Applied Machine Learning

Ensemble Learning

Computer Science, Fall 2022
Instructor: Xuhong Zhang

Ensemble Learning

- Consider a set of classifiers $h_1, \ldots, h_L$

- **Idea**: construct a classifier $H(x)$ that combines the individual decisions of $h_1, \ldots, h_L$
  - E.g., could have the member classifiers vote, or
  - could use different members for different regions of the instance space
  - Works well if the members each have low error rates

- Successful ensembles requires **diversity**
  - Classifiers should make different mistakes
  - Can have different types of base learners

Slide credit: Léon Bottou

Practical Application: Netflix Price

- Goal: predict how a user will rate a movie
  - Based on the user's ratings for other movies
  - And other peoples' ratings
  - With no other information about the movies

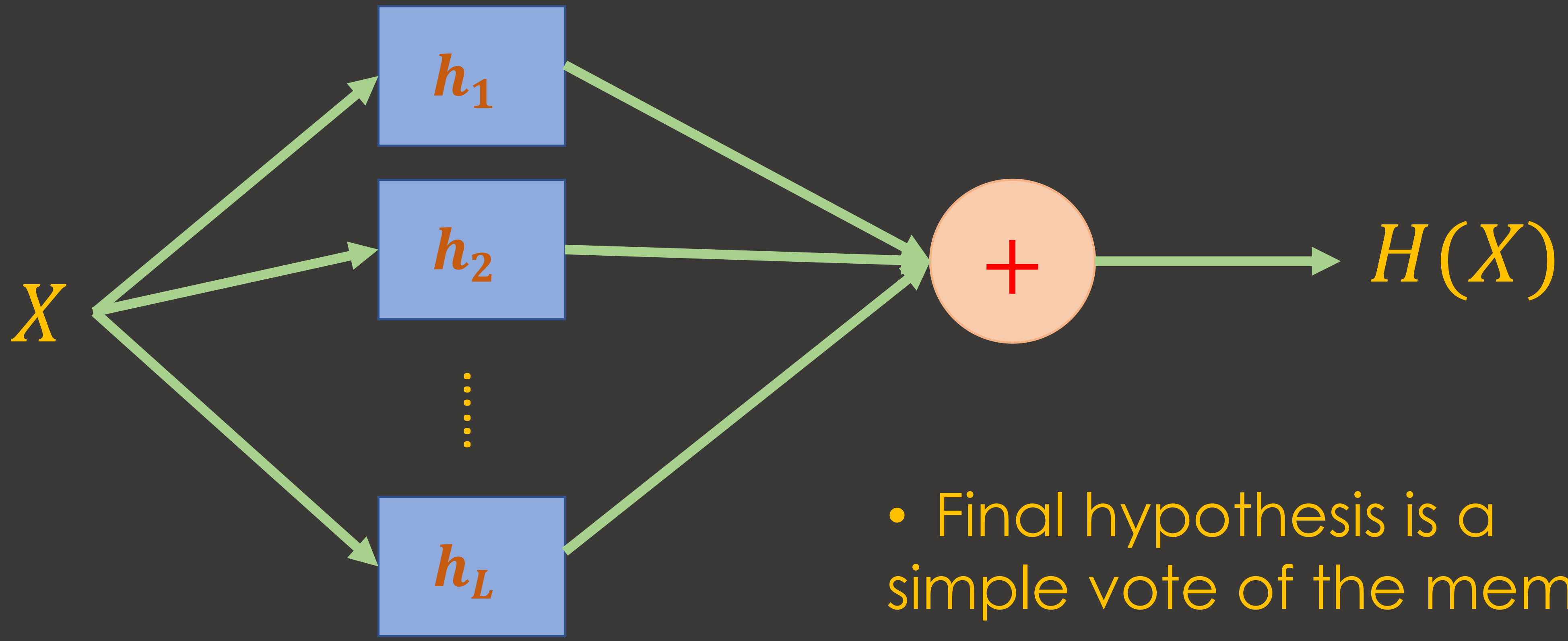
- This application is called "collaborative filtering"

- **Netflix Prize**: $1M to the first team to do 10% better than Netflix' system (2007-2009)

- **Winner**: Bellkor's Pragmatic Chaos—an ensemble of more than 800 rating systems.

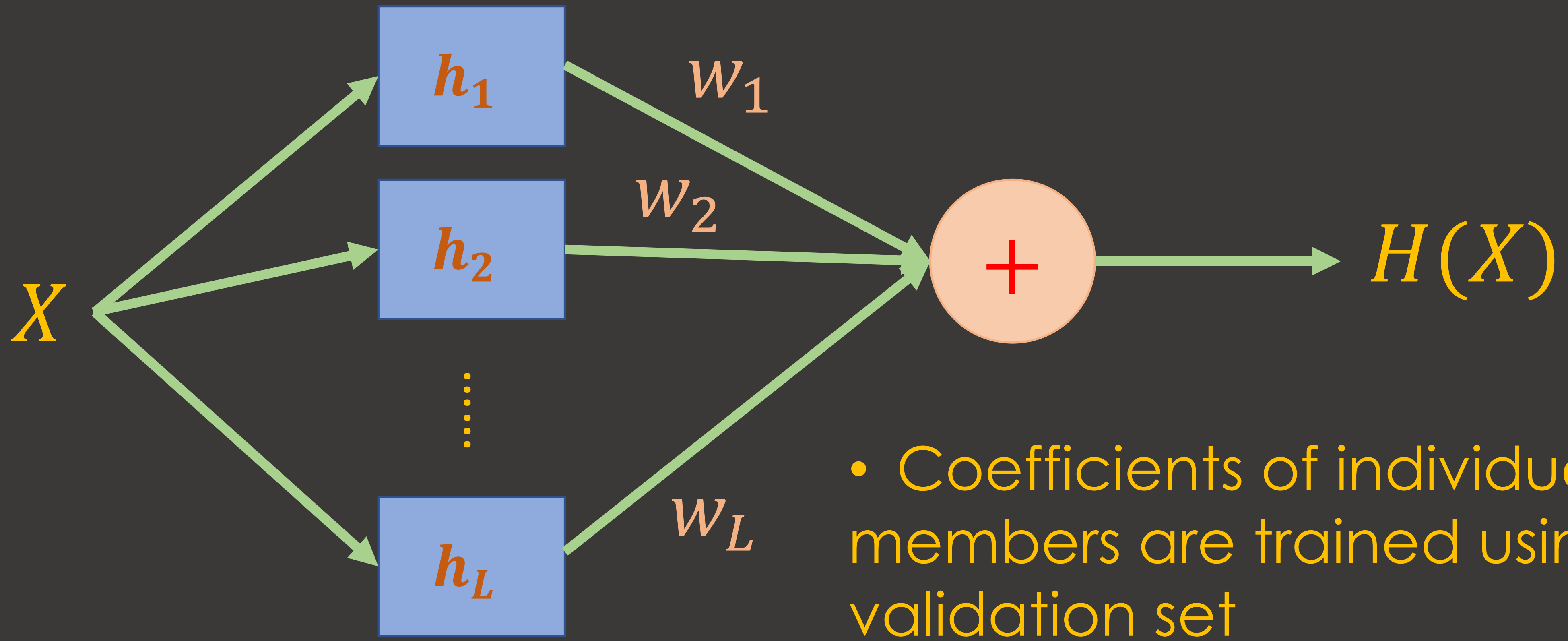
Slide credit: Léon Bottou

Combining Classifiers



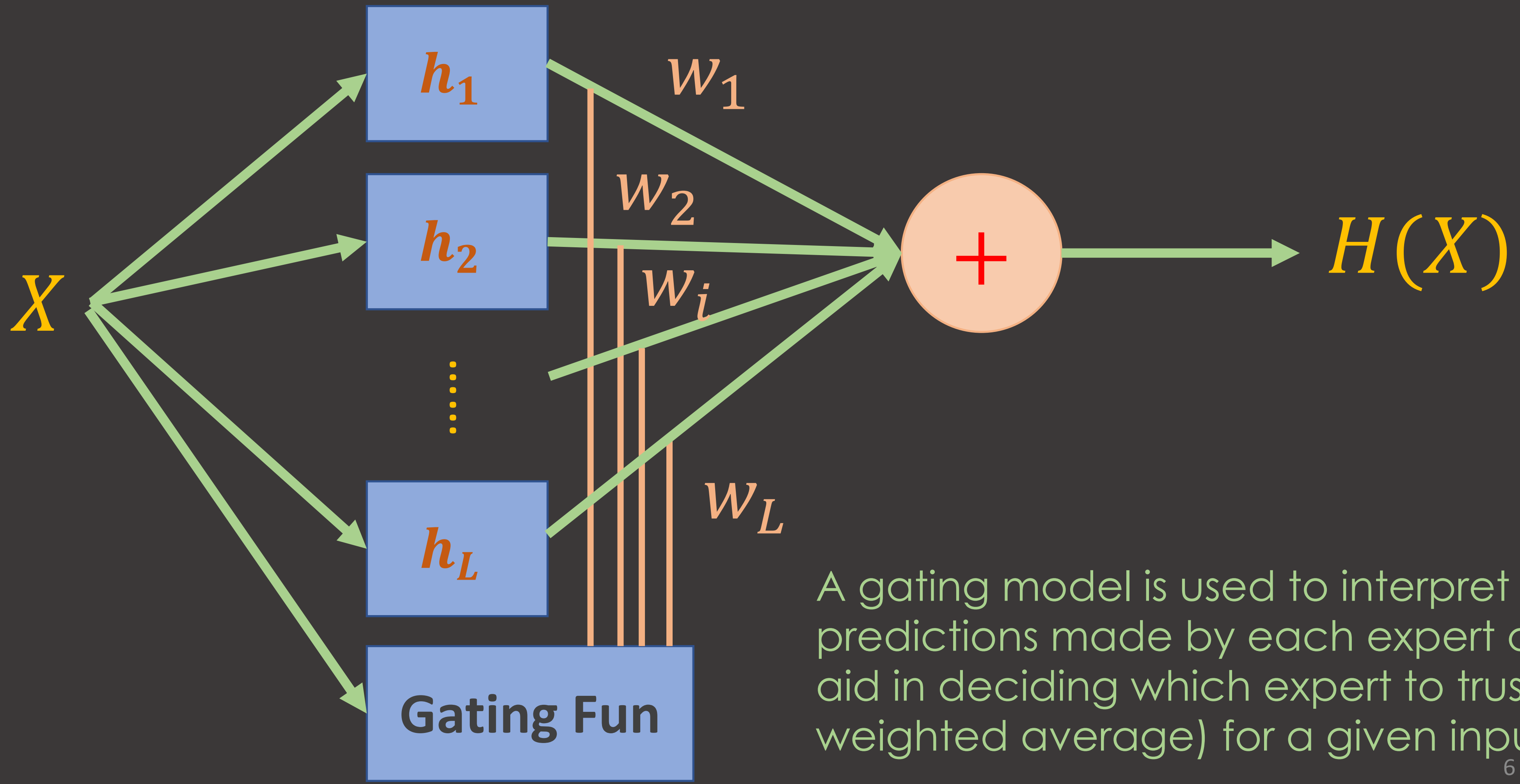
- Final hypothesis is a simple vote of the members

