Introduction to Machine Learning
Ensemble Learning

Ashley Gao

William & Mary

November 25, 2023

Ensemble Learning

- Consider a set of classifiers $h_1, ..., h_L$
- Idea: construct a classifier $H(\boldsymbol{x})$ that combines the individual decisions of $h_1, ..., h_L$
  - e.g. could have the member classifiers vote
  - e.g. could use different members for different regions of the instance space
  - works well if the members each have low error rates
- Successful ensembles require diversity
  - Classifiers should make different mistakes
 - Can have different types of base learners

Practical Application: Netflix Prize

- Goal: predict how a user will rate a movie
  - Based on the user's ratings for other movies
  - and other people's 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 the Netflix' system (2007-2009)
- Winner: Bellkor's Pragmatic Chaos
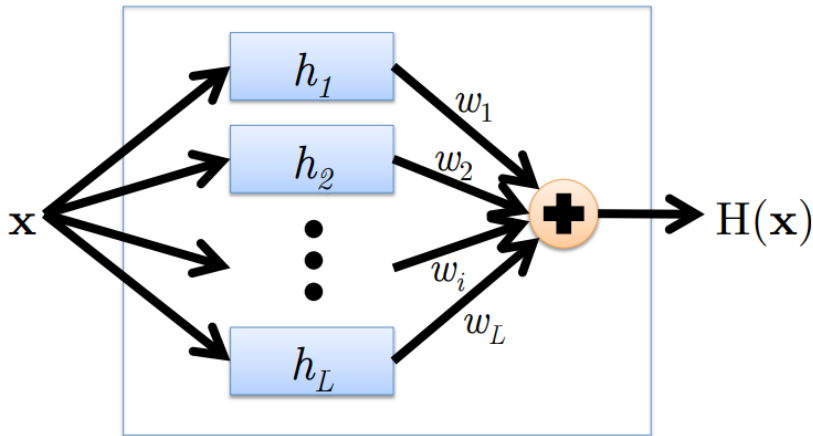 - An ensemble of more than 800 rating systems

Combining Classifiers: Averaging

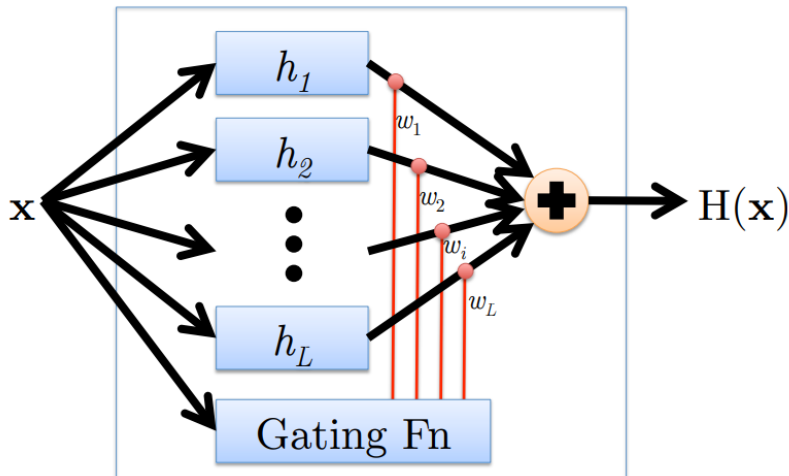- Final hypothesis is a simple vote of the members
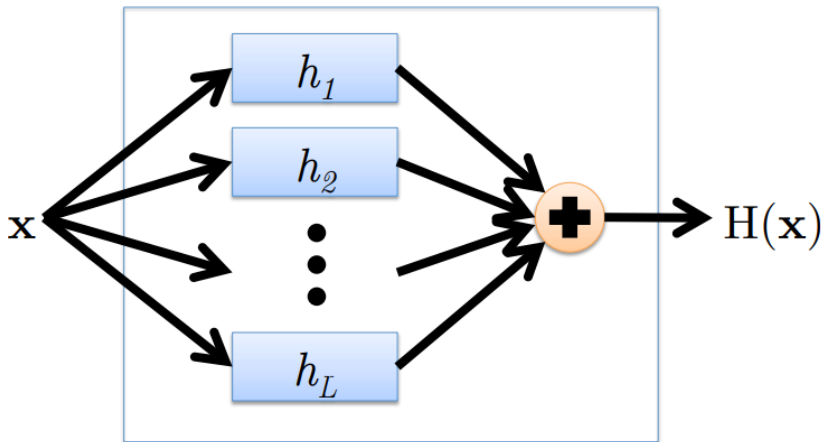
Combining Classifiers: Weighted Averaging

