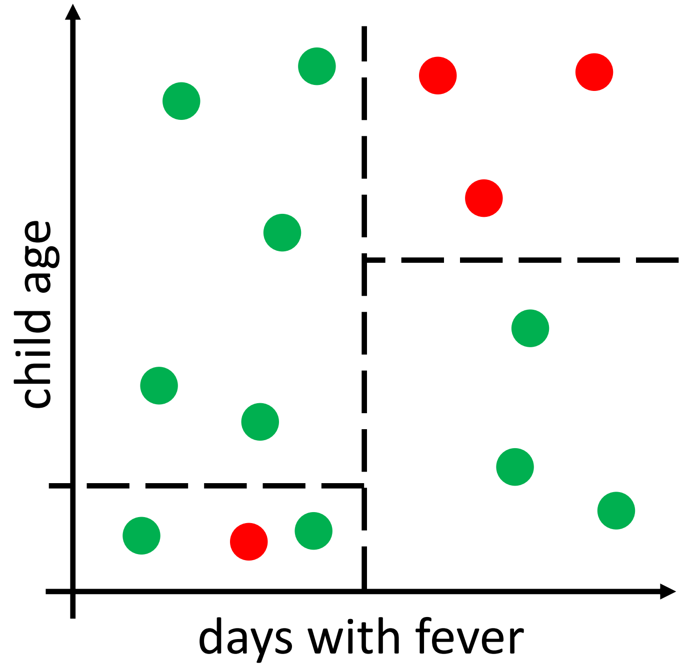
$N_0 \leftarrow \arg\max_N g_N$

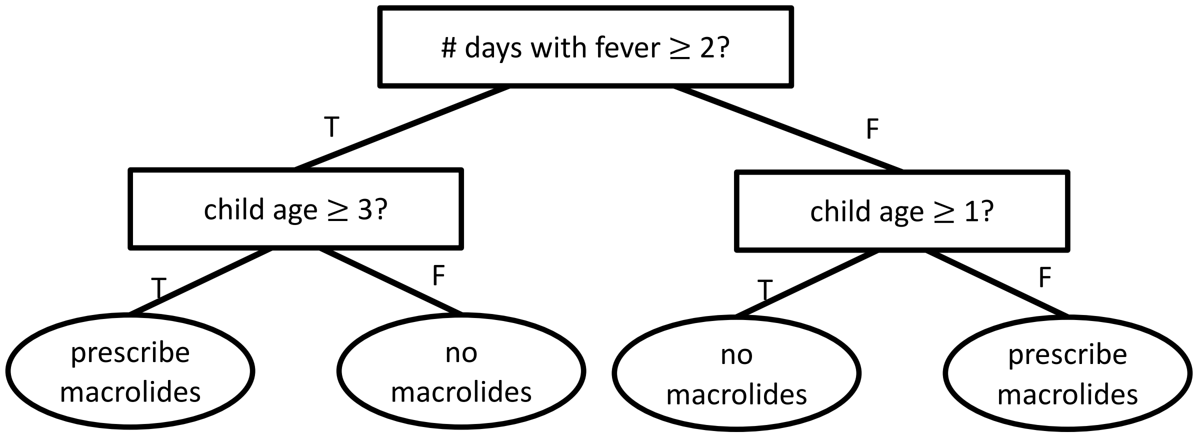
if $g_{N_0} > 0$:

return PostPruneTree$(T_N, Z_{\text{train}}, Z_{\text{val}})$

else:

return T





Post Pruning

def PostPruneTree($T, Z_{\text{train}}, Z_{\text{val}}$):

  for each internal node N of T:

    $T_N \leftarrow \text{Replace}\left(T, N, \text{LeafNode}\left(\text{Mode}(Z_{\text{train}}[N])\right)\right)$

    $g_N \leftarrow \text{Loss}(T, Z_{\text{val}}) - \text{Loss}(T_N, Z_{\text{val}})$

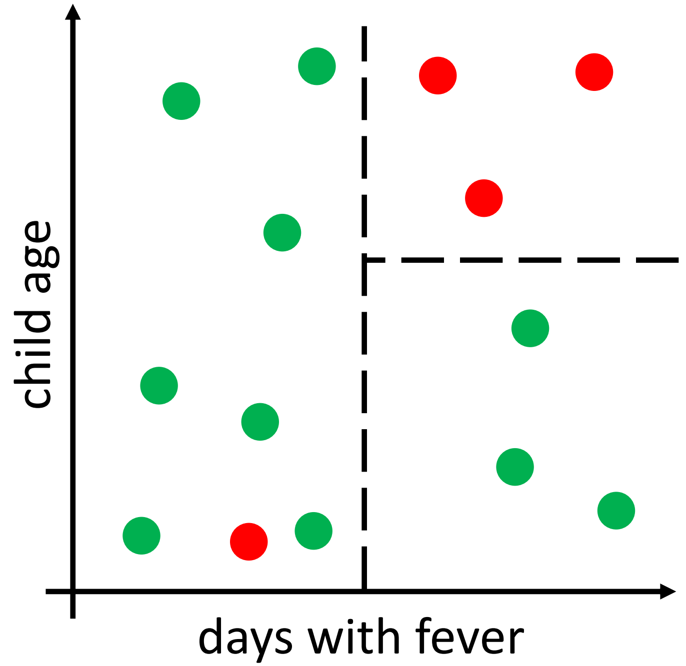
  $N_0 \leftarrow \arg\max_N g_N$

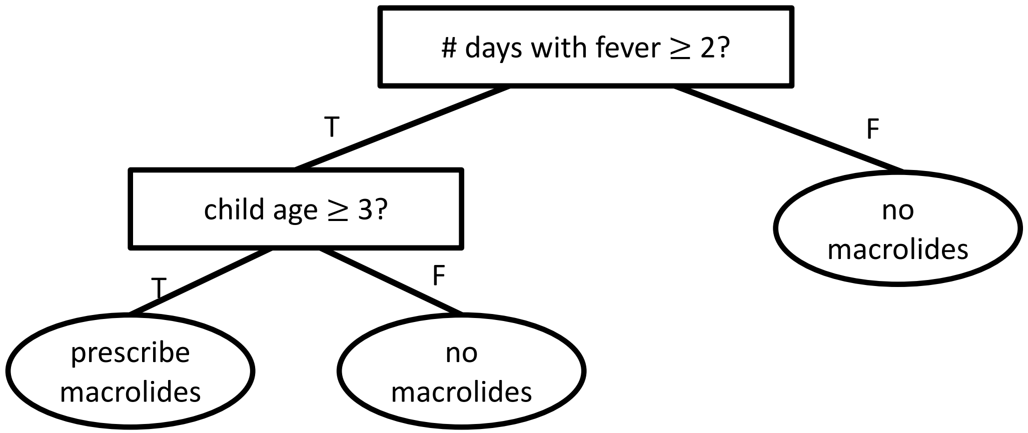
  if $g_{N_0} > 0$:

    return PostPruneTree($T_N, Z_{\text{train}}, Z_{\text{val}}$)

  else:

    return T





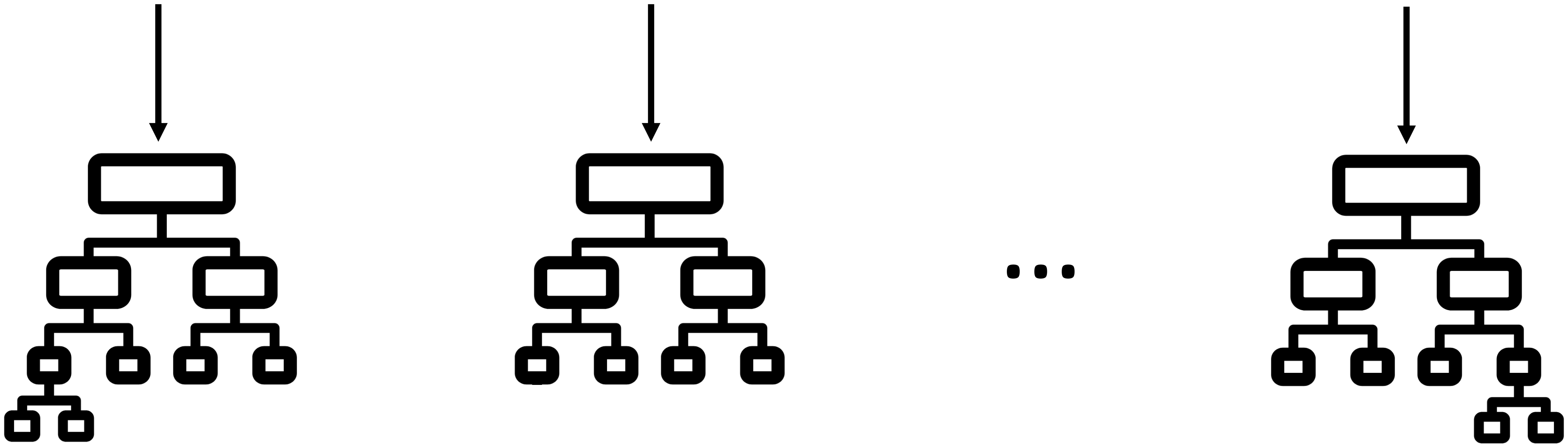
Decision Tree Shortcomings

- Hard to manage bias-variance tradeoff
  - Small depth → High bias, low variance
  - Large depth → Small bias, high variance
  - What if a different decision boundary would have worked?

- Can we manage this tradeoff in a more principled way?

- **Idea:** Random forests
  - Grow large decision trees
  - Rather than prune, average many of them!