Combining Classifiers: Weighted Average



- Coefficients of individual members are trained using a validation set

Combining Classifiers: Gating

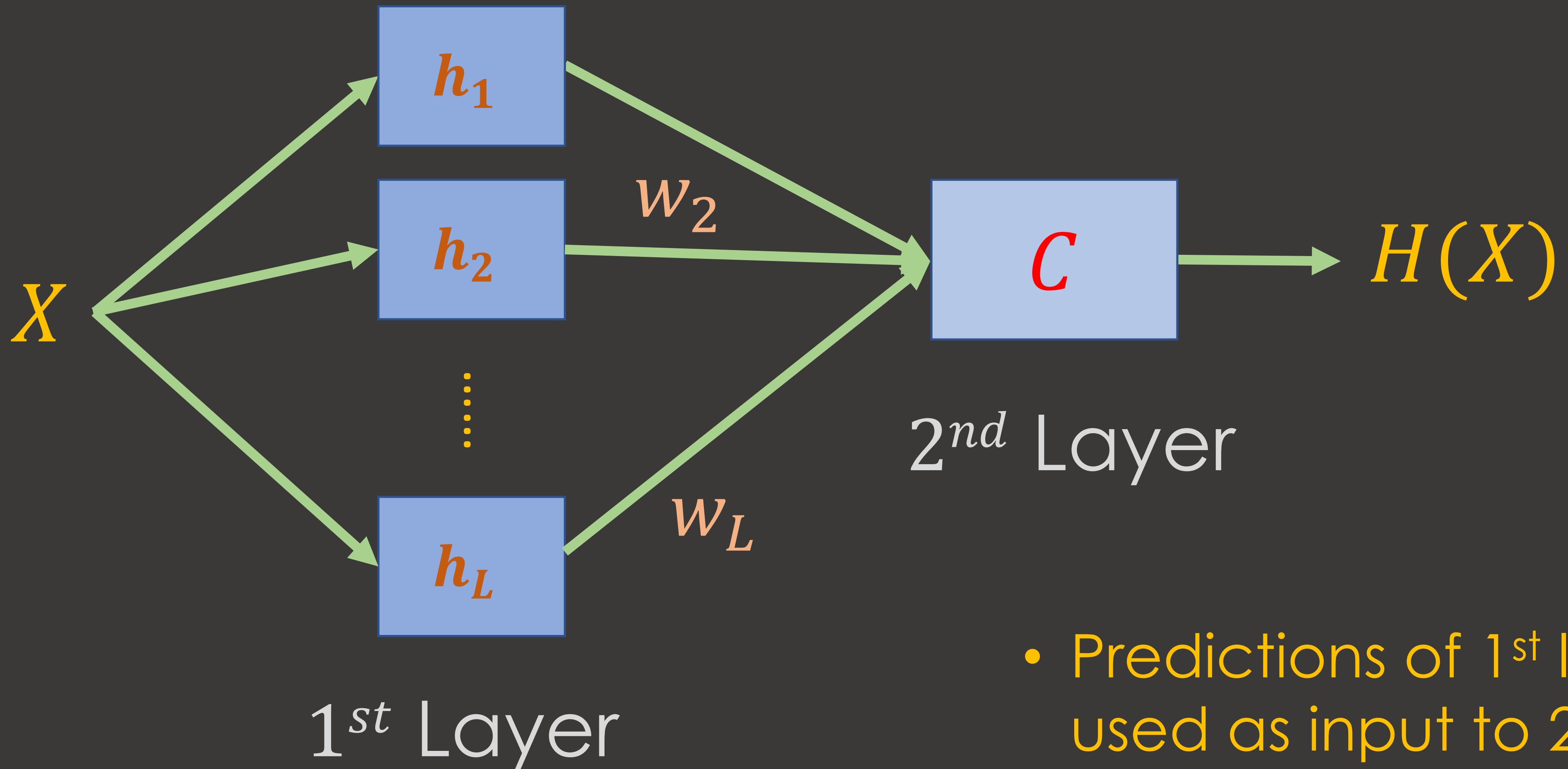


A gating model is used to interpret the predictions made by each expert and to aid in deciding which expert to trust (or weighted average) for a given input.

Combining Classifiers : Ensemble

- There are four elements in the approach:
  - Division of a task into subtasks
    - Replicate the entire dataset
    - Participate training data into subdataset
    - Only train with subfeature
  - Develop an expert for each subtask
  - Use a gating model to decide which/how expert to use
  - Pool predictions and gating model output to make a prediction

Combining Classifiers: Stacking



- Predictions of 1st layer used as input to 2nd layer

How to Achieve Diversity

Cause of the Problem	Diversification Strategy
Pattern was difficult	Hopeless
Overfitting	Vary the training sets
Some features are noisy	Vary the set of input features

Slide credit: Léon Bottou

Manipulating the Training Data

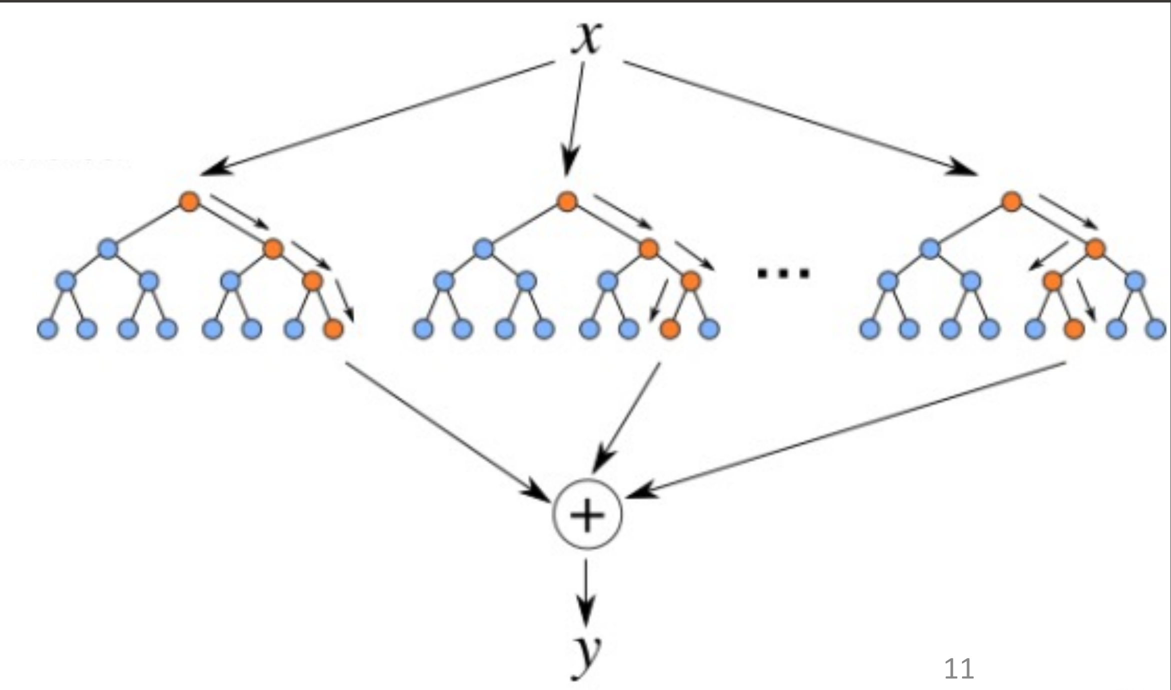
- Bootstrap replication:
  - Given n training examples, construct a new training set by sampling n instances with replacement
  - Excludes ~30% of the training instances

- Bagging:
  - Create bootstrap replicates of training set
  - Train a classifier (e.g., a decision tree) for each replicate
  - Estimate classifier performance using out-of-bootstrap data
  - Average output of all classifiers

Slide credit: Léon Bottou

Manipulating the features

- **Random Forests**
  - Construct decision trees on bootstrap replicas.
    - ❑Restrict the node decisions to a small subset of features picked randomly for each node
  - Do not prune the trees
    - ❑Estimate tree performance on out of bootstrap data
  - Average the output of all trees



Slide credit: Léon Bottou

Decision Stumps

- Consider a decision tree with one root node directly connected to N different leaf nodes
  - The test needs to have N possible outcomes
- Each leaf node is an expert that uses its own particular function to predict the output from the input
- Learning a decision stump is tricky if the test has discreate outcomes because we do not have a continuous space in which to optimize parameters

Creating a continuous search space

- If the test at the root node uses a softmax to assign probabilities to the leaf nodes we get a continuous search space:
  - Small changes to the parameters of the softmax "manager" cause small changes to the expected log probability of predicting the correct answer
  - The standard softmax function $\sigma: \mathbb{R}^K \to (0,1)^K$ is defined when $K \geq 1$ by the formula

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1, \ldots, K \text{ and } z = (z_1, \ldots, z_K) \in \mathbb{R}^K$$

  - $\sigma(0,10) = \sigma_1(0,10) = \left(\frac{1}{1+e^{10}}, \frac{e^{10}}{1+e^{10}}\right) \approx (0.00005, \;\; 0.99995)$

Creating a continuous search space

- A mixture of experts can be viewed as a probabilistic way of viewing a decision stump so that the tests and leaf functions can be learned by maximum likelihood
  - It can be generalized to a full decision tree by having a softmax at each internal node of the tree

AdaBoost [Freund & Schapire, 1997]

- A meta-learning algorithm with great theoretical and empirical performance
- Turns a base learner (i.e., a "weak hypothesis") into a high performance classifier
- Creates an ensemble of weak hypotheses by repeatedly emphasizing mis-predicted instances

AdaBoost

- 1: Initiallize a vector of n uniform weights w_1
- 2: for $t = 1, \ldots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t

$$\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} w_{t,i}$$

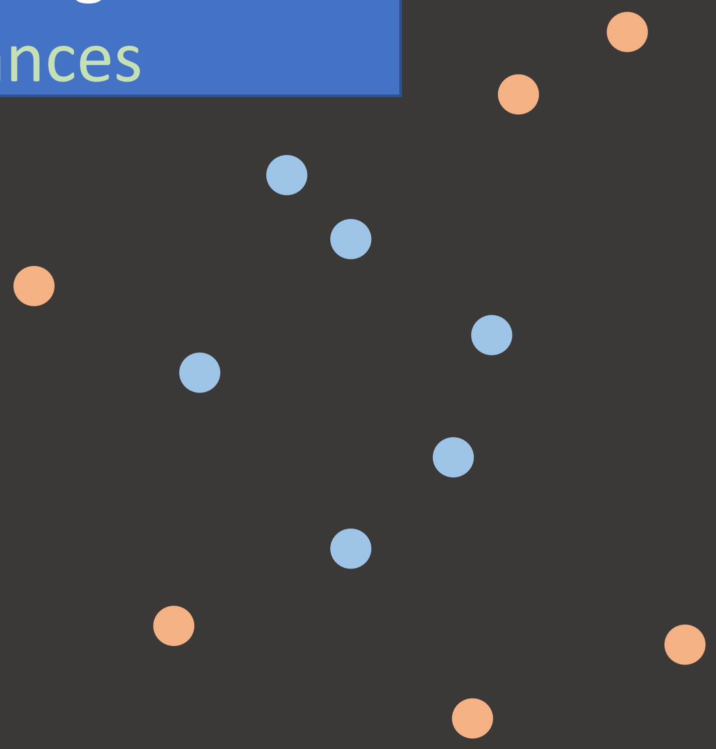
- 5: Choose $\beta_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$

- 7: Normalize w_{t+1} to be a distribution
- 8: end for
- 9: Return the hypothesis

$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

W is the weights for instances



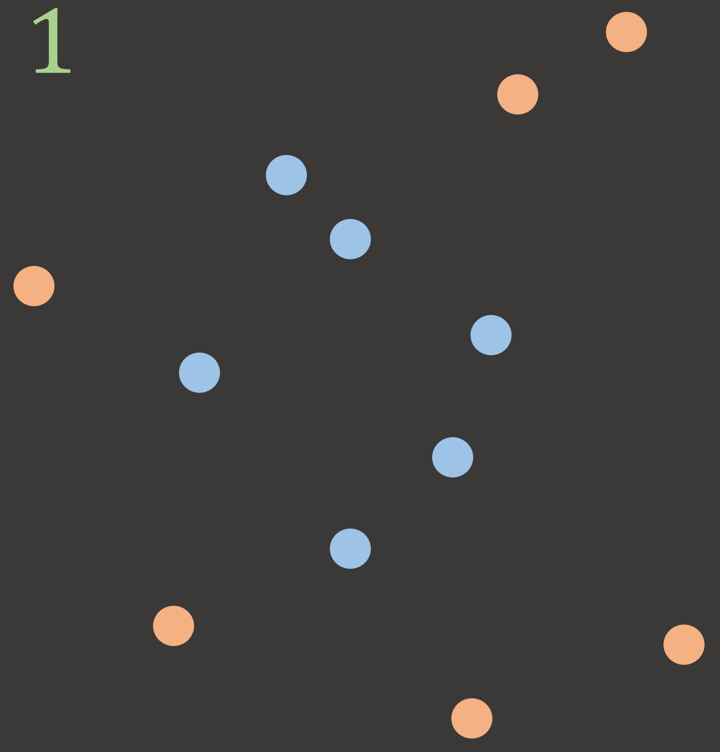
Size of point represents the instance's weight

Each time t, we only train one classifier

AdaBoost

- 1: Initiallize a vector of n uniform weights w_1
- 2: for $t = 1, \dots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t
$$\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} w_{t,i}$$
- 5: Choose $\beta_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
- 6: Update all instance weights:
$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$
- 7: Normalize w_{t+1} to be a distribution
- 8: end for
- 9: Return the hypothesis
$$H(x) = sign\left(\sum_{t=1}^{T} \beta_t h_t(x)\right)$$