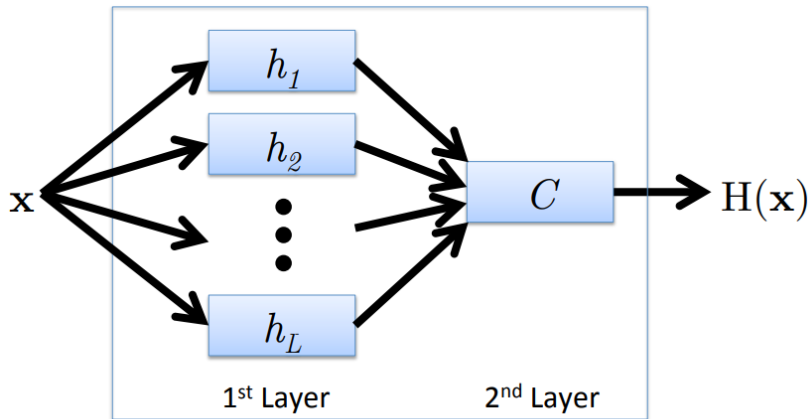- Coefficients of individual members are tuned using a validation set

Combining Classifiers: Gating

- Coefficients of individual members depend on the input
- Train gating function via the validation set

Combining Classifiers: Stacking

- Predictions of the first layer used as input to the second laer
- Train the second layer on the validation set

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

Manipulating the Training Data

- Bootstrap replication
    - Given *n* training examples, construct a new training set by sampling *n* instances with replacement
    - Exclude about 30% of the training instances
- Bagging
    - Create bootstrap replicates of the 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
- Boosting
    - (in just a minute ...)

Manipulating the Features

- Random Forest
  - 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 rees
    - Estimate tree performance on out-of-bootstrap data
 - Average the output of all trees
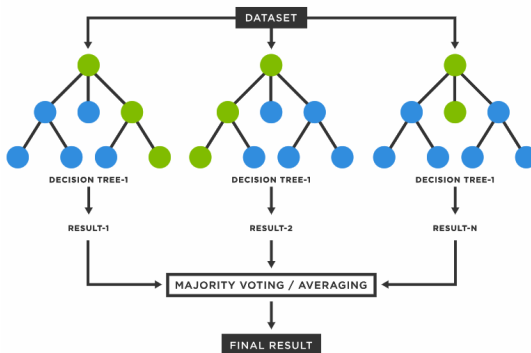
Boosting

AdaBoost

- Developed by Freund & Schapire in 1997
- A meta-learning algorithm with great theoretical and empirical performance
- Turns a base learner (e.g. "weak hypothesis") into a high performance classifier
- Create an ensemble of weak hypotheses by repeatedly emphasizing mispredicted instances

Weak Learners and Classifiers

- (Informal) A weak learner is a learning algorithm that outputs a hypothesis (e.g. a classifier) that performs slightly better than chance.
  - e.g. it predicts the correct label with probability 0.51 in binary label case
- We are interested in weak learners that are computationally efficient
  - Decision tree
  - Even simpler - decision stumps: Decision trees with a single split

- Suppose these are the data
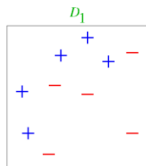
- These weak classifiers, which are decision stumps, consist of the set of horizontal and vertical half-spaces.
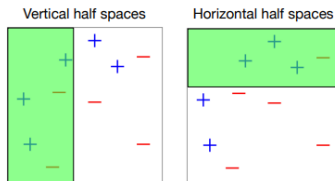
Weak Classifiers

- A single weak classifier is not capable of making the training error small
- But if we can guarantee that it performs slightly better than chance
  - The weighted error of classifier h according to the given weights $\boldsymbol{w} = (w_1, ..., w_N)$ is at most $\frac{1}{2} - \gamma$ for some $\gamma > 0$
- Using it with AdaBoost gives us a universal function approximator
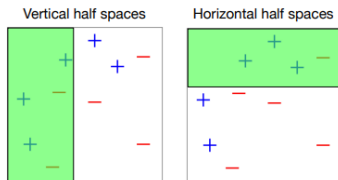
- Now let's see how AdaBoost combines a set of weak classifiers in order to make a better ensemble of classifiers...