Random Forests



Random Forests

- Train many decision trees and average them!
  - Large depth → High variance, low bias
  - Averaging many decision trees → average away “irrelevant” variance

- Very powerful model family in practice

Ensembles

- More generally, **ensembles** are an effective strategy for mitigating the bias-variance tradeoff

- **Approaches so far:**
  - Different model family
  - Feature engineering

- **Ensembles:**
  - Combine models to reduce bias without increasing variance

Ensemble Learning

- **Step 1:** Learn a set of "base" models $f_1, \ldots, f_k$
- **Step 2:** Construct model $F(x)$ that combines predictions of $f_1, \ldots, f_k$

Example: Netflix Movie Recommendations

- **Goal:** Predict how a user will rate a movie based on:
  - The user's ratings for other movies
  - Other users' ratings for this movie (and others)
  - **No features!**

- **Netflix Prize (2007-2009):** $1 million for the first team to do 10% better than the existing Netflix recommendation system

- **Winner:** BellKor's Pragmatic Chaos
  - An ensemble of 800+ rating systems

Ensembles of Decision Trees

- **Strategy 1:** Random forests
- **Strategy 2:** Gradient boosted decision trees
- Among the most powerful and widely-used models for “tabular” data (i.e., not images, text, graphs, or other highly structured data)

Ensemble Design Decisions

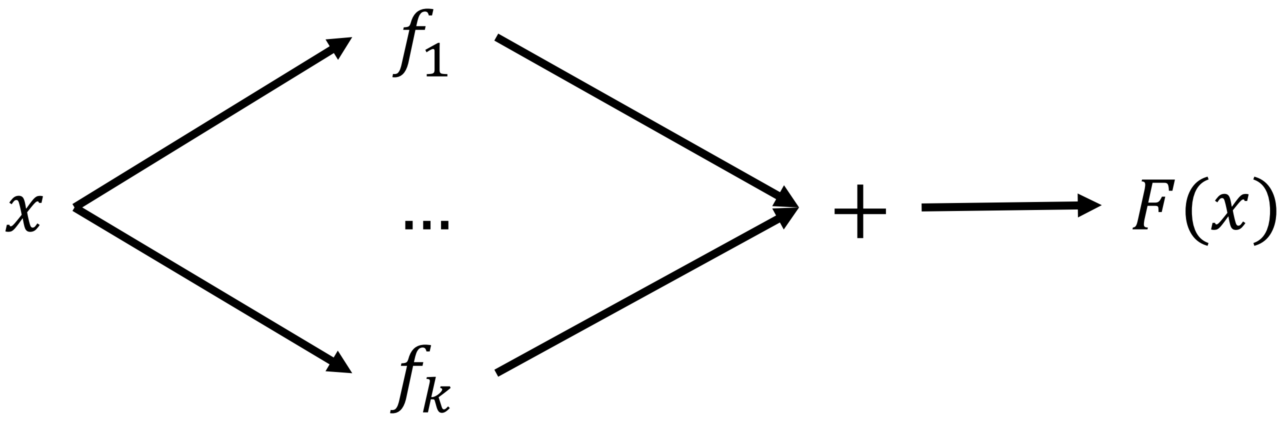
- How to learn the base models?
- How to combine the learned base models?

Ensemble Design Decisions

- How to learn the base models?
- **How to combine the learned base models?**

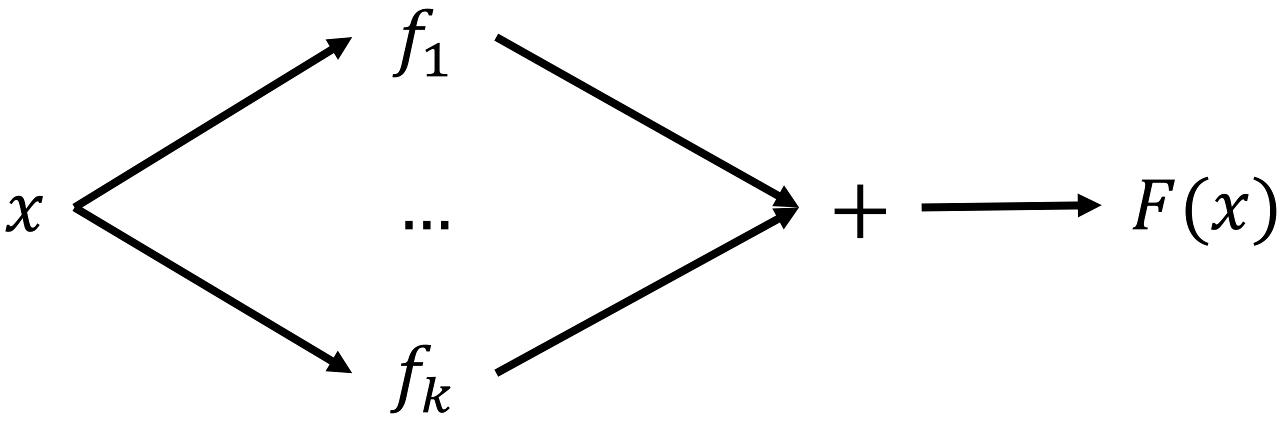
Combining Learned Base Models

- **Regression:** Average predictions $F(x) = \frac{1}{k}\sum_{i=1}^{k} f_i(x)$
  - Works well if the base models have similar performance



Combining Learned Base Models

- **Classification:** Majority vote $F(x) = 1\left(\sum_{i=1}^{k} f_i(x) \geq \frac{k}{2}\right)$ (for binary)
    - Can also average probabilities for classification



Combining Learned Base Models

- Can use weighted average:

$$F(x) = \sum_{i=1}^{k} \beta_i \cdot f_i(x)$$

- Can fit weights using linear regression on second training set

- More generally, can fit a second layer model

$$F(x) = g_\beta\big(f_1(x), \dots, f_k(x)\big)$$

Combining Learned Base Models

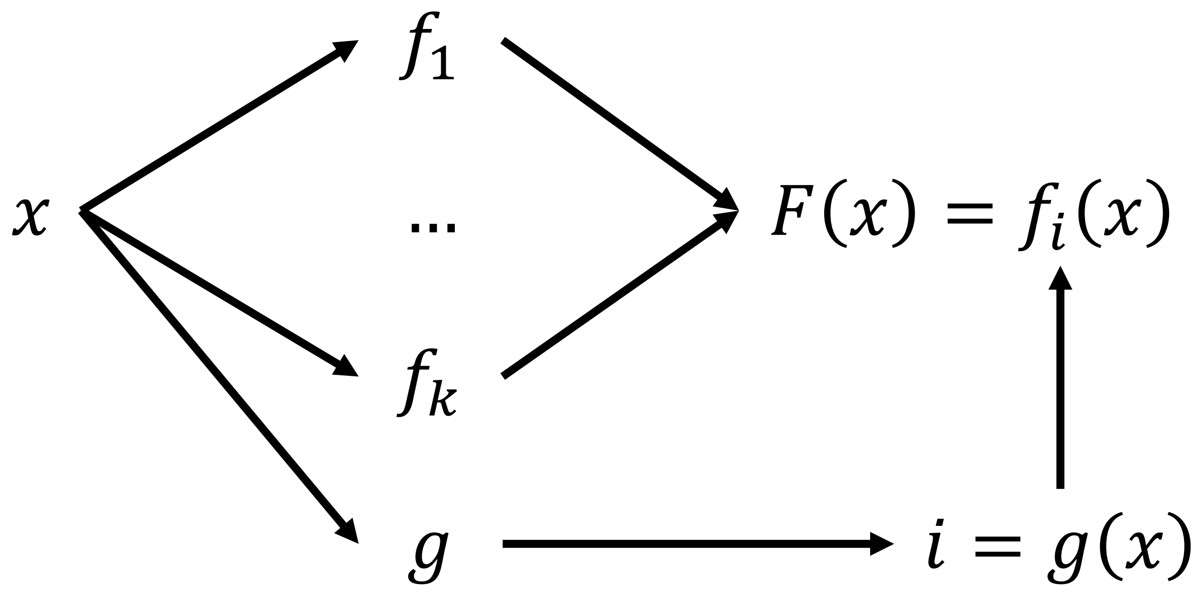
- Second model as "mixture of experts":

$$F(x) = \sum_{i=1}^{k} g(x)_i \cdot f_i(x)$$

- Second stage model predicts weights over "experts" $f_i(x)$

Combining Learned Base Models

- Second model as "mixture of experts":
  - **Special case:** $g(x)$ is one-hot
  - **Advantage:** Only need to run $g(x)$ and $f_{g(x)}(x)$



Ensemble Design Decisions

- How to learn the base models?
- How to combine the learned base models?

Ensemble Design Decisions

- **How to learn the base models?**

- How to combine the learned base models?

Learning Base Models

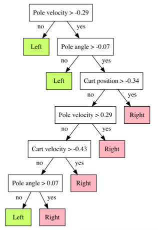
- Successful ensembles require **diversity**
  - Different model families
  - Different training data
  - Different features
  - Different hyperparameters

- **Intuition:** Models should make **independent** mistakes

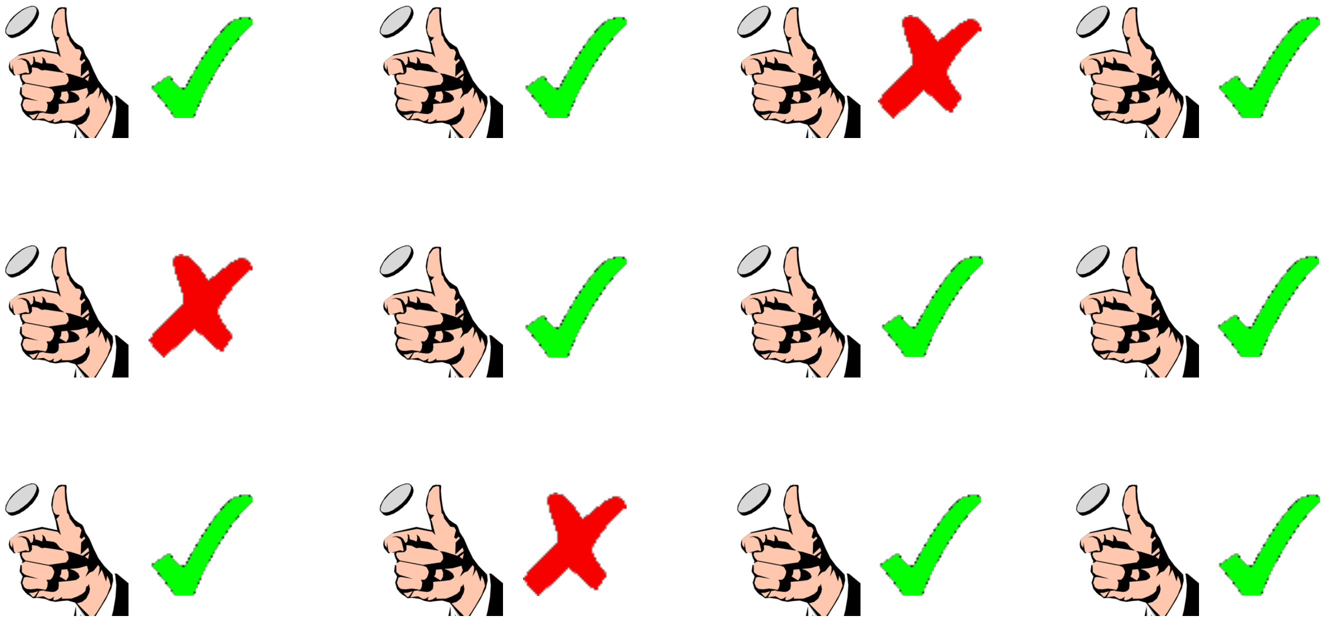
Learning Base Models

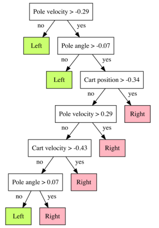
- **Intuition:** Models should make **independent** mistakes