$t = 1$



Size of point represents the instance's weight

AdaBoost

- 1: Initiallize a vector of n uniform weights w_1
- 2: for $t = 1, \dots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t

$$\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} w_{t,i}$$

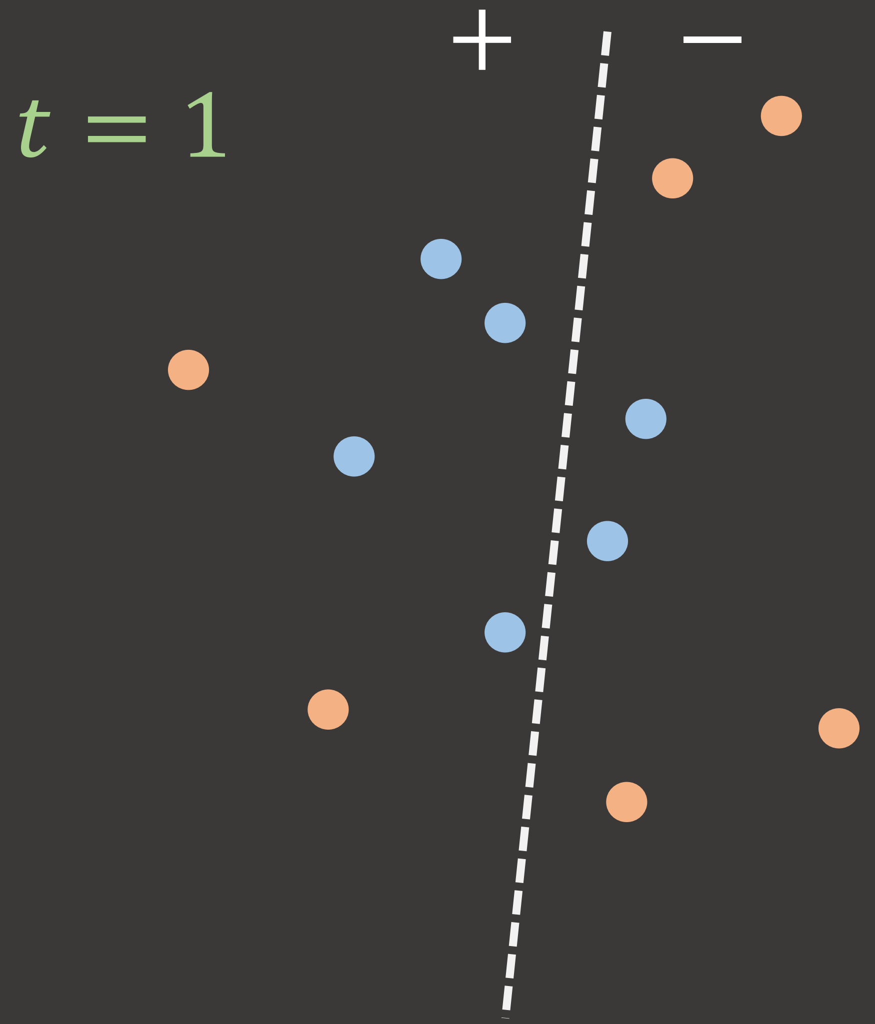
- 5: Choose $\beta_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$

- 7: Normalize w_{t+1} to be a distribution
- 8: end for
- 9: Return the hypothesis

$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

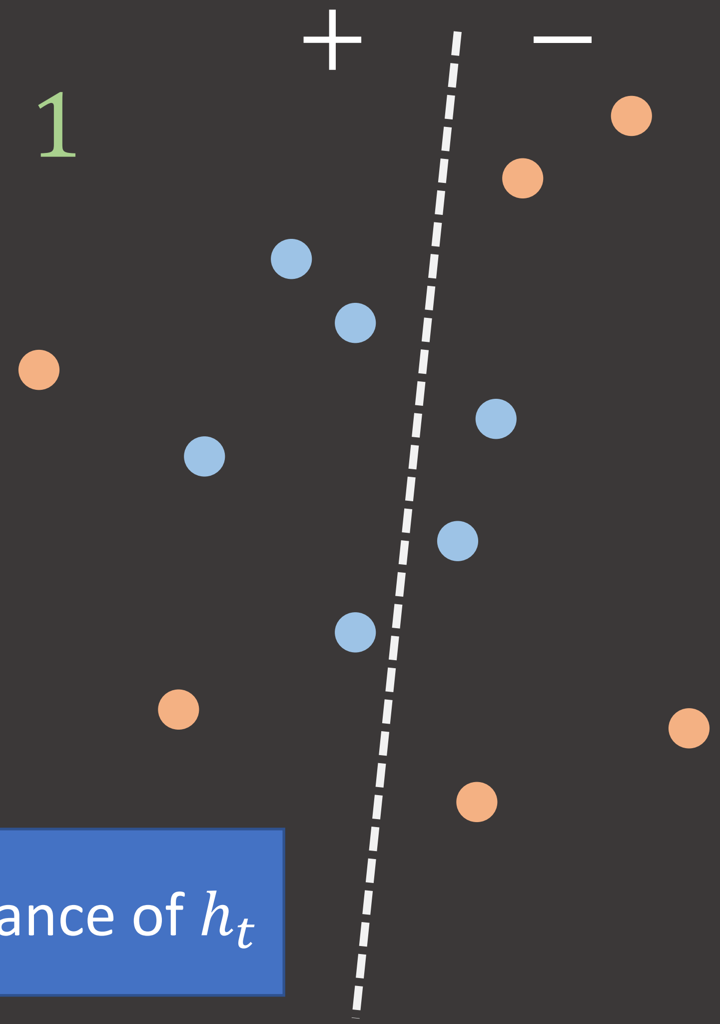
Size of point represents the instance's weight



AdaBoost

- 1: Initiallize a vector of n uniform weights w_1
- 2: for $t = 1, \ldots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t
$$\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} w_{t,i}$$
- 5: Choose $\beta_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:
$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$
- 7: Normalize w_{t+1} to be a distribution
- 8: end for
- 9: Return the hypothesis
$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

$t = 1$



β_t measures the importance of h_t

If $\epsilon_t \leq 0.5$, then $\beta_t \geq 0$ (β_t grows as ϵ_t gets smaller)

AdaBoost

- 1: Initiallize a vector of n uniform weights w_1
- 2: for $t = 1, \ldots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t

$$\epsilon_t = \sum_{i:y_i \neq h_t(x_i)} w_{t,i}$$

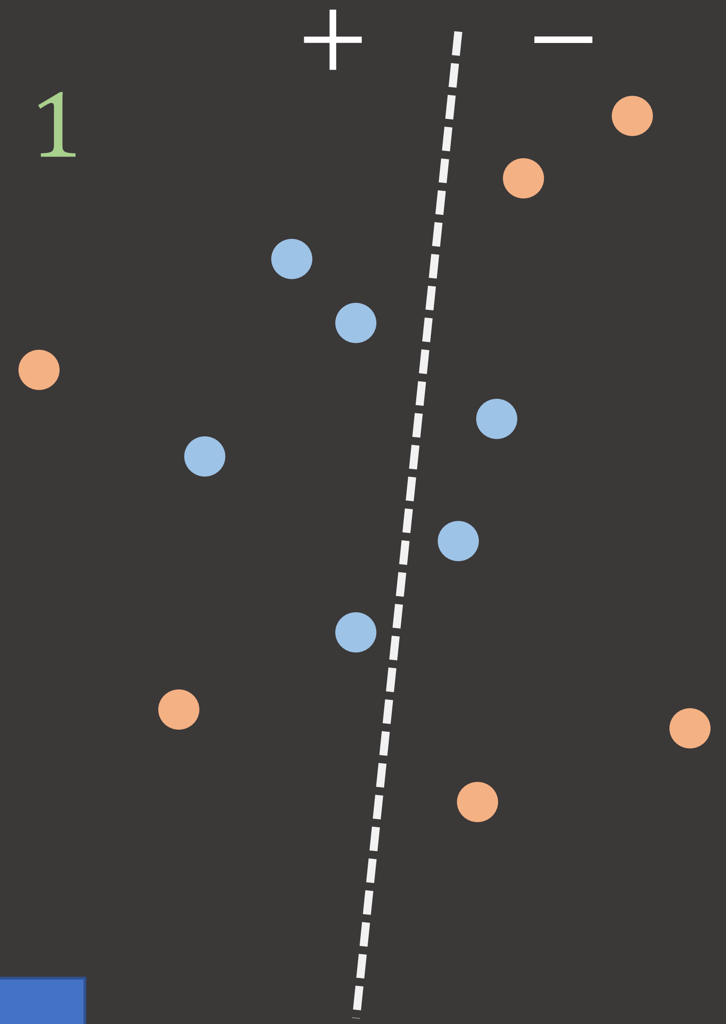
- 5: Choose $\beta_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$

- 7: Normalize w_{t+1} to be a distribution
- 8: end for
- 9: Return the hypothesis

$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

$t = 1$



Weights of correct prediction are multiplied by $e^{-\beta_t} \leq 1$

AdaBoost

- 1: Initiallize a vector of n uniform weights w_1
- 2: for $t \; = \; 1, \dots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t

$$\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} w_{t,i}$$

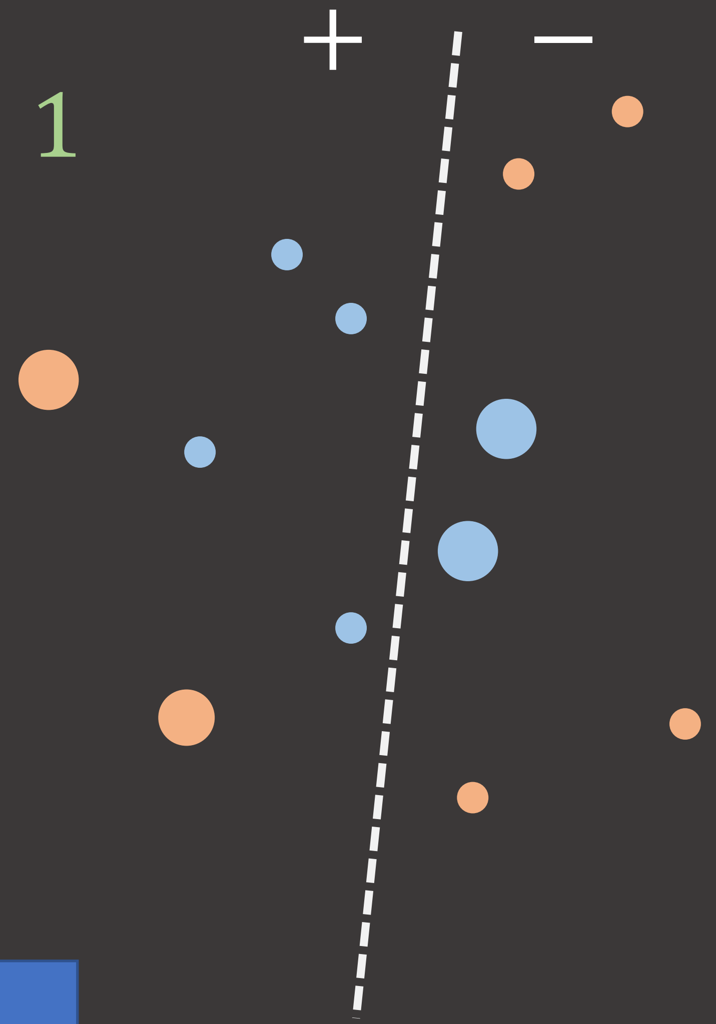
- 5: Choose $\beta_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$

- 7: Normalize w_{t+1} to be a distribution
- 8: end for
- 9: Return the hypothesis

$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

$t = 1$



Disclaimer: Note that resized points in the illustration above are not necessarily to scale with β_t

AdaBoost

- 1: Initiallize a vector of n uniform weights w_1
- 2: for $t = 1, \dots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t

$$\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} w_{t,i}$$

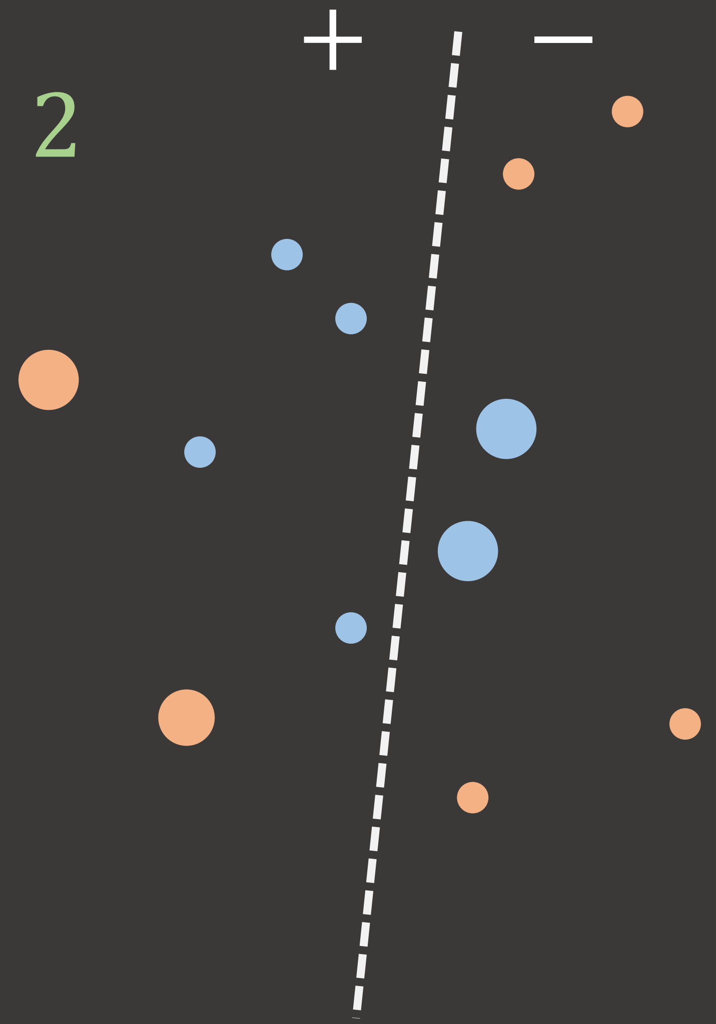
- 5: Choose $\beta_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$