- The size of a point or instance represents the instance's weight
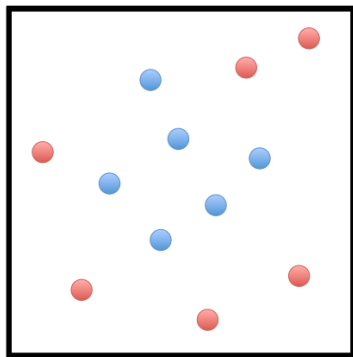
1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3:     Train model h_t on X, y with weights $\mathbf{w}_t$
4:     Compute the weighted training error of h_t
5:     Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$
6:     Update all instance weights:
$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$
7:     Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis
$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x}) \right)$$

1: Initialize a vector of n uniform weights $\mathbf{w}_1$

2: **for** $t = 1, \ldots, T$

3:     Train model h_t on X, y with weights $\mathbf{w}_t$

4:     Compute the weighted training error of h_t

5:     Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$

6:     Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7:     Normalize $\mathbf{w}_{t+1}$ to be a distribution

8: **end for**

9: **Return** the hypothesis

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

$t = 1$

AdaBoost

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3:      Train model h_t on X, y with weights $\mathbf{w}_t$
4:      Compute the weighted training error of h_t
5:      Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$
6:      Update all instance weights:
$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$
7:      Normalize $\mathbf{w}_{t+1}$ to be a distribution
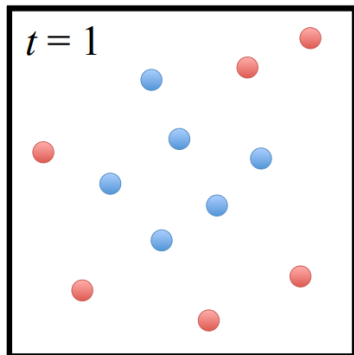8: **end for**
9: **Return** the hypothesis
$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x}) \right)$$

AdaBoost

- β_t measures the importance of h_t
- if $\epsilon_t \leq 0.5$ then $\beta_t \geq 0$ (can trivially gurantee)

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3:    Train model h_t on X, y with weights $\mathbf{w}_t$
4:    Compute the weighted training error of h_t
5:    Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$
6:    Update all instance weights:
$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$
7:    Normalize $\mathbf{w}_{t+1}$ to be a distribution
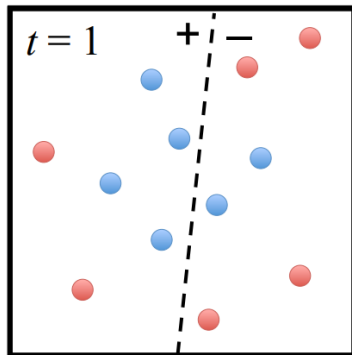8: **end for**
9: **Return** the hypothesis
$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x})\right)$$

$t = 1$

AdaBoost

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

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3:    Train model h_t on X, y with weights $\mathbf{w}_t$
4:    Compute the weighted training error of h_t
5:    Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$
6:    Update all instance weights:
$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$
7:    Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis
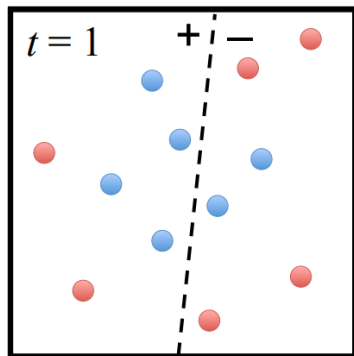$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x}) \right)$$

$t = 1$ $+ \mid -$

AdaBoost

- Note: resized points in the illustration are not necessarily to scale with β_t

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3:     Train model h_t on X, y with weights $\mathbf{w}_t$
4:     Compute the weighted training error of h_t
5:     Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$
6:     Update all instance weights:
        $$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$
7:     Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis
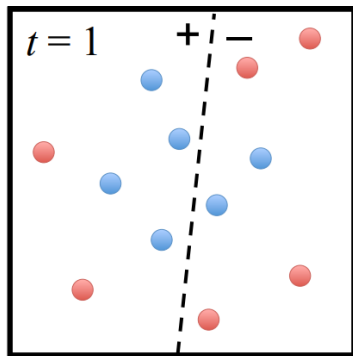$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x}) \right)$$