x_1 x_2 x_3 x_4

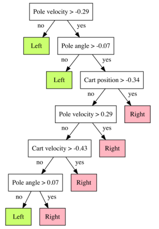


$\text{acc} = \frac{3}{4}$





$\text{acc} = \frac{3}{4}$



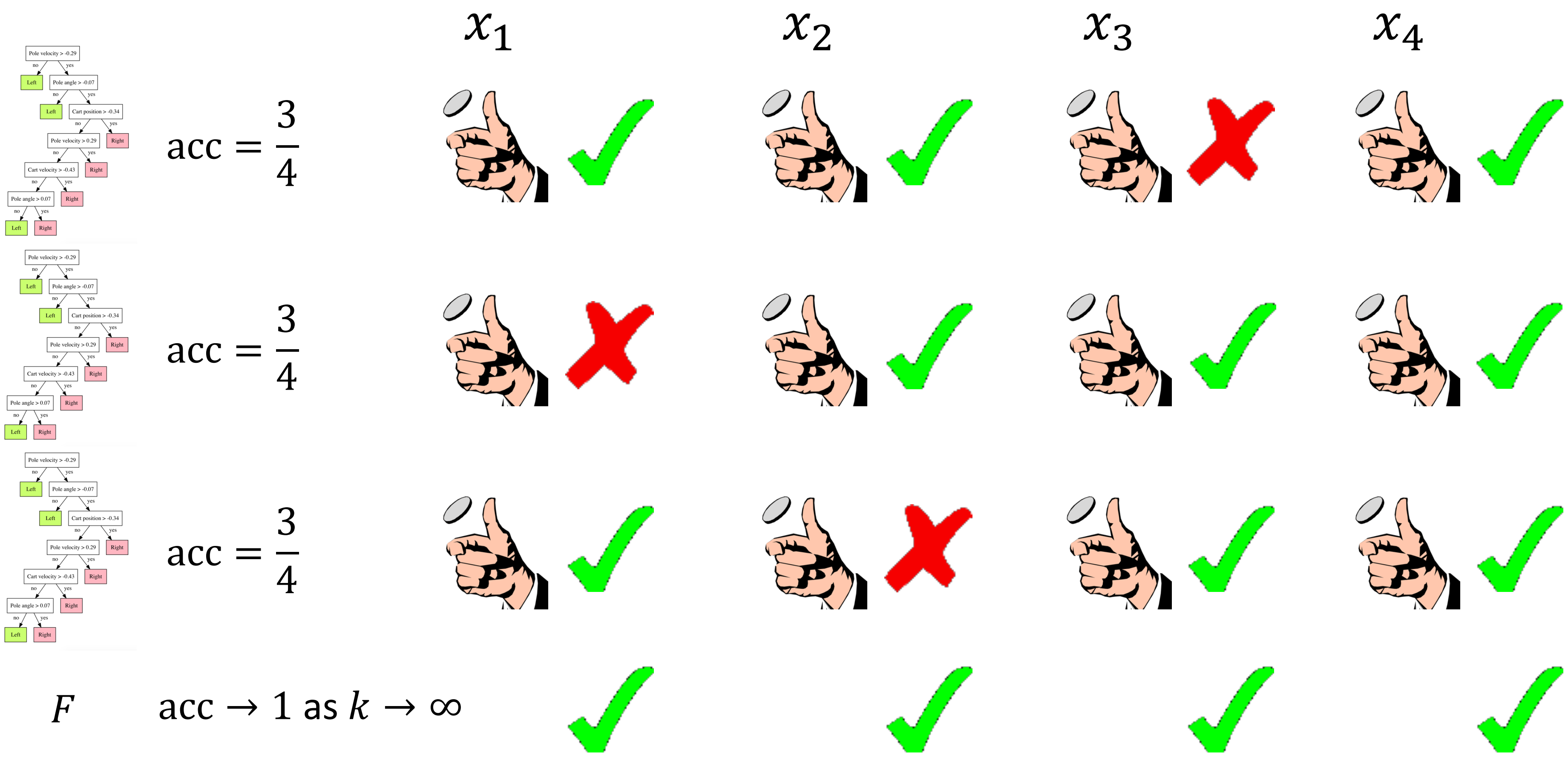
$\text{acc} = \frac{3}{4}$

F

$$\text{acc} = 1 - \left(1 - \frac{3}{4}\right)^3 - 3 \cdot \frac{3}{4} \cdot \left(1 - \frac{3}{4}\right)^2 \approx 0.84$$

Learning Base Models

- **Intuition:** Models should make **independent** mistakes



Learning Base Models

- Ensemble can be built from different learning algorithms
  - **Example:** Decision tree, logistic regression, kNN, …
- What if we want an ensemble of decision trees?
  - **Issue:** Decision tree learning algorithm is deterministic
  - **Solution:** Randomize the learning algorithm (may sacrifice performance)!
- Randomize decisions inside learning algorithm
  - **Example:** Randomize splits weighted (somehow) by information gain
  - **Issue:** Very specific to the algorithm
  - **Solution:** Randomize input to learning algorithm (i.e., training data)!

Randomizing Learning Algorithms

- **Bagging:** Randomize training data ("Boostrap Aggregating")
  - **Random examples:** Subsample examples $\{(x, y)\}$ (obtain $X \in \mathbb{R}^{n' \times d}$)
    Random features: Subsample features x_j (obtain $X \in \mathbb{R}^{n \times d'}$)

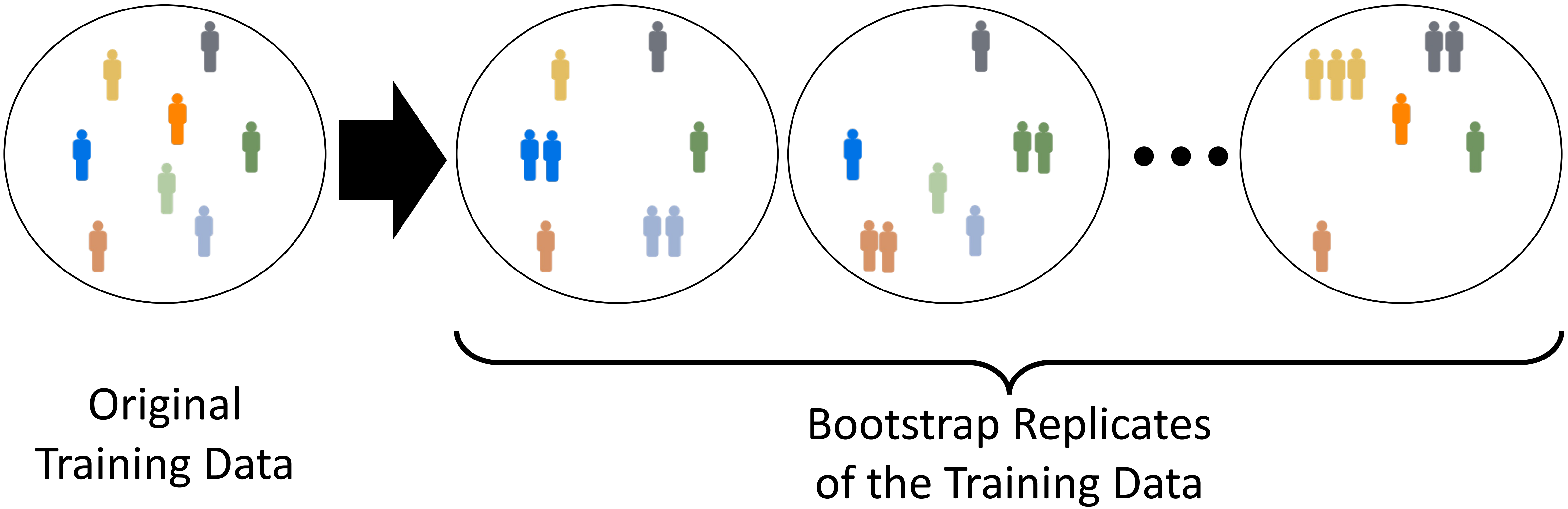
- Meta-strategy that can build ensembles from arbitrary base learners

- Can be thought of as a form of regularization

Bootstrap

- Subsample examples $\{(x, y)\}$ **with replacement** (obtain $X \in \mathbb{R}^{n \times d}$)
- Excludes $\left(1 - \frac{1}{n}\right)^n$ of the training examples
  - Separately in each of the replicates
  - As $n \to \infty$, excludes $\to \frac{1}{e} \approx 36.8\%$ examples
- Has good statistical properties