- 7: Normalize w_{t+1} to be a distribution
- 8: end for
- 9: Return the hypothesis

$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

$t = 2$



AdaBoost

- 1: Initiallize a vector of n uniform weights w_1
- 2: for $t = 1, \dots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t

$$\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} w_{t,i}$$

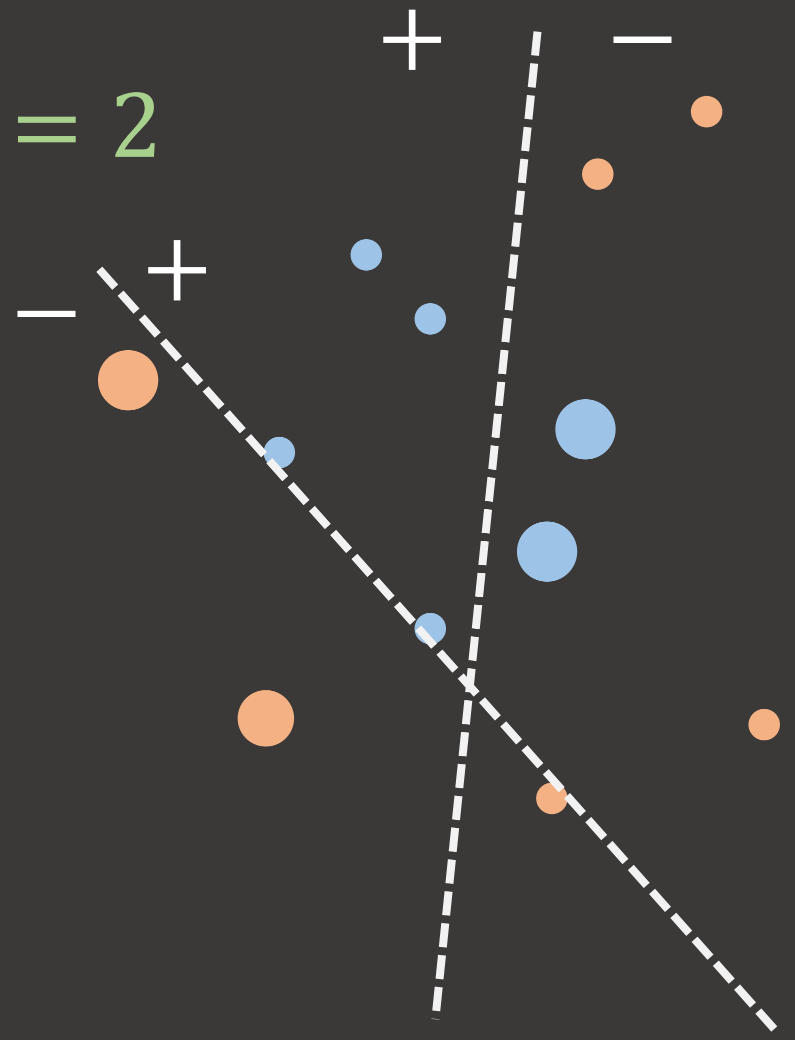
- 5: Choose $\beta_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$

- 7: Normalize w_{t+1} to be a distribution
- 8: end for
- 9: Return the hypothesis

$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

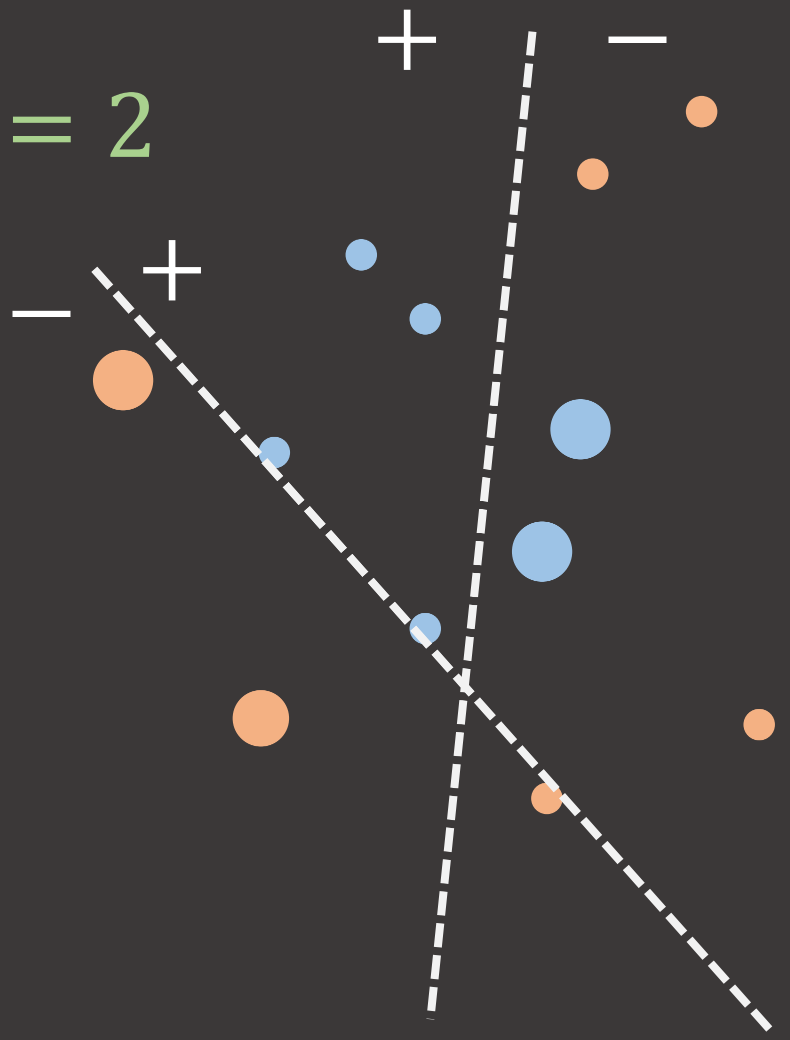
$t = 2$



AdaBoost

- 1: Initiallize a vector of n uniform weights w_1
- 2: for $t = 1, \ldots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t
$$\epsilon_t = \sum_{i:y_i \neq h_t(x_i)} w_{t,i}$$
- 5: Choose $\beta_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:
$$w_{t+1,i} = w_{t,i}e^{-\beta_t y_i h_t(x_i)}$$
- 7: Normalize w_{t+1} to be a distribution
- 8: end for
- 9: Return the hypothesis
$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

$t = 2$



β_t measures the importance of h_t

If $\epsilon_t \leq 0.5$, then $\beta_t \geq 0$ (β_t grows as ϵ_t gets smaller)

AdaBoost

- 1: Initiallize a vector of n uniform weights w_1
- 2: for $t \;=\; 1, \ldots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t

$$\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} w_{t,i}$$

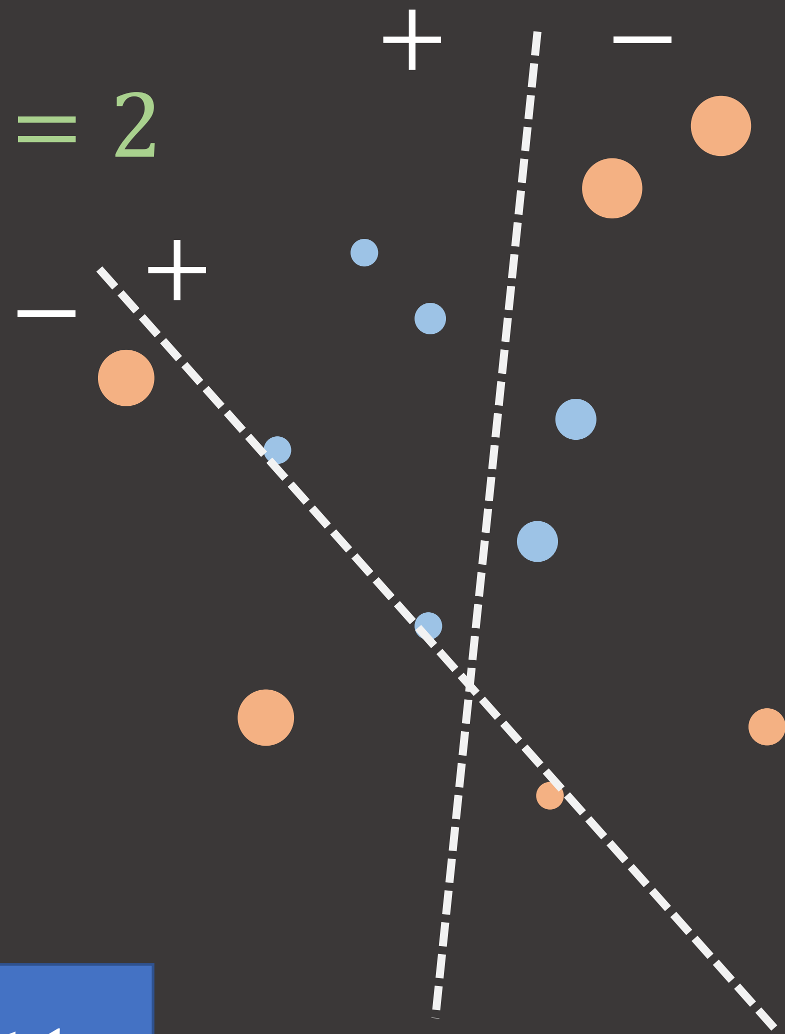
- 5: Choose $\beta_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$

- 7: Normalize w_{t+1} to be a distribution
- 8: end for
- 9: Return the hypothesis

$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

$t = 2$



Weights of correct prediction are multiplied by $e^{-\beta_t} \leq 1$

AdaBoost

- 1: Initiallize a vector of n uniform weights w_1
- 2: for $t = 1, \ldots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t

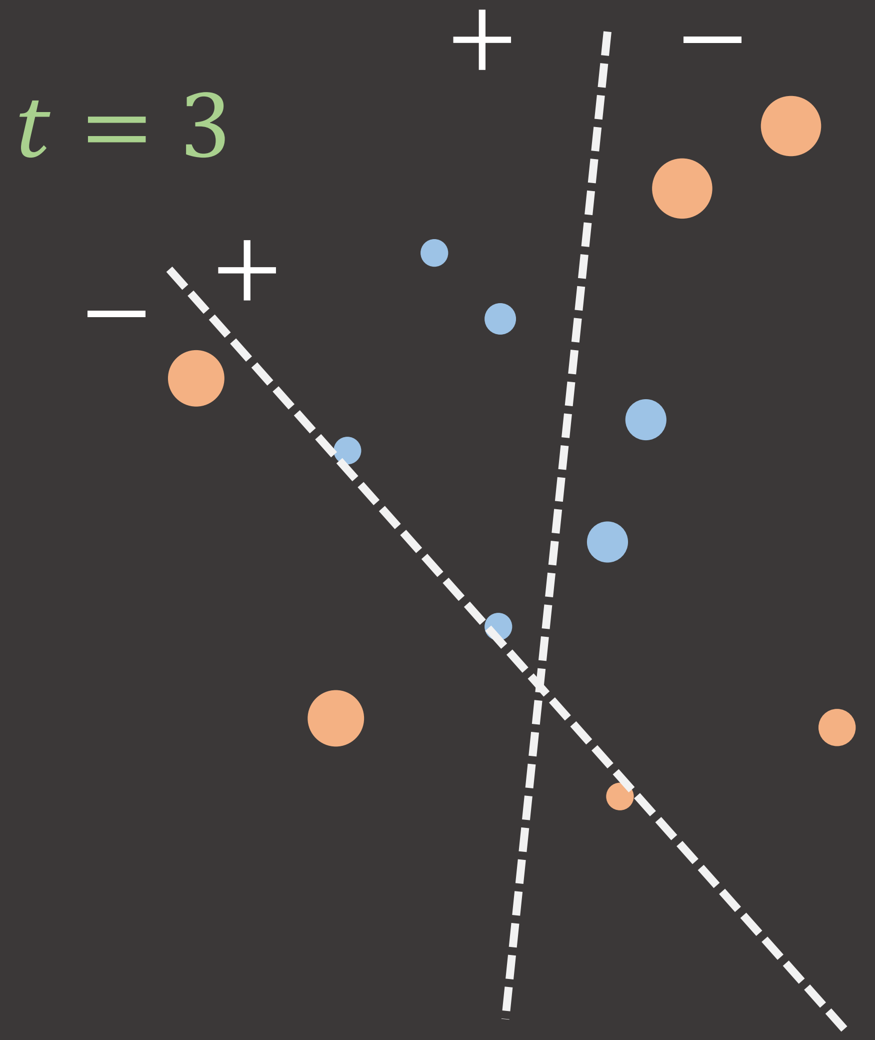
$$\epsilon_t = \sum_{i:y_i \neq h_t(x_i)} w_{t,i}$$