AdaBoost

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3:     Train model h_t on X, y with weights $\mathbf{w}_t$
4:     Compute the weighted training error of h_t
5:     Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$
6:     Update all instance weights:
$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$
7:     Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis
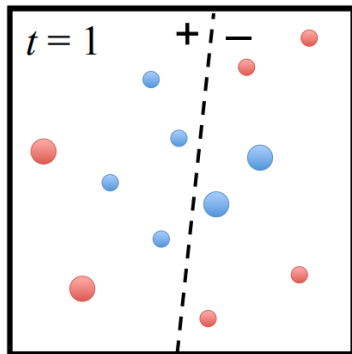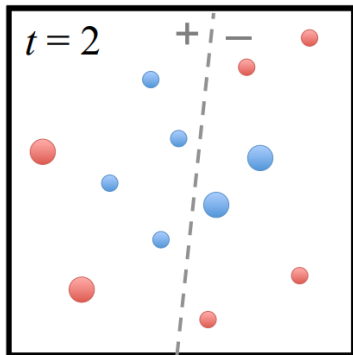$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x}) \right)$$

AdaBoost

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3:      Train model h_t on X, y with weights $\mathbf{w}_t$
4:      Compute the weighted training error of h_t
5:      Choose $\beta_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
6:      Update all instance weights:
        $w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$
7:      Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

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

AdaBoost

- β_t measures the importance of h_t
- if $\epsilon_t \leq 0.5$ then $\beta_t \geq 0$ (can trivially gurantee)

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3:     Train model h_t on X, y with weights $\mathbf{w}_t$
4:     Compute the weighted training error of h_t
5:     Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$
6:     Update all instance weights:
$$w_{t+1,i} = w_{t,i} \exp \left(-\beta_t y_i h_t(\mathbf{x}_i) \right)$$
7:     Normalize $\mathbf{w}_{t+1}$ to be a distribution
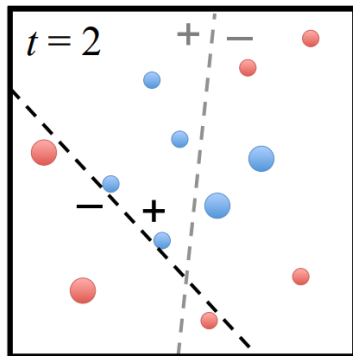8: **end for**
9: **Return** the hypothesis
$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x}) \right)$$

$t = 2$

AdaBoost

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

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3:     Train model h_t on X, y with weights $\mathbf{w}_t$
4:     Compute the weighted training error of h_t
5:     Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$
6:     Update all instance weights:
        $w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$
7:     Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis
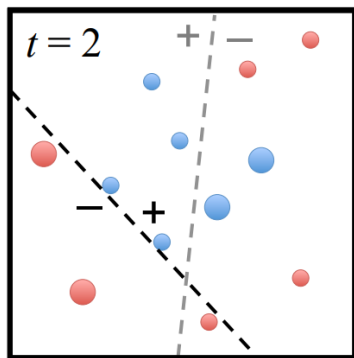$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x}) \right)$$

AdaBoost

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3:      Train model h_t on X, y with weights $\mathbf{w}_t$
4:      Compute the weighted training error of h_t
5:      Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$
6:      Update all instance weights:
$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$
7:      Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis
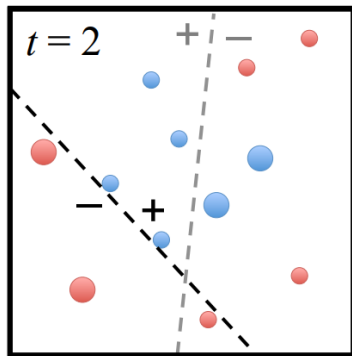$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x}) \right)$$

AdaBoost

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

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3:     Train model h_t on X, y with weights $\mathbf{w}_t$
4:     Compute the weighted training error of h_t
5:     Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$
6:     Update all instance weights:
$$w_{t+1,i} = w_{t,i} \exp \left(-\beta_t y_i h_t(\mathbf{x}_i) \right)$$
7:     Normalize $\mathbf{w}_{t+1}$ to be a distribution
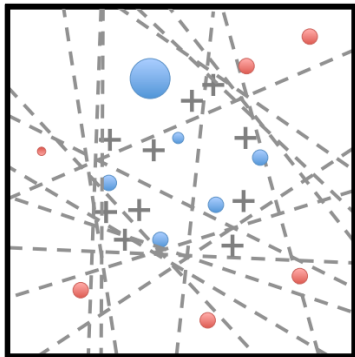8: **end for**
9: **Return** the hypothesis
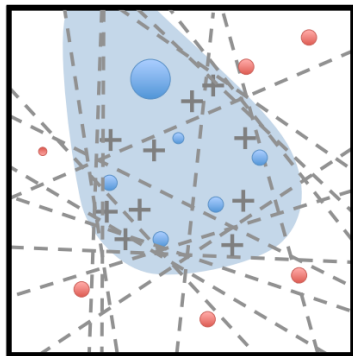$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x}) \right)$$