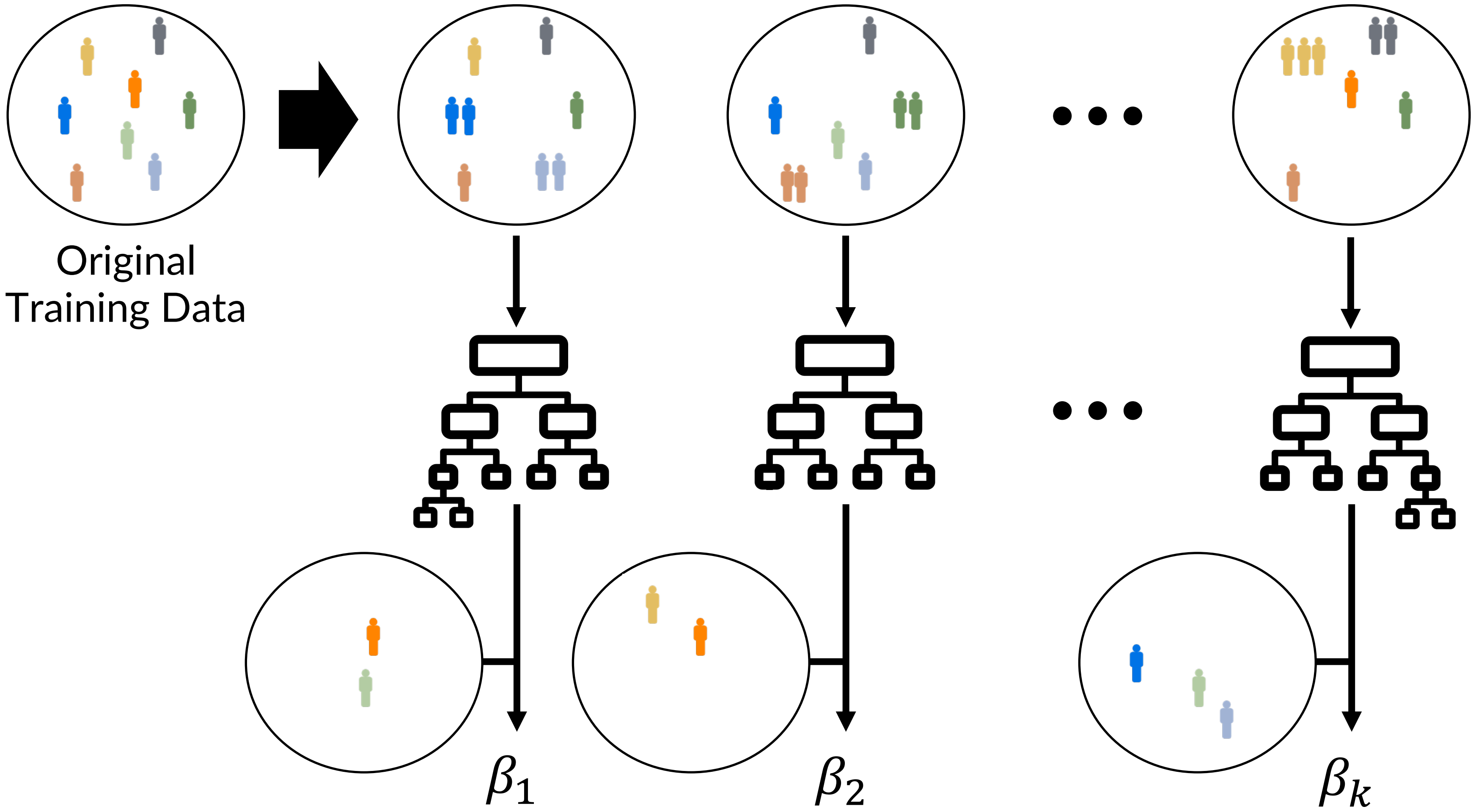
Randomizing Learning Algorithms



Ensemble Learning

- **Step 1:** Create bootstrap replicates of the original training dataset
- **Step 2:** Train a classifier for each replicate
- **Step 3 (Optional):** Use held-out validation set to weight models
  - Can just use average predictions

Ensemble Learning



Random Forests

- Ensemble of decision trees using bagging
  - Typically use simple average (over probabilities for classification)

- **Intuition:**
  - Large decision trees are good nonlinear models, but high variance
  - Random forests average over many decision trees to reduce variance without increasing bias

Random Forests

- **Tweak 1:** Randomize features in learning algorithm instead of bagging
  - At DT node splitting step, subsample $\approx \sqrt{d}$ features
  - Allows each tree to use all features, but not at every node
  - **Aside:** If a few features are highly predictive, then they will be selected in many trees, causing the base models to be highly correlated

- **Tweak 2:** Train **unpruned** decision trees
  - Ensures base models have higher capacity
  - **Intuition:** Skipping pruning increases variance

Bias Variance Tradeoff for Random Forests

- Naïvely, skipping pruning yields high variance
- Introduce randomness to average away “excess” variance
  - Without randomness, all models in the random forest would be the same (large) decision tree, so the random forest would still have very large variance
- Randomness should ideally make base models more independent

AdaBoost (Freund & Schapire 1997)

- Like bagging, meta-algorithm that turns base models into ensemble
  - **Provably learns** for base models achieving any error rate > 0.5
- Uses **different training example weights** (instead of different subsamples or different features) to introduce diversity
  - In particular, **upweights** currently incorrectly predicted examples
- Base models should satisfy the following:
  - High-bias/low-capacity (e.g., depth-limited decision trees, linear classifiers)
  - Able to incorporate sample weights during learning
  - **Specific to classification (discuss general losses later)**

AdaBoost (Freund & Schapire 1997)

- **Input**
  - Training dataset Z
  - Learning algorithm $\text{Train}(Z, w)$ that can handle weights w
  - Hyperparameter T indicating number of models to train

- **Output**
  - Ensemble of models $F(x) = \sum_{t=1}^{T} \beta_t \cdot f_t(x)$

Aside: Learning with Weighted Examples

- Many algorithms can directly incorporate weights into the loss

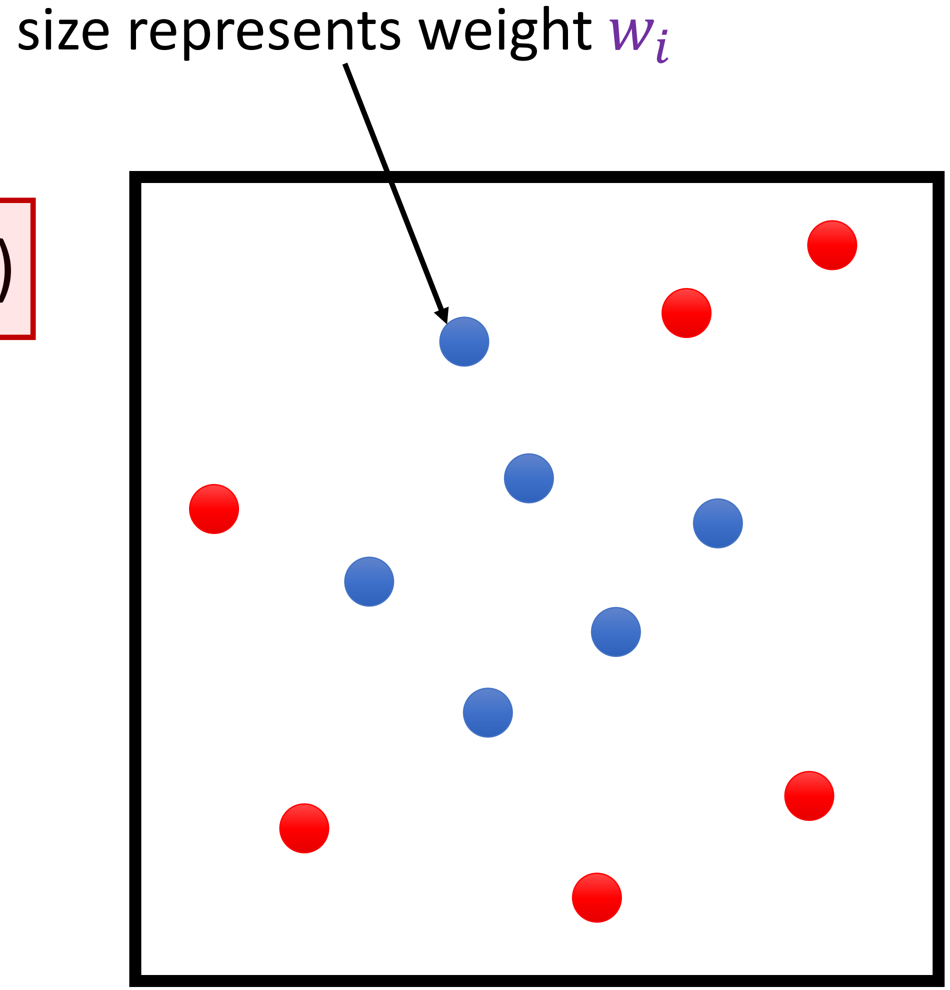
- For maximum likelihood estimation:

$$\ell(\beta; Z, w) = \sum_{i=1}^{n} w_i \cdot \log p_\beta(y_i \mid x_i)$$

- Alternatively, can subsample the data proportional to weights w_i

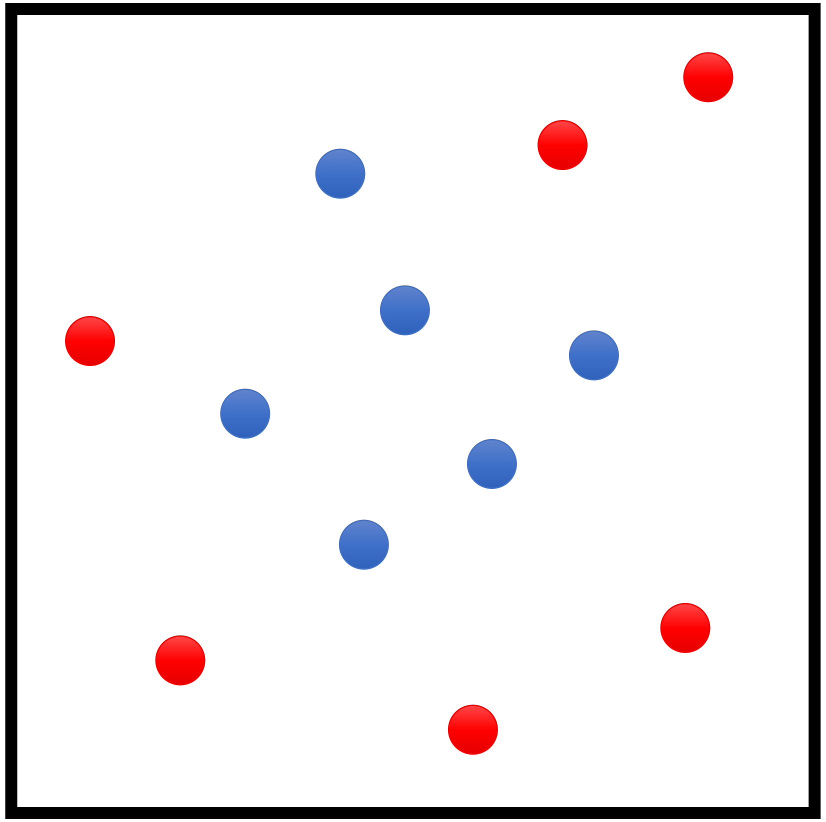
AdaBoost

1. $w_1 \leftarrow \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \ldots, T\}$
3. $\quad f_t \leftarrow \text{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all i)
7. **return** $F(x) = \text{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$