- 5: Choose $\beta_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$

- 7: Normalize w_{t+1} to be a distribution
- 8: end for
- 9: Return the hypothesis

$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$



AdaBoost

- 1: Initiallize a vector of n uniform weights w_1
- 2: for $t = 1, \ldots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t

$$\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} w_{t,i}$$

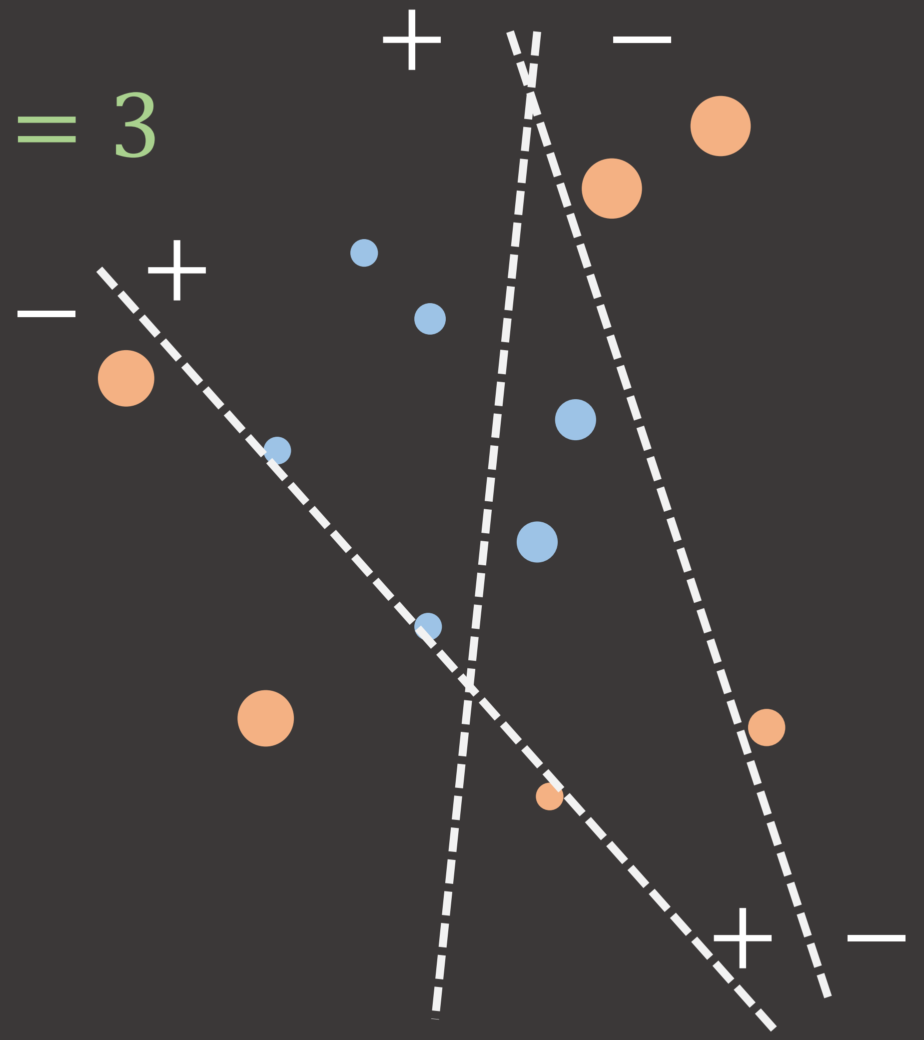
- 5: Choose $\beta_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$

- 7: Normalize w_{t+1} to be a distribution
- 8: end for
- 9: Return the hypothesis

$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

$t = 3$



AdaBoost

- 1: Initiallize a vector of n uniform weights w_1
- 2: for $t = 1, \ldots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t

$$\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} w_{t,i}$$

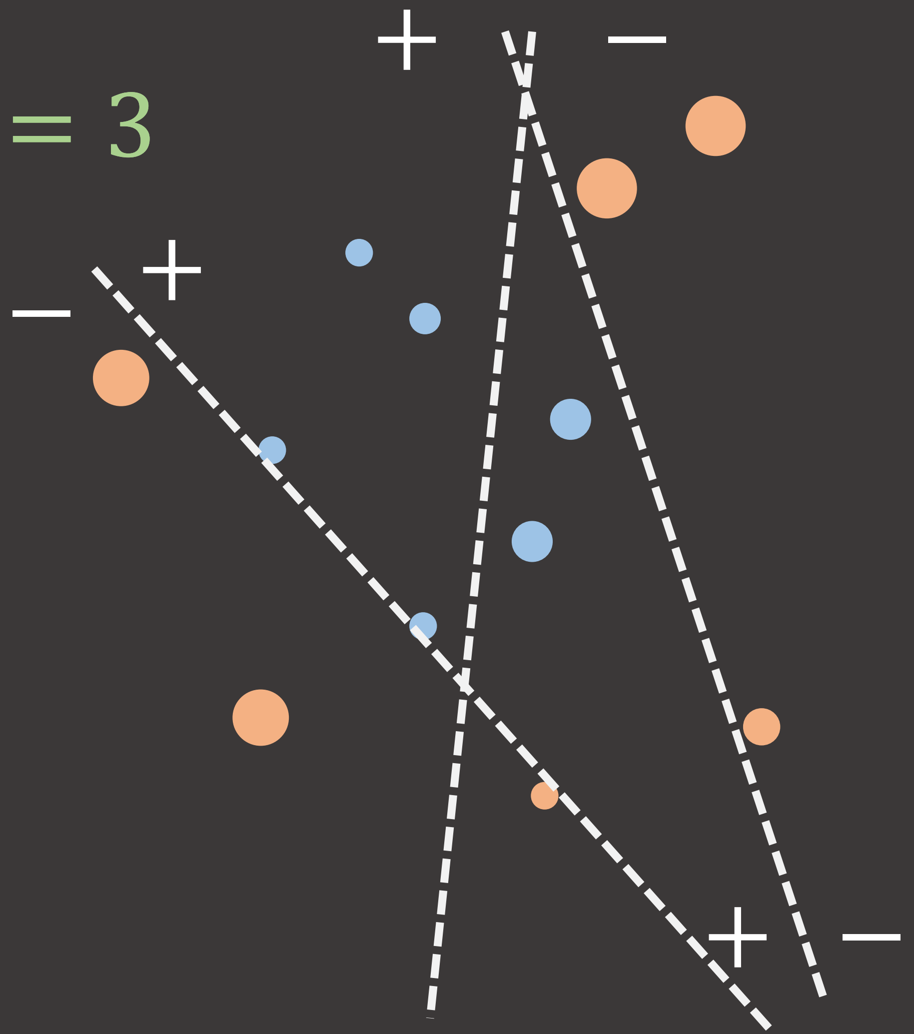
- 5: Choose $\beta_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$

- 7: Normalize w_{t+1} to be a distribution
- 8: end for
- 9: Return the hypothesis

$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

$t = 3$



β_t measures the importance of h_t

If $\epsilon_t \leq 0.5$, then $\beta_t \geq 0$ (β_t grows as ϵ_t gets smaller)

AdaBoost

- 1: Initiallize a vector of n uniform weights w_1
- 2: for $t = 1, \ldots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t

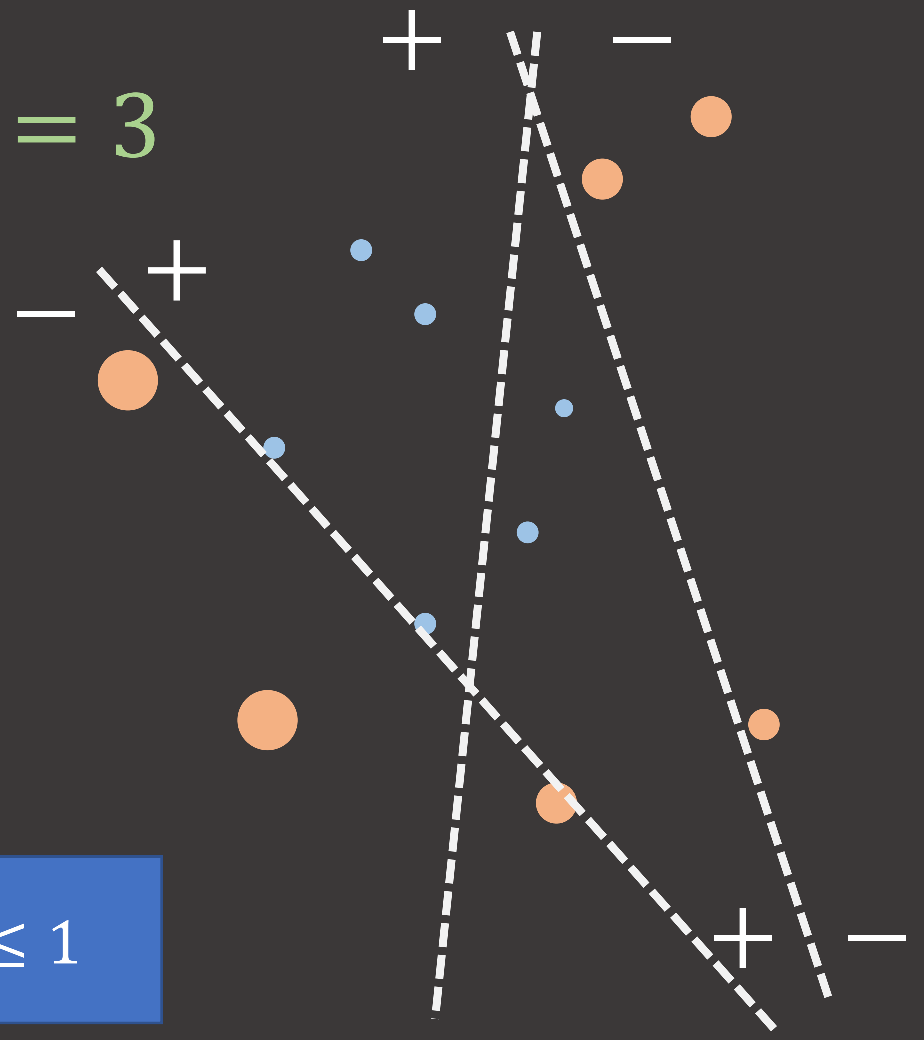
$$\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} w_{t,i}$$

- 5: Choose $\beta_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$\boxed{w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}}$$

- 7: Normalize w_{t+1} to be a distribution
- 8: end for
- 9: Return the hypothesis

$t = 3$



Weights of correct prediction are multiplied by $e^{-\beta_t} \leq 1$

Weights of correct prediction are multiplied by $e^{-\beta_t} \geq 1$

AdaBoost

- 1: Initiallize a vector of n uniform weights w_1
- 2: for $t = 1, \dots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t

$$\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} w_{t,i}$$

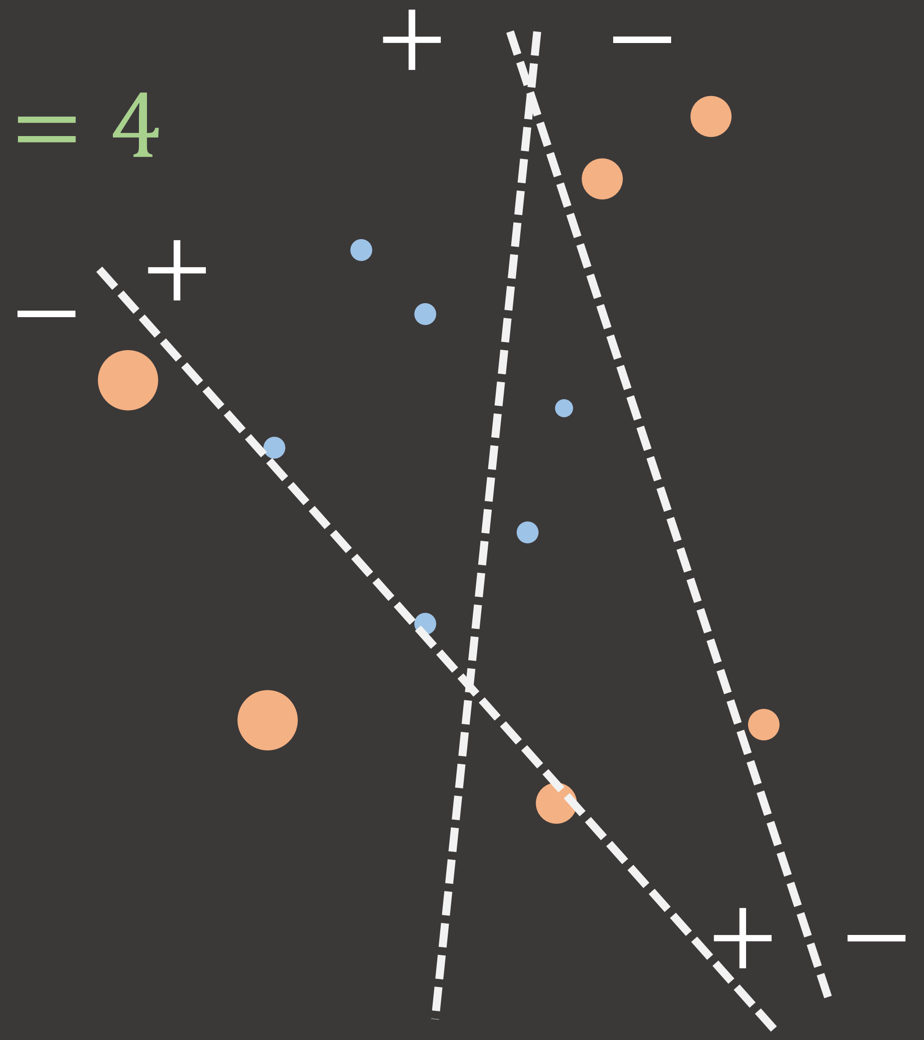
- 5: Choose $\beta_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$

- 7: Normalize w_{t+1} to be a distribution
- 8: end for
- 9: Return the hypothesis

$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

$t = 4$



AdaBoost

- 1: Initiallize a vector of n uniform weights w_1
- 2: for $t = 1, \dots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t

$$\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} w_{t,i}$$

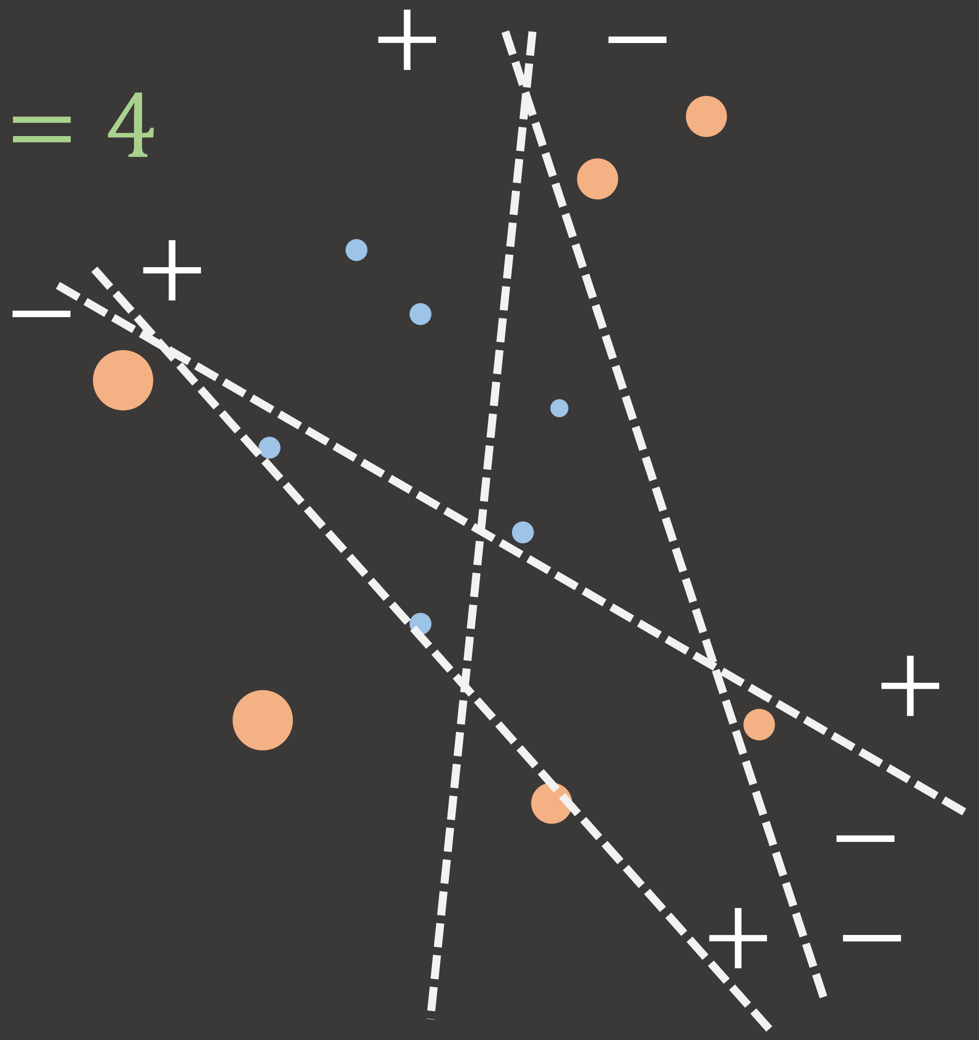
- 5: Choose $\beta_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$

- 7: Normalize w_{t+1} to be a distribution
- 8: end for
- 9: Return the hypothesis

$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

$t = 4$



AdaBoost

- 1: Initiallize a vector of n uniform weights w_1
- 2: for $t = 1, \ldots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t

$$\epsilon_t = \sum_{i:y_i \neq h_t(x_i)} w_{t,i}$$

- 5: Choose $\beta_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i}e^{-\beta_t y_i h_t(x_i)}$$

- 7: Normalize w_{t+1} to be a distribution
- 8: end for
- 9: Return the hypothesis

$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

$t = 4$

AdaBoost

- 1: Initiallize a vector of n uniform weights w_1
- 2: for $t = 1, \ldots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t

$$\epsilon_t = \sum_{i:y_i \neq h_t(x_i)} w_{t,i}$$

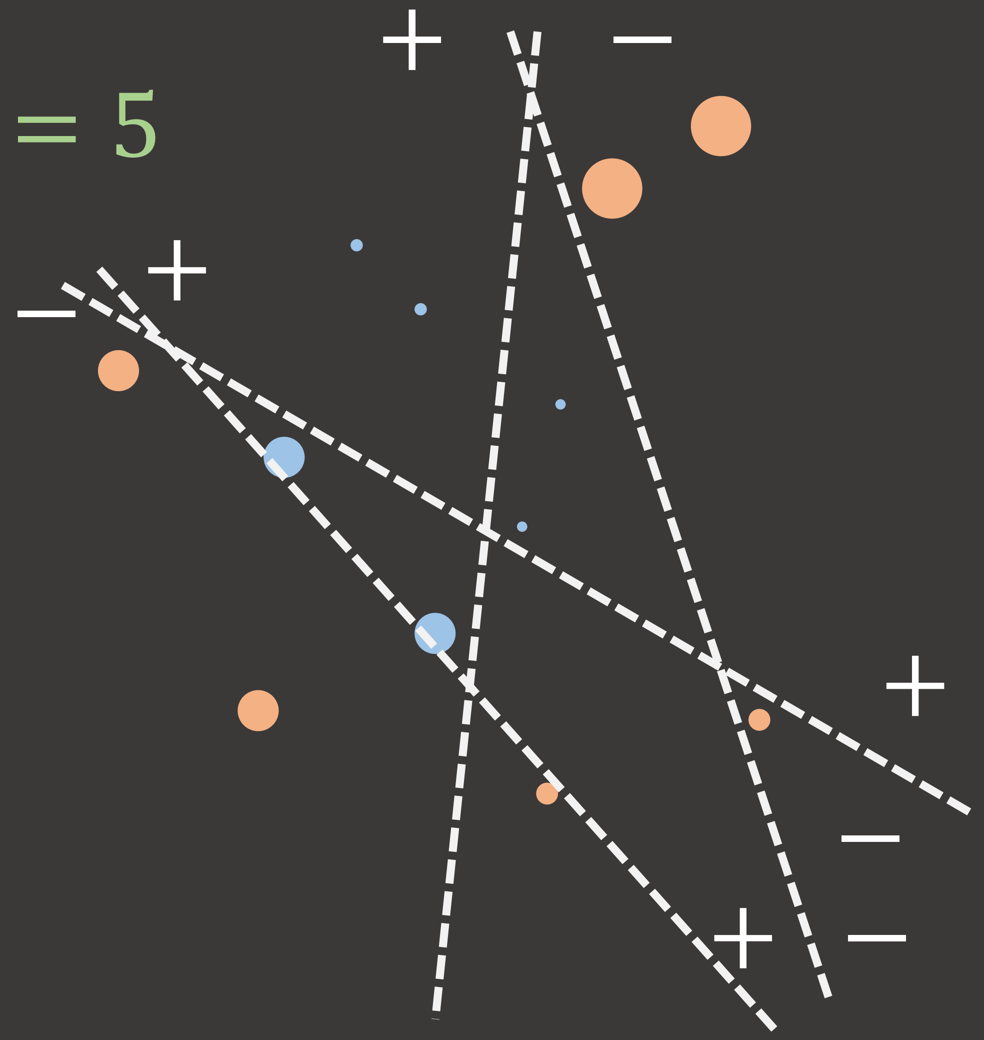
- 5: Choose $\beta_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$

- 7: Normalize w_{t+1} to be a distribution
- 8: end for
- 9: Return the hypothesis

$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

$t = 5$



AdaBoost

- 1: Initiallize a vector of n uniform weights w_1
- 2: for $t = 1, \ldots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t

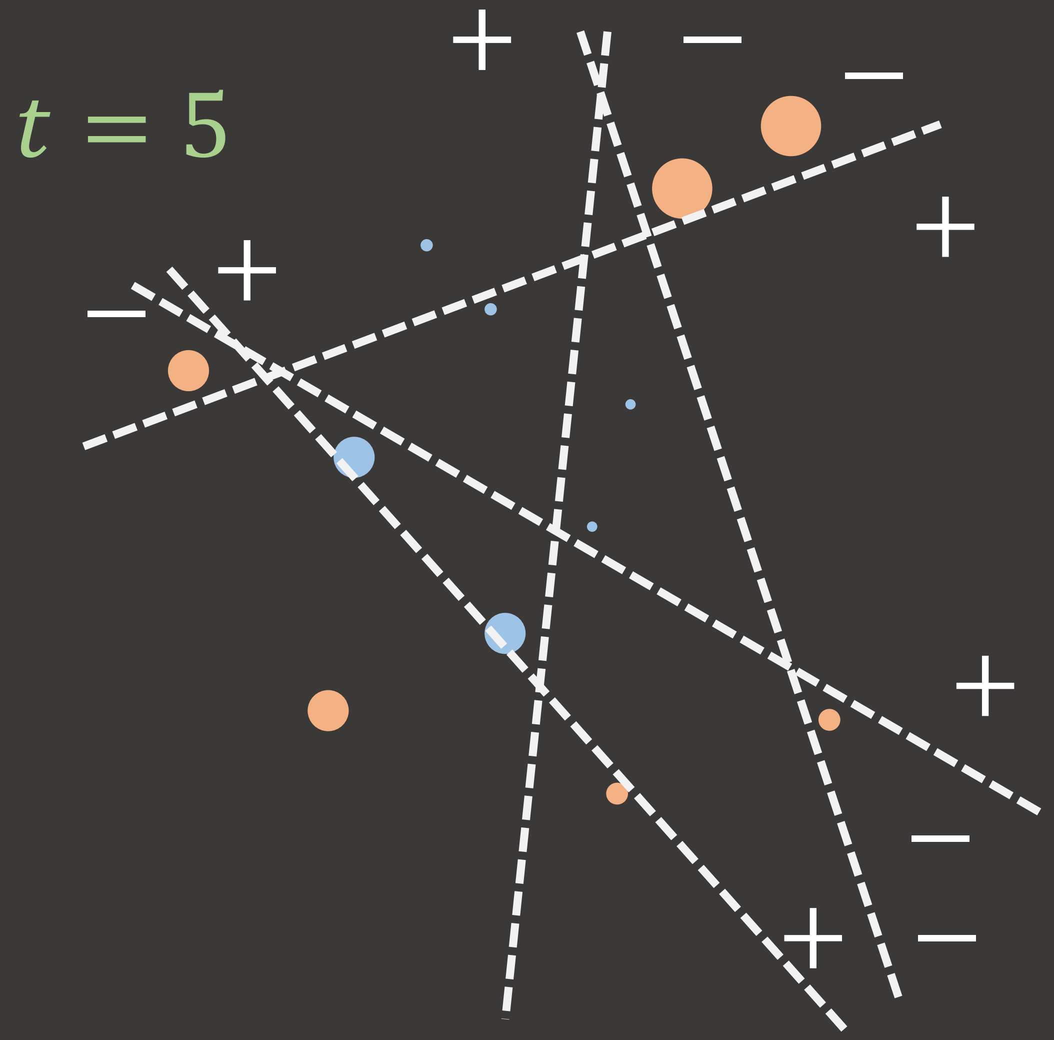
$$\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} w_{t,i}$$

- 5: Choose $\beta_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$

- 7: Normalize w_{t+1} to be a distribution
- 8: end for
- 9: Return the hypothesis

$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$



AdaBoost

- 1: Initiallize a vector of n uniform weights w_1
- 2: for $t = 1, \ldots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t

$$\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} w_{t,i}$$

- 5: Choose $\beta_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$

- 7: Normalize w_{t+1} to be a distribution
- 8: end for
- 9: Return the hypothesis