AdaBoost

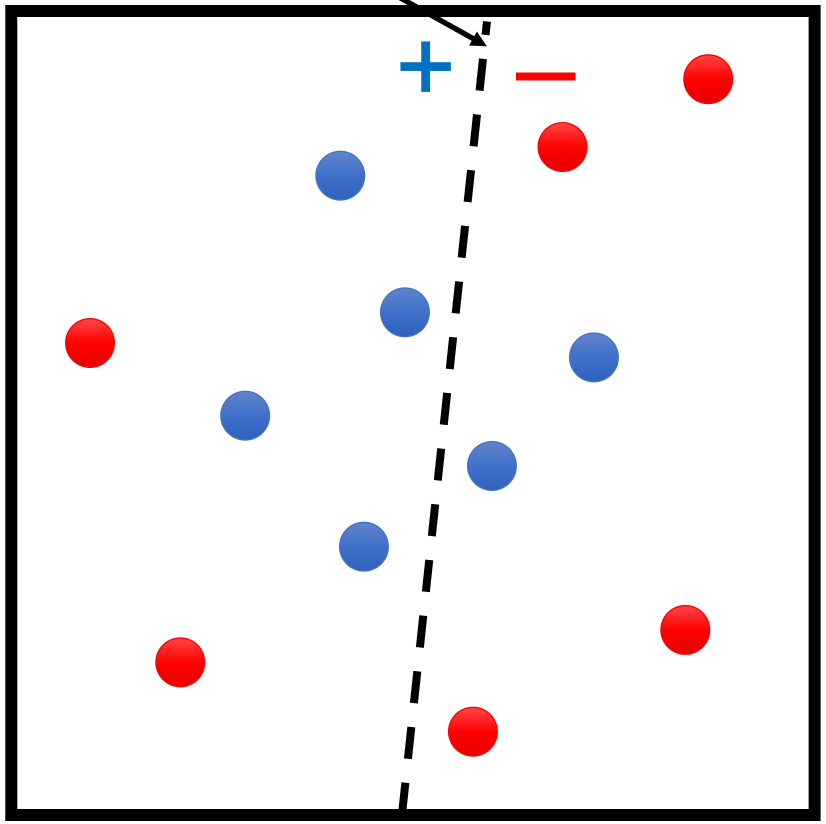
1. $w_1 \leftarrow \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \dots, T\}$
3. $\quad f_t \leftarrow \text{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all i)
7. **return** $F(x) = \text{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$



AdaBoost

1. $w_1 \leftarrow \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \ldots, T\}$
3. $\quad f_t \leftarrow \text{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all i)
7. **return** $F(x) = \text{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$

focus on linear classifiers f_t

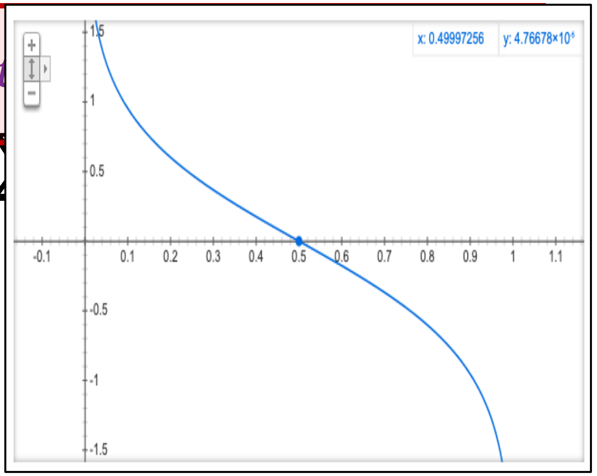


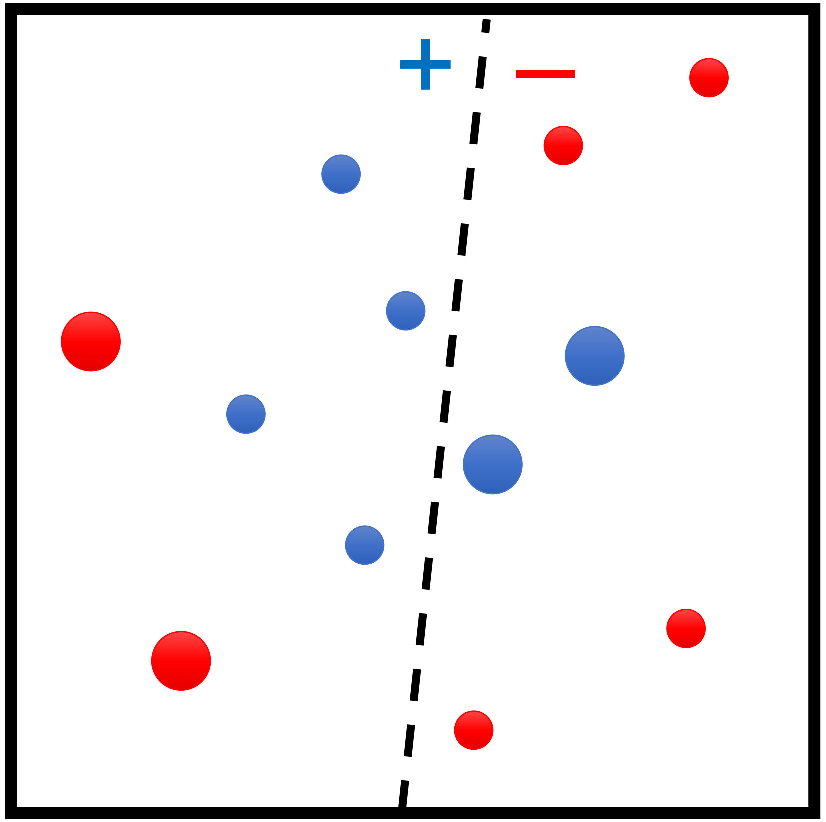
$t = 1$

AdaBoost

1. $w_1 \leftarrow \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \ldots, T\}$
3. $\quad f_t \leftarrow \text{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t}$
7. **return** $F(x) = \text{sign}($

β_t becomes larger as
ϵ_t becomes smaller





$t = 1$

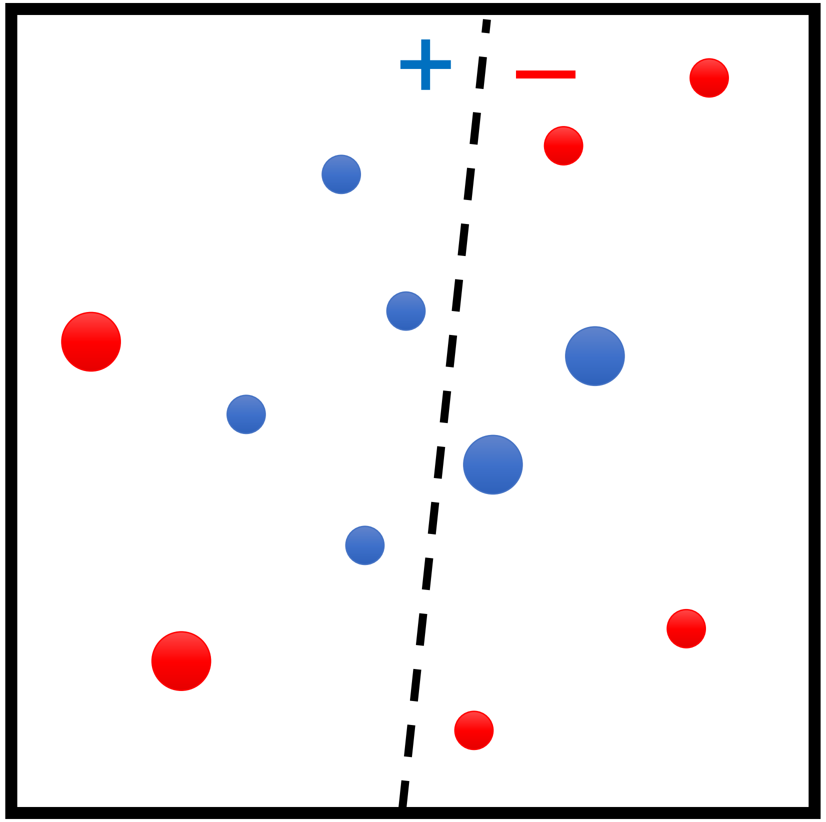
AdaBoost

1. $w_1 \leftarrow \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \ldots, T\}$
3. $\quad f_t \leftarrow \text{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all i)
7. **return** $F(x) = \text{sign}\left(\sum_{t=1}^{T} \beta_t \cdot f_t(x)\right)$