$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

$t = 5$

AdaBoost

- 1: Initiallize a vector of n uniform weights w_1
- 2: for $t = 1, \dots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t

$$\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} w_{t,i}$$

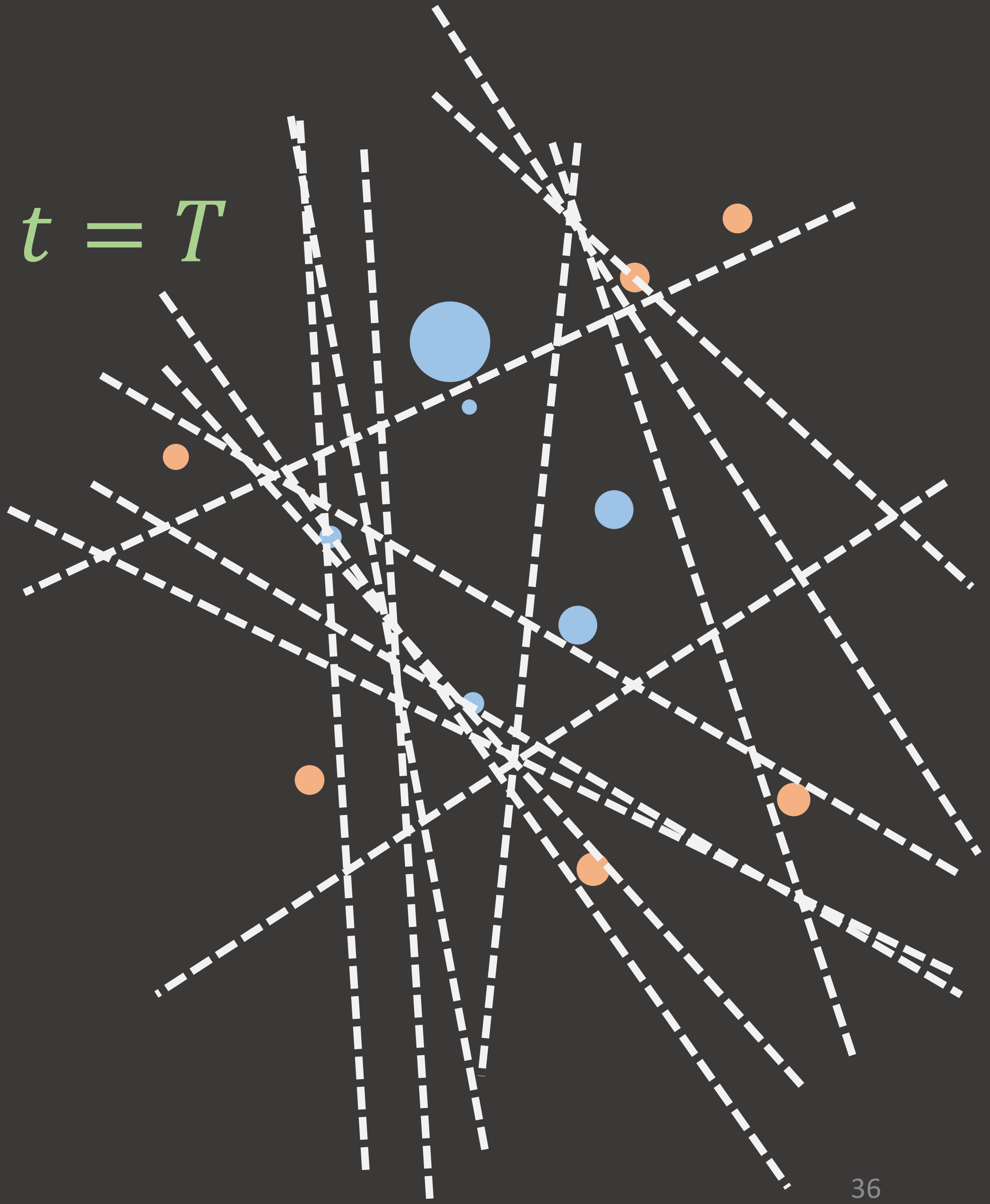
- 5: Choose $\beta_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$

- 7: Normalize w_{t+1} to be a distribution
- 8: end for
- 9: Return the hypothesis

$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

$t = T$



AdaBoost

- 1: Initiallize a vector of n uniform weights w_1
- 2: for $t = 1, \dots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t

$$\epsilon_t = \sum_{i:y_i \neq h_t(x_i)} w_{t,i}$$

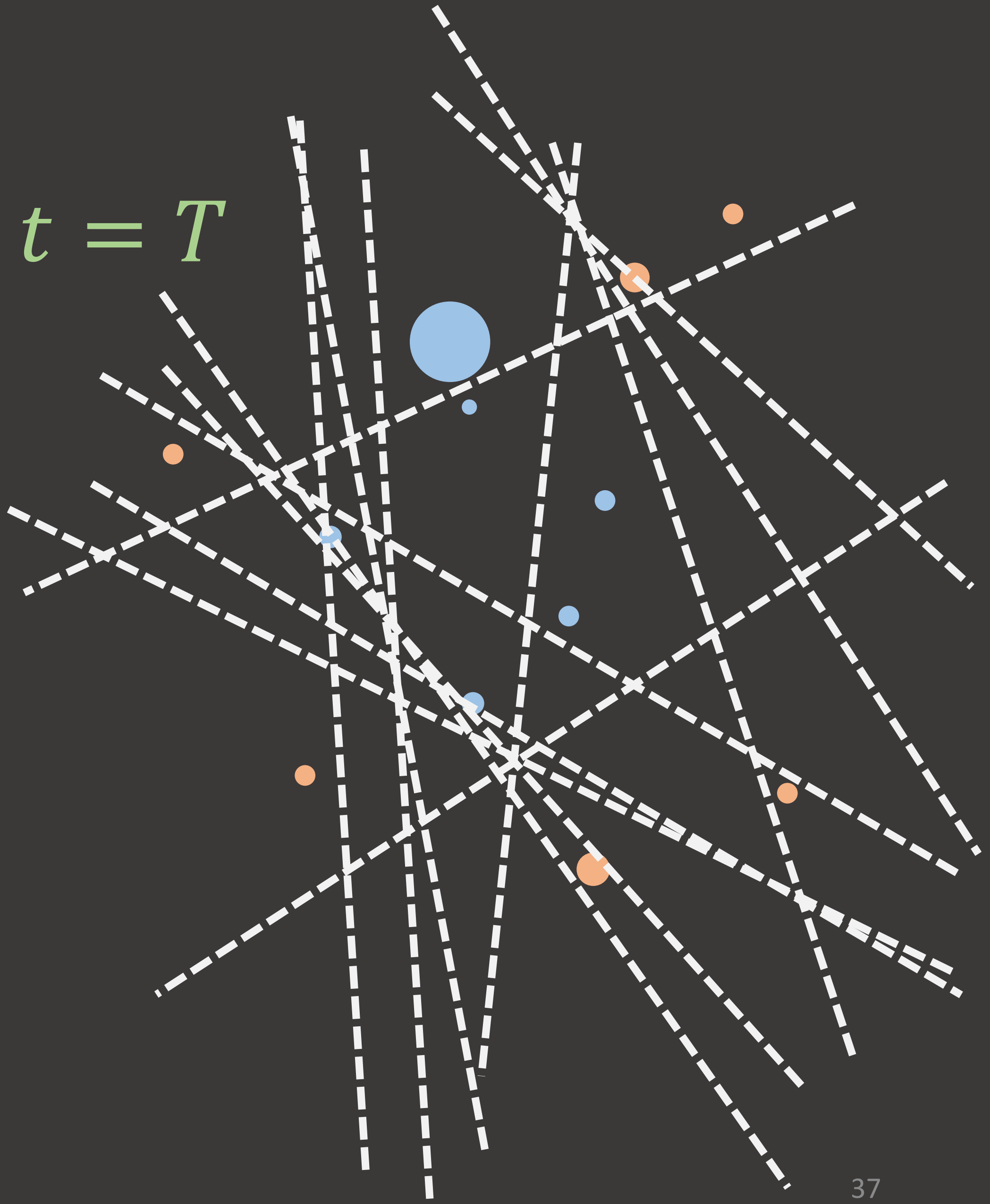
- 5: Choose $\beta_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$

- 7: Normalize w_{t+1} to be a distribution
- 8: end for
- 9: Return the hypothesis

$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

$t = T$



AdaBoost

- 1: Initiallize a vector of n uniform weights w_1
- 2: for $t = 1, \ldots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t

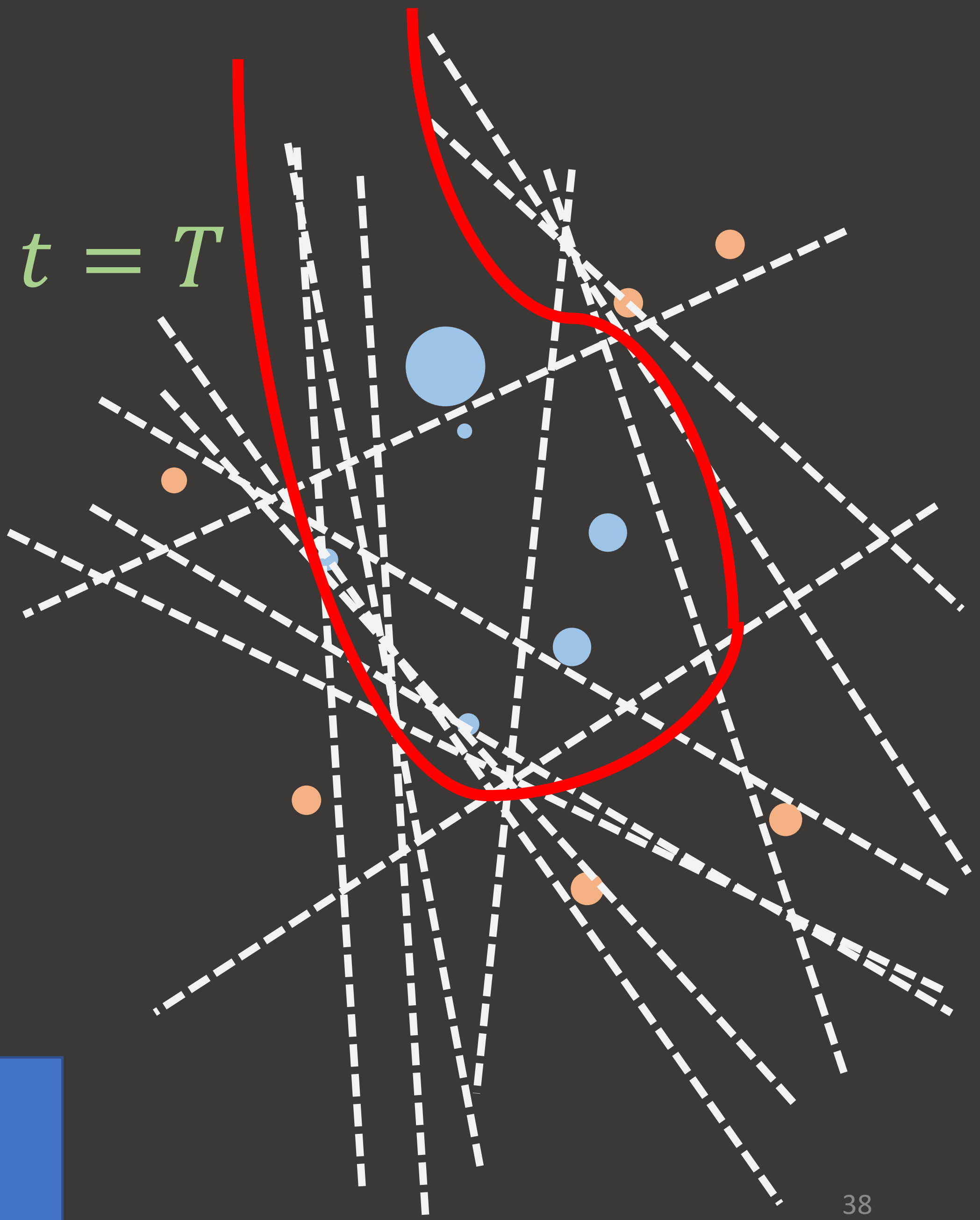
$$\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} w_{t,i}$$

- 5: Choose $\beta_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$

- 7: Normalize w_{t+1} to be a distribution
- 8: end for
- 9: Return the hypothesis
  $H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$

$t = T$



Final model is weighted combination of members
Each member weighted by its importance

AdaBoost

INPUT : training data X, y $= \{(x_i, y_i)\}_{i=1}^{n}$

- 1: Initiallize a vector of n uniform instance weights $w_1 = [\frac{1}{n}, \dots, \frac{1}{n}]$
- 2: for $t = 1, \dots, T$
- 3: Train model h_t on X, y with weights w_t
- 4: Compute the weighted training error of h_t:

$$\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} w_{t,i}$$

- 5: Choose $\beta_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$

- 7: Normalize w_{t+1} to be a distribution

$$w_{t+1,i} = \frac{w_{t+1,i}}{\sum_{j=1}^{n} w_{t+1,j}} \quad \forall i = 1, \dots, n$$

- 8: end for
- 9: Return the hypothesis

$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

w_t is a vector of weights over the instances at iteration t

All points start with equal weight

How can we deal with algorithms that are not directly support weights for instance?