Use convention $y_i \in \{-1, +1\}$

If correct ($y_i = f_t(x_i)$) then multiply by $e^{-\beta_t}$

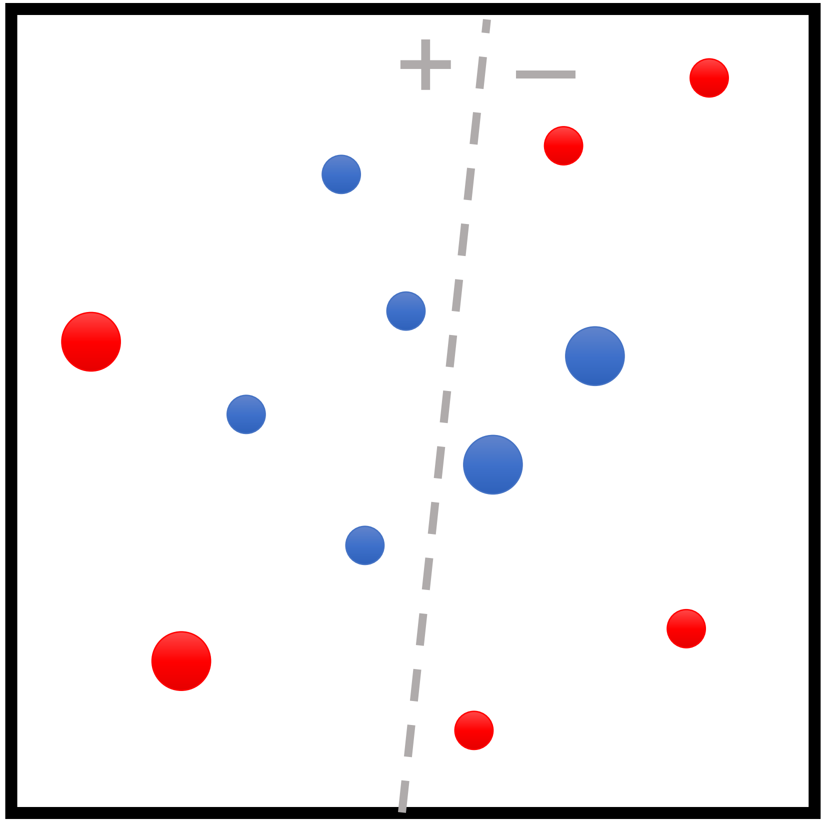
If incorrect ($y_i \neq f_t(x_i)$) then multiply by e^{β_t}



$t = 1$

AdaBoost

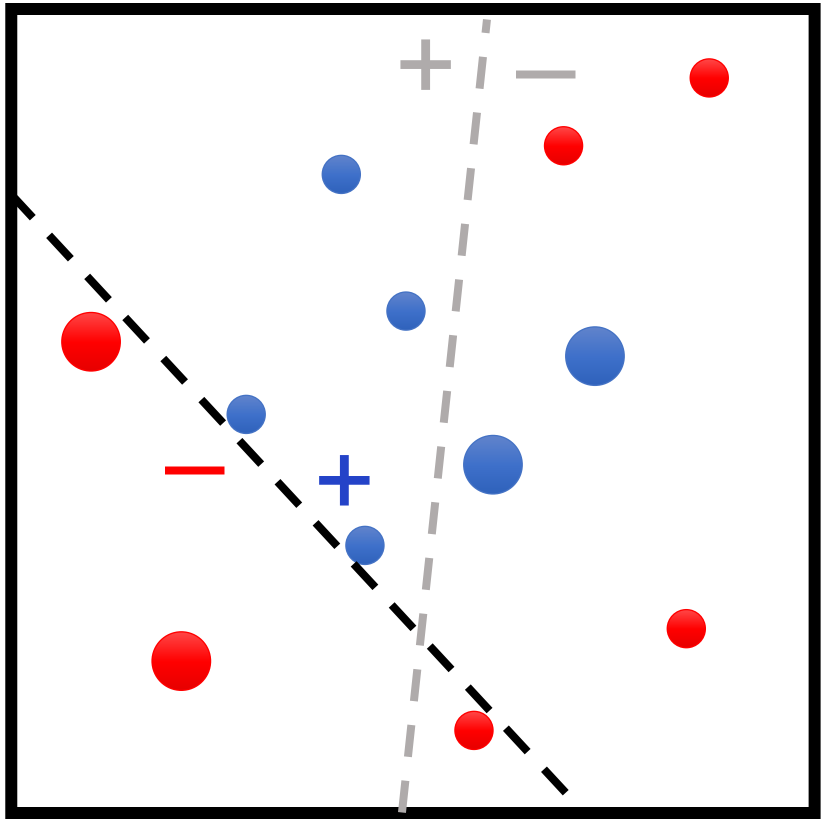
1. $w_1 \leftarrow \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \dots, T\}$
3. $\quad f_t \leftarrow \text{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all i)
7. **return** $F(x) = \text{sign}\left(\sum_{t=1}^{T} \beta_t \cdot f_t(x)\right)$



$t = 1$

AdaBoost

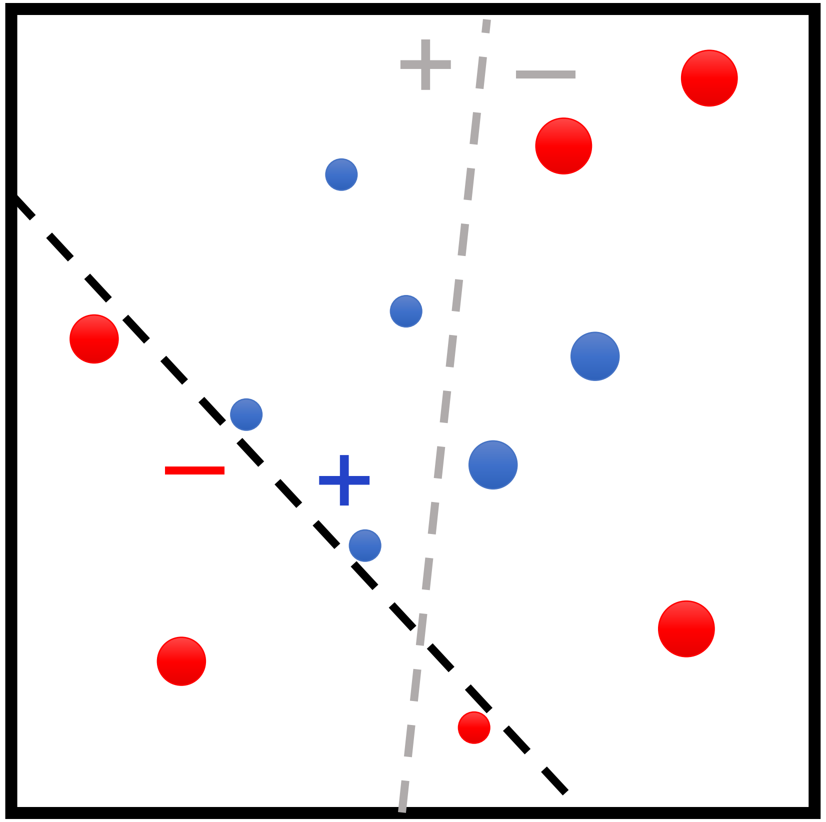
1. $w_1 \leftarrow \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \ldots, T\}$
3. $\quad f_t \leftarrow \mathrm{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \mathrm{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all i)
7. **return** $F(x) = \mathrm{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$



$t = 2$

AdaBoost

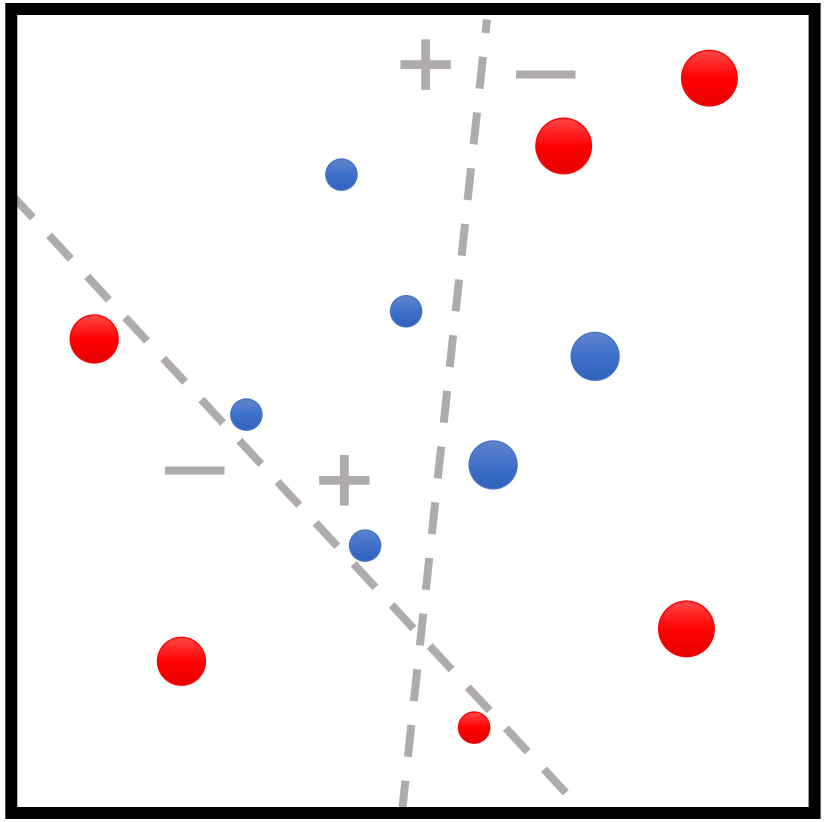
1. $w_1 \leftarrow \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \dots, T\}$
3. $\quad f_t \leftarrow \text{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all i)
7. **return** $F(x) = \text{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$



$t = 2$

AdaBoost

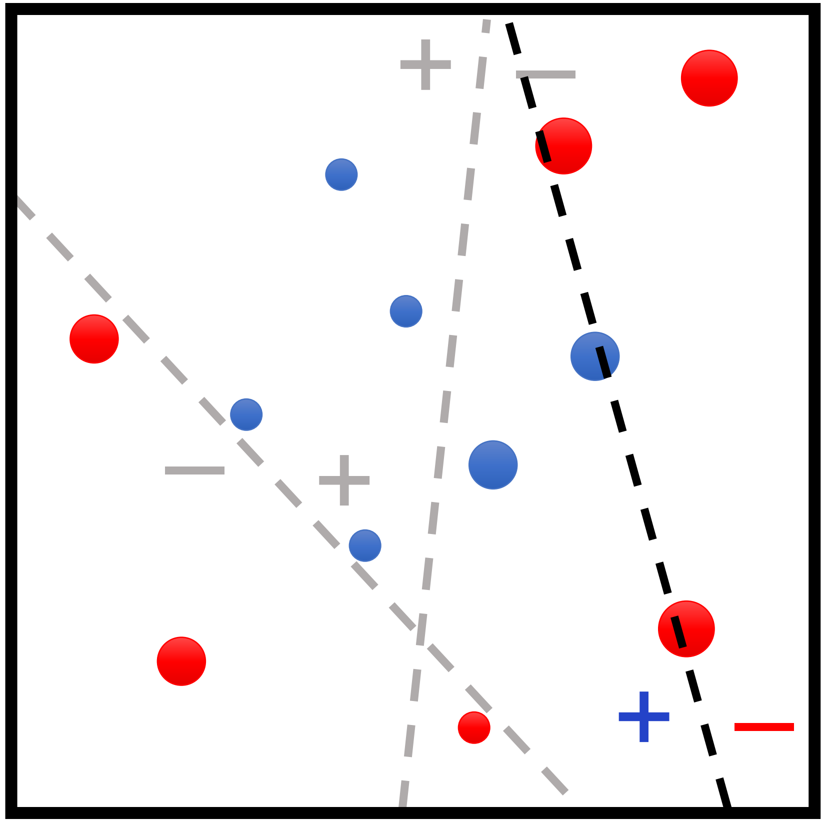
1. $w_1 \leftarrow \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \dots, T\}$
3. $\quad f_t \leftarrow \mathrm{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \mathrm{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all i)
7. **return** $F(x) = \mathrm{sign}\left(\sum_{t=1}^{T} \beta_t \cdot f_t(x)\right)$



$t = 2$

AdaBoost

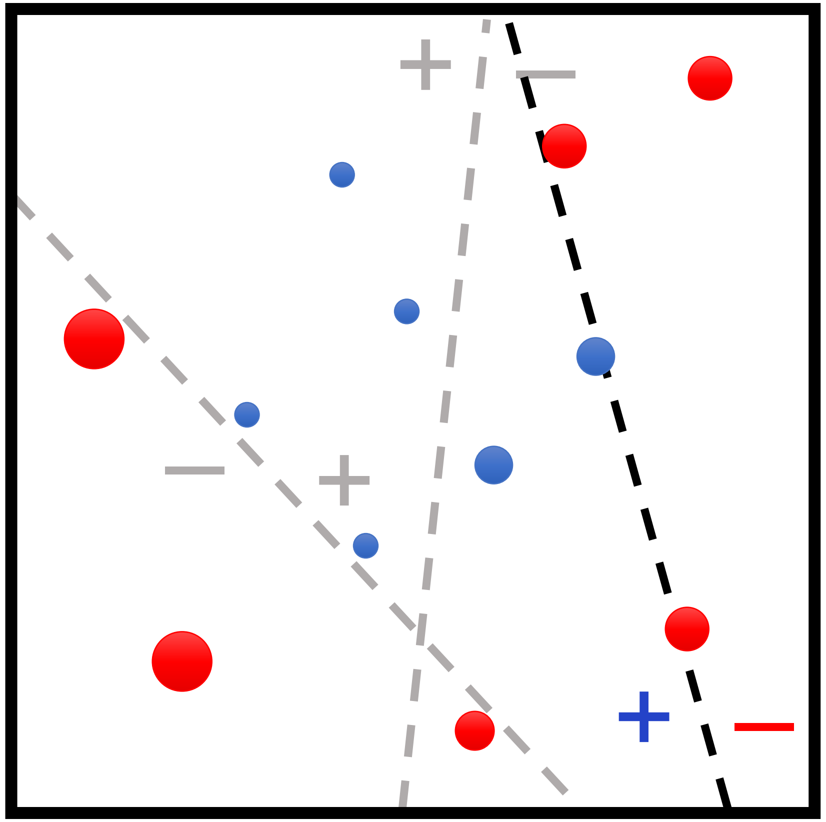
1. $w_1 \leftarrow \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \dots, T\}$
3. $\quad f_t \leftarrow \mathrm{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \mathrm{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all i)
7. **return** $F(x) = \mathrm{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$



$t = 3$

AdaBoost

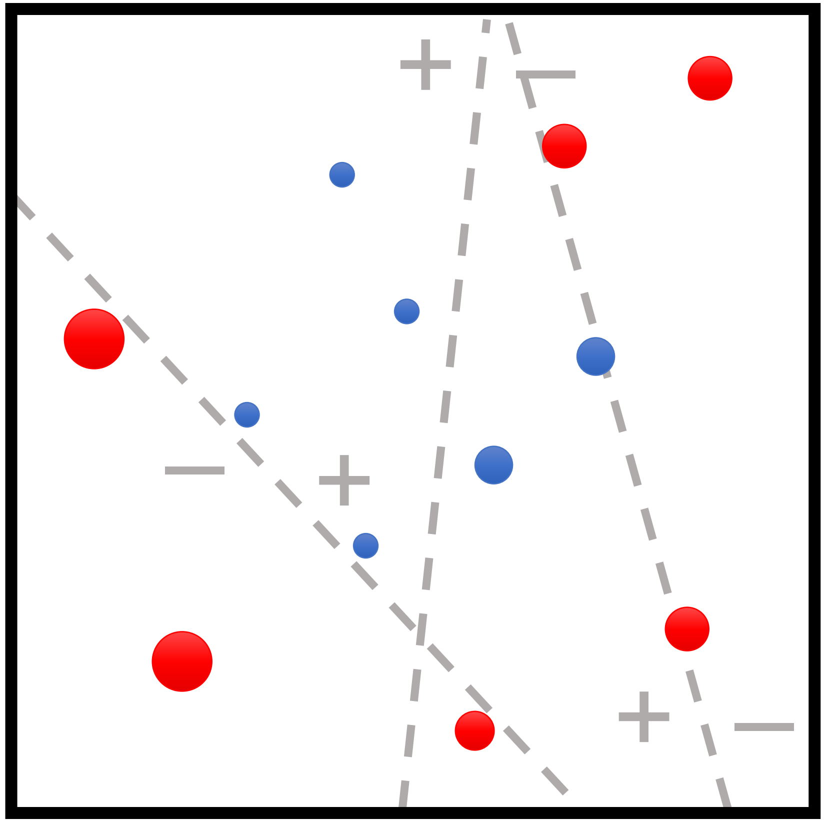
1. $w_1 \leftarrow \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \dots, T\}$
3. $\quad f_t \leftarrow \text{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all i)
7. **return** $F(x) = \text{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$



$t = 3$

AdaBoost

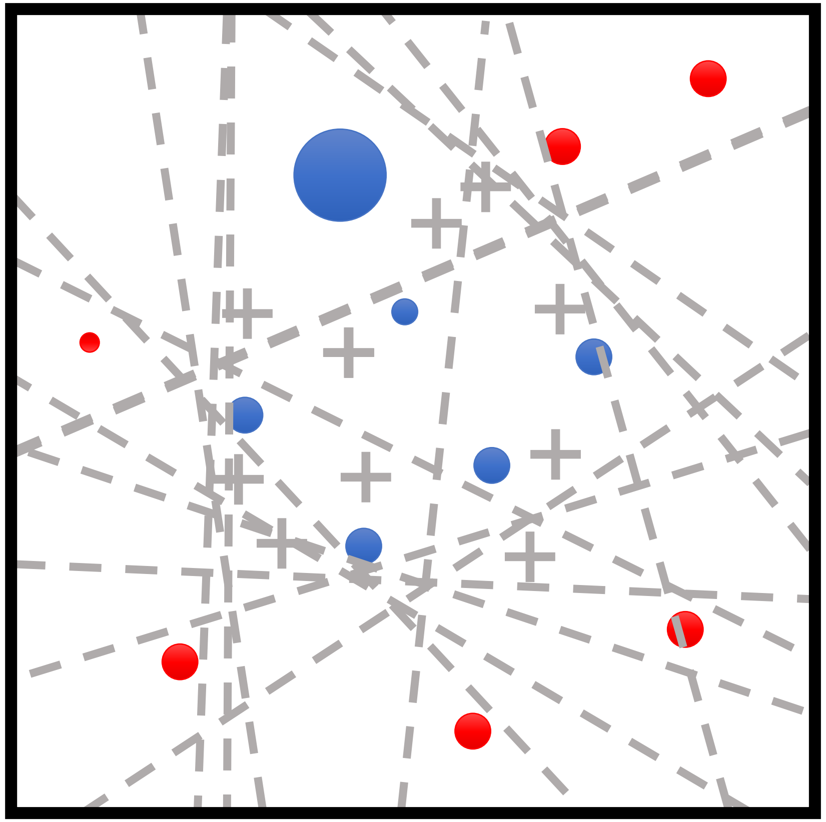
1. $w_1 \leftarrow \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \dots, T\}$
3. $\quad f_t \leftarrow \mathrm{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \mathrm{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all i)
7. **return** $F(x) = \mathrm{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$



$t = 3$

AdaBoost

1. $w_1 \leftarrow \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \dots, T\}$
3. $\quad f_t \leftarrow \mathrm{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \mathrm{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all i)
7. **return** $F(x) = \mathrm{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$