Training a Model with Weighted Instances

- For algorithms like logistic regression, can simply incorporate weights w into the cost function
  - Essentially, weigh the cost of misclassification differently for Each instance

$$J_{reg}(\theta) = -\sum_{i=1}^{n} w_i[y_i \log h_\theta(x_i) + (1 - y_i)\log(1 - h_\theta(x_i))]$$

- For algorithms that don't directly support instance weights (e.g., ID3 decision Trees, etc.), use weighted bootstrap sampling
  - Form training set by resampling instances with replacement according to w

Base Learner Requirements

- AdaBoost works best with "weak" learners
  - Should not be complex
  - Typically high bias classifiers
  - Works even when weak learner has an error rate just slightly under 0.5 (i.e., just slightly better than random)
    - Can prove training error goes to 0 in $O(\log n)$ iterations

- Examples
  - Decision stumps (1 level decision trees)
  - Depth-limited decision trees
  - Linear classifiers

AdaBoost

INPUT : training data X, y $= \{(x_i, y_i)\}_{i=1}^{n}$

- 1: Initiallize a vector of n uniform weights $w_1 = [\frac{1}{n}, \dots, \frac{1}{n}]$
- 2: for $t = 1, \dots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t:

$$\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} w_{t,i}$$

- 5: Choose $\beta_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$

- 7: Normalize w_{t+1} to be a distribution

$$w_{t+1,i} = \frac{w_{t+1,i}}{\sum_{j=1}^{n} w_{t+1,j}} \quad \forall i = 1, \dots, n$$

- 8: end for
- 9: Return the hypothesis

$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

Error is the sum the weights of all misclassified instances

AdaBoost

INPUT : training data X, y $= \{(x_i, y_i)\}_{i=1}^{n}$

- 1: Initiallize a vector of n uniform weights $w_1 = [\frac{1}{n}, \dots, \frac{1}{n}]$
- 2: for $t = 1, \dots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t:

  $$\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} w_{t,i}$$
- 5: Choose $\beta_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

  $$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$
- 7: Normalize w_{t+1} to be a distribution

  $$w_{t+1,i} = \frac{w_{t+1,i}}{\sum_{j=1}^{n} w_{t+1,j}} \quad \forall i = 1, \dots, n$$
- 8: end for
- 9: Return the hypothesis

  $$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

- β_t measures the importance of h_t
- If $\epsilon_t \leq 0.5$, then $\beta_t \geq 0$
  - Trivial, otherwise flip h_t's Predictions
- β_t grows as error h_t's shrinks

AdaBoost

INPUT : training data X, y $= \{(x_i, y_i)\}_{i=1}^{n}$

- 1: Initiallize a vector of n uniform weights $w_1 = [\frac{1}{n}, \ldots, \frac{1}{n}]$
- 2: for $t = 1, \ldots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t:
  $$\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} w_{t,i}$$
- 5: Choose $\beta_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:
  $$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$
- 7: Normalize w_{t+1} to be a distribution
  $$w_{t+1,i} = \frac{w_{t+1,i}}{\sum_{j=1}^{n} w_{t+1,j}} \quad \forall i = 1, \ldots, n$$
- 8: end for
- 9: Return the hypothesis
  $$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

Will be ≤ 1

This is the same as:

$$w_{t+1,i} = w_{t,i} \times \begin{cases} e^{-\beta\delta} & if\ h_t(x_i) = y_i \\ e^{\beta\delta} & if\ h_t(x_i) \neq y_i \end{cases}$$

Will be ≥ 1

Essentially, this emphasizes misclassified instances.

AdaBoost

INPUT : training data X, y $= \{(x_i, y_i)\}_{i=1}^{n}$

- 1: Initiallize a vector of n uniform weights $w_1 = [\frac{1}{n}, \dots, \frac{1}{n}]$
- 2: for $t = 1, \dots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t:

$$\epsilon_t = \sum_{i:y_i \neq h_t(x_i)} w_{t,i}$$

- 5: Choose $\beta_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$

- 7: Normalize w_{t+1} to be a distribution

$$w_{t+1,i} = \frac{w_{t+1,i}}{\sum_{j=1}^{n} w_{t+1,j}} \quad \forall i = 1, \dots, n$$

- 8: end for
- 9: Return the hypothesis

$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

Make w_{t+1} sum to 1

AdaBoost

INPUT : training data X, y $= \{(x_i, y_i)\}_{i=1}^{n}$

- 1: Initiallize a vector of n uniform weights $w_1 = [\frac{1}{n}, \dots, \frac{1}{n}]$
- 2: for $t = 1, \dots, T$
- 3: Train model h_t on X, y with instance weights w_t
- 4: Compute the weighted training error of h_t:

$$\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} w_{t,i}$$

- 5: Choose $\beta_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} e^{-\beta_t y_i h_t(x_i)}$$

- 7: Normalize w_{t+1} to be a distribution

$$w_{t+1,i} = \frac{w_{t+1,i}}{\sum_{j=1}^{n} w_{t+1,j}} \quad \forall i = 1, \dots, n$$

- 8: end for
- 9: Return the hypothesis

$$H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$$

Member classifiers with less error are given more weight in the final ensemble hypothesis

Final prediction is a weighted combination of each member's prediction

Dynamic Behavior of AdaBoost

- If a point is repeatedly misclassified…
  - Each time, its weight is increased
  - Eventually it will be emphasized enough to generate a hypothesis that correctly predicts it

- Successive member hypotheses focus on the hardest parts of the instance space
  - Instances with highest weight are often outliers

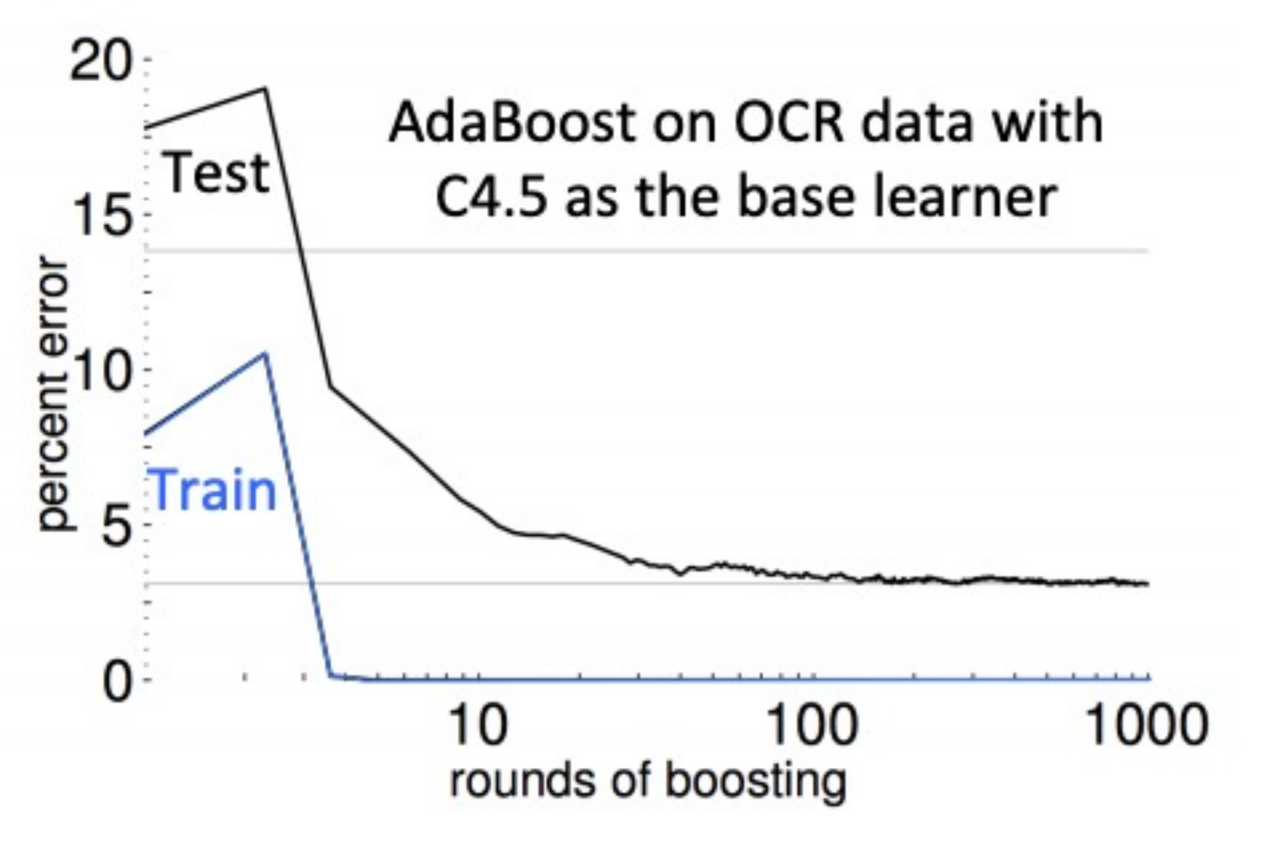
AdaBoost and Overfitting

- In theory, it predicted that AdaBoost *would always overfit* as T grew large
  - Hypothesis keeps growing more complex

- In practice, AdaBoost *often did not overfit*, contradicting the theory

- Also, AdaBoost does not explicitly regularize the model

Explaining Why AdaBoost Works



- Empirically, boosting resists overfitting
- Note that it continues to drive down the test error even ***after*** the training error reaches zero

Figure from Schapire: " Explaining AdaBoost"

AdaBoost in Practice

Strength:

- Fast and simple to program
- No parameters to tune (besides T)
- No assumptions on weak learner

When boosting can fail:

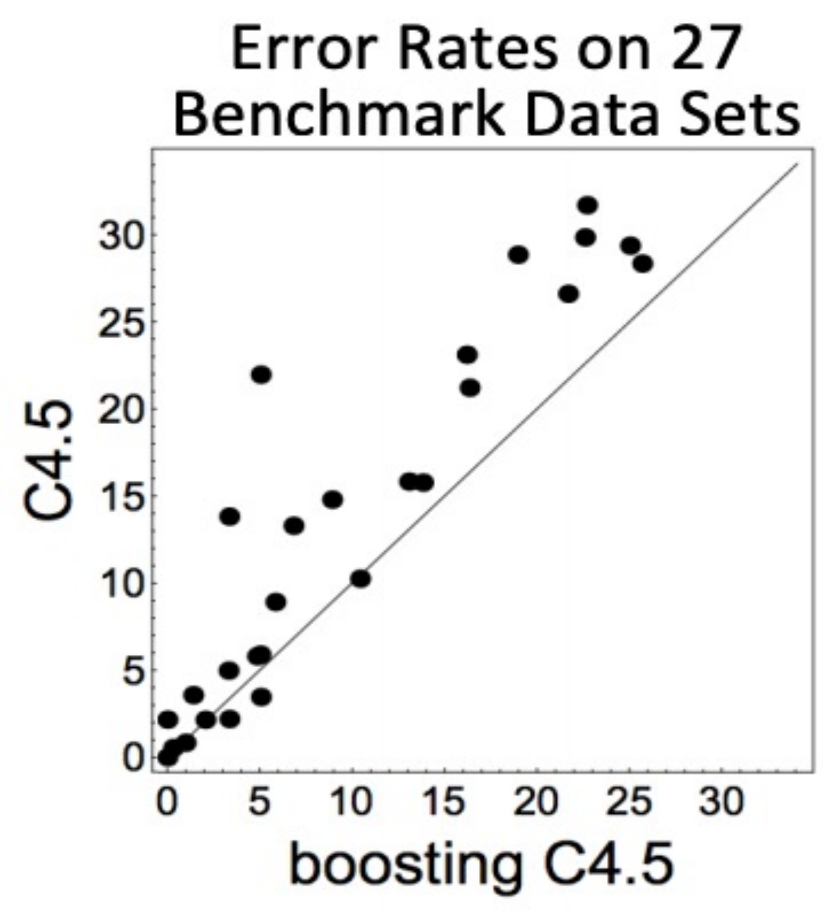
- Given insufficient data
- Overly complex weak hypotheses
- Can be susceptible to noise
- When there are a large number of outliers

Boosted Decision Trees

- Boosted decision trees are one of the best "off-the-shelf" classifiers
    - i.e., no parameter tuning

- Limit member hypothesis complexity by limiting tree depth

- Gradient boosting methods are typically used with trees in practice



"AdaBoost with trees is the best off-the-shelf classifier in the world" –Breiman, 1996
(Also, see results by Caruana & Niculescu-Mizil, ICML 2006)

Figure from Freud and Schapire: "A short introduction to Boosting"