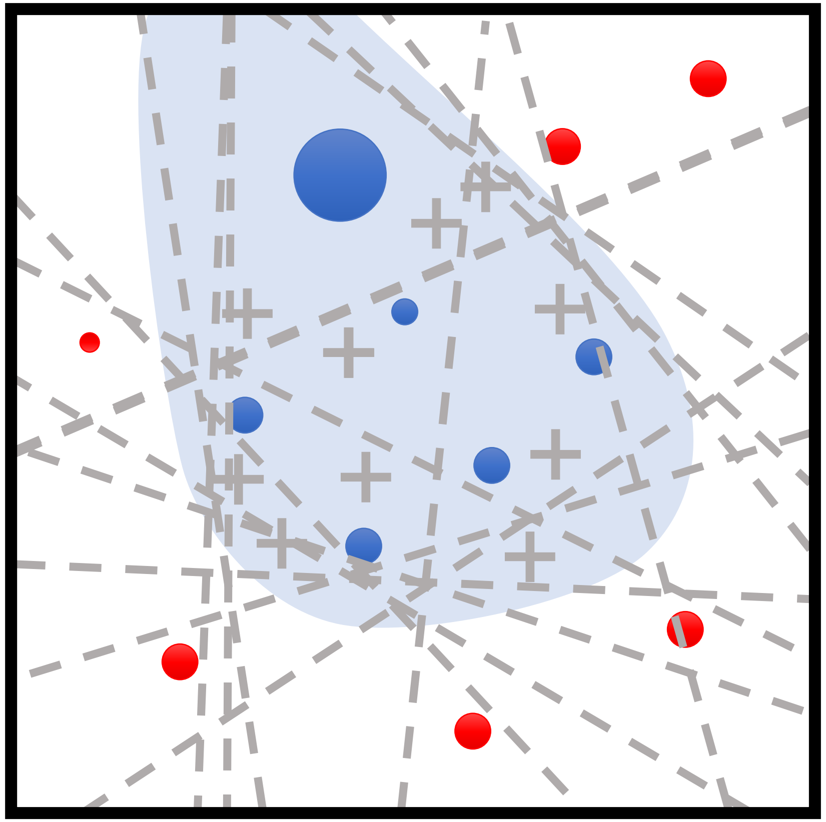
$t = T$

Under certain assumptions, training error goes to zero in $O(\log n)$ iterations

AdaBoost

1. $w_1 \leftarrow \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \ldots, T\}$
3. $\quad f_t \leftarrow \mathrm{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \mathrm{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all i)
7. **return** $F(x) = \mathrm{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$

final model is average of base models weighted by their performance



AdaBoost Weighting Strategy

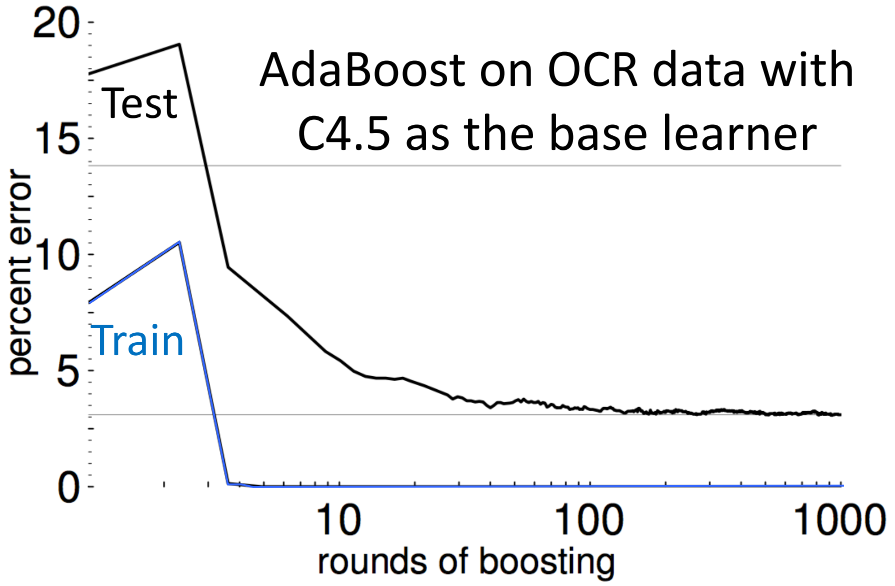
- On each iteration:
  - Misclassified examples are upweighted
  - Correctly classified are downweighted

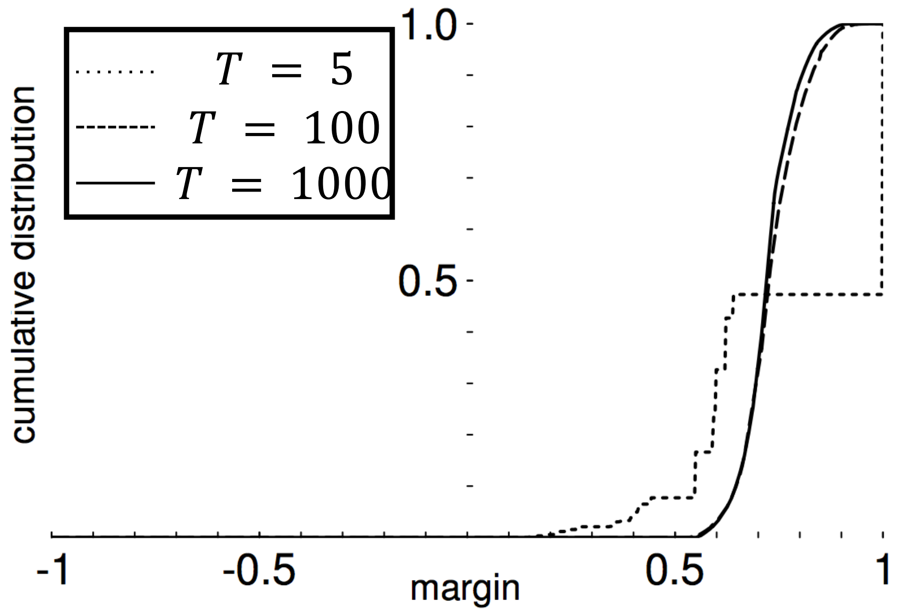
- If an example is repeatedly misclassified, it will eventually be upweighted so much that it is correctly classified

- Emphasizes “hardest” parts of the input space
  - Instances with highest weight are often outliers

AdaBoost and Overfitting

- Basic ML theory predicts AdaBoost always overfits as $T \rightarrow \infty$
  - Hypothesis keeps growing more complex!
  - In practice, AdaBoost often does not overfit





AdaBoost Summary

- **Strengths:**
  - Fast and simple to implement
  - No hyperparameters (except for T)
  - Very few assumptions on base models
- **Weaknesses:**
  - Can be susceptible to noise/outliers when there is insufficient data
  - No way to parallelize
  - Small gains over complex base models
  - **Specific to classification!**

Boosting as Gradient Descent

- Set of heuristics inspired by AdaBoost

Boosting as Gradient Descent

- Both algorithms: new model = old model + update
- **Gradient Descent:**

$$\theta_{t+1} = \theta_t - \alpha \cdot \nabla_\theta L(\theta_t; Z)$$

- **Boosting:**

$$F_{t+1}(x) = F_t(x) + \beta_{t+1} \cdot f_{t+1}(x)$$

- Here, $F_t(x) = \sum_{i=1}^{t} \beta_i \cdot f_i(x)$

Boosting as Gradient Descent

- Assuming $\beta_t = 1$ for all t, then:

$$F_t(x_i) + f_{t+1}(x_i) = F_{t+1}(x_i)$$

Boosting as Gradient Descent

- Assuming $\beta_t = 1$ for all t, then:

$$F_t(x_i) + f_{t+1}(x_i) = F_{t+1}(x_i) \approx y_i$$

- Rewriting this equation, we have

$$f_{t+1}(x_i) = F_{t+1}(x_i) - F_t(x_i) \approx \underbrace{y_i - F_t(x_i)}_{\text{"residuals", i.e., error of the current model}}$$

Boosting as Gradient Descent

- In other words, at each step, boosting is training the next model f_{t+1} to approximate the residual:

$$f_{t+1}(x_i) \approx \underbrace{y_i - F_t(x_i)}_{\text{“residuals”, i.e., error of the current model}}$$

- **Idea:** Train f_{t+1} directly to predict residuals $y_i - F_t(x_i)$

- **This strategy works for regression as well!**

Boosting as Gradient Descent

- **Algorithm:** For each $t \in \{1, \ldots, T\}$:
  - **Step 1:** Train f_{t+1} using dataset

$$Z_{t+1} = \{(x_i, y_i - F_t(x_i))\}_{i=1}^{n}$$

  - **Step 2:** Take

$$F_{t+1}(x) = F_t(x) + f_{t+1}(x)$$

- Return the final model F_T

Boosting as Gradient Descent

- Consider losses of the form

$$L(F;Z) = \frac{1}{n}\sum_{i=1}^{n} \tilde{L}(F(x_i); y_i)$$

- In other words, sum of individual label-level losses $\tilde{L}(\hat{y}; y)$ of a prediction $\hat{y} = F(x)$ if the ground truth label is y
- For example, $\tilde{L}(\hat{y}; y) = \frac{1}{2}(\hat{y} - y)^2$ yields the MSE loss

Boosting as Gradient Descent

- Residuals are the gradient of the squared error $\tilde{L}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$:

$$-\frac{\partial \tilde{L}}{\partial \hat{y}}(F_t(x_i); y_i) = y_i - F_t(x_i) = \mathrm{residual_i}$$

- For general $\tilde{L}$, instead of $\{(x_i, y_i - F_t(x_i))\}_{i=1}^{n}$ we can train f_{t+1} on

$$Z_{t+1} = \left\{\left(x_i, -\frac{\partial \tilde{L}}{\partial \hat{y}}(F_t(x_i); y_i)\right)\right\}_{i=1}^{n}$$

Boosting as Gradient Descent

- **Algorithm:** For each $t \in \{1, \ldots, T\}$:
  - **Step 1:** Train f_{t+1} using dataset

$$Z_{t+1} = \{(x_i, y_i - F_t(x_i))\}_{i=1}^{n}$$

  - **Step 2:** Take

$$F_{t+1}(x) = F_t(x) + f_{t+1}(x)$$

- Return the final model F_T

Boosting as Gradient Descent

- **Algorithm:** For each $t \in \{1, \ldots, T\}$:
  - **Step 1:** Train f_{t+1} using dataset

$$Z_{t+1} = \left\{ \left(x_i, -\frac{\partial \tilde{L}}{\partial \hat{y}} (F_t(x_i); y_i) \right) \right\}_{i=1}^{n}$$

  - **Step 2:** Take

$$F_{t+1}(x) = F_t(x) + f_{t+1}(x)$$

- Return the final model F_T

Boosting as Gradient Descent

- Casts ensemble learning in the **loss minimization framework**
  - **Model family:** Sum of base models $F_T(x) = \sum_{t=1}^{T} f_t(x)$
  - **Loss:** Any differentiable loss expressed as

$$L(F; Z) = \sum_{i=1}^{n} \tilde{L}(F(x_i), y_i)$$

- Gradient boosting is a general paradigm for training ensembles with specialized losses (e.g., most NLL losses)

Gradient Boosting in Practice

- Gradient boosting with depth-limited decision trees (e.g., depth 3) is one of the most powerful off-the-shelf classifiers available
  - **Caveat:** Inherits decision tree hyperparameters

- XGBoost is a very efficient implementation suitable for production use
  - A popular library for gradient boosted decision trees
  - Optimized for computational efficiency of training and testing
  - Used in many competition winning entries, across many domains
  - https://xgboost.readthedocs.io