INPUT: training data $X, y = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$,
the number of iterations T

1: Initialize a vector of n uniform weights $\mathbf{w}_1 = \left[\frac{1}{n}, \ldots, \frac{1}{n}\right]$
2: **for** $t = 1, \ldots, T$
3:     Train model h_t on X, y with instance weights $\mathbf{w}_t$
4:     Compute the weighted training error rate of h_t:

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

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

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right) \quad \forall i = 1, \ldots, n$$

7:     Normalize $\mathbf{w}_{t+1}$ 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

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

$\mathbf{w}_t$ is a vector of weights over the instances at iteration t

All points start with equal weight

46

INPUT: training data $X, y = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$,
the number of iterations T

1: Initialize a vector of n uniform weights $\mathbf{w}_1 = \left[\frac{1}{n}, \ldots, \frac{1}{n}\right]$
2: **for** $t = 1, \ldots, T$
3:    Train model h_t on X, y with instance weights $\mathbf{w}_t$
4:    Compute the weighted training error rate of h_t:

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

We need a way to weight instances differently when learning the model...

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

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right) \quad \forall i = 1, \ldots, n$$

7:    Normalize $\mathbf{w}_{t+1}$ 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

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

Training 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

$$\mathcal{J}_{\text{reg}}(\boldsymbol{\theta}) = - \sum_{i=1}^{n} w_i [y_1 \log h_{\boldsymbol{\theta}}(\boldsymbol{x}_i) + (1 - y_i)\log(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}_i))] + \lambda ||\boldsymbol{\theta}_{[1:d]}||_2^2 \tag{1}$$

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

Bse Learner Requirements

- AdaBoost works with "weak" learners
    - Should not be complex
    - Typically high-bias classifiers
    - Works even when the 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 $X, y = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$,
the number of iterations T

1: Initialize a vector of n uniform weights $\mathbf{w}_1 = \left[\frac{1}{n}, \ldots, \frac{1}{n}\right]$

2: **for** $t = 1, \ldots, T$

3:      Train model h_t on X, y with instance weights $\mathbf{w}_t$

4:      Compute the weighted training error rate of h_t:

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

> Error is the sum the weights of all misclassified instances

5:      Choose $\beta_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$

6:      Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right) \quad \forall i = 1, \ldots, n$$

7:      Normalize $\mathbf{w}_{t+1}$ 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

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

INPUT: training data $X, y = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$,
the number of iterations T

1: Initialize a vector of n uniform weights $\mathbf{w}_1 = \left[\frac{1}{n}, \ldots, \frac{1}{n}\right]$

2: **for** $t = 1, \ldots, T$

3:   Train model h_t on X, y with instance weights $\mathbf{w}_t$

4:   Compute the weighted training error rate of h_t:

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

5:   Choose $\beta_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$

6:   Update all instance weights

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

- 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

7:   Normalize $\mathbf{w}_{t+1}$ to be a dist

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

8: **end for**

9: **Return** the hypothesis

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

INPUT: train[...] the n[...]

1: Initialize a vect[...]
2: **for** $t = 1, \ldots, T$
3:     Train model
4:     Compute the[...]

$$\epsilon_t = \sum_{i : y_i \neq h_t(\mathbf{x}}$$

5:     Choose $\beta_t = \frac{1}{2} \ln\left(\frac{}{\epsilon_t}\right)$

6:     Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right) \quad \forall i = 1, \ldots, n$$

7:     Normalize $\mathbf{w}_{t+1}$ 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

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

This is the same as:

$$w_{t+1,i} = w_{t,i} \times \begin{cases} e^{-\beta_t} & \text{if } h_t(\mathbf{x}_i) = y_i \\ e^{\beta_t} & \text{if } h_t(\mathbf{x}_i) \neq y_i \end{cases}$$

will be ≤ 1

will be ≥ 1

Essentially this emphasizes misclassified instances.

AdaBoost

INPUT: training data $X, y = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$,
the number of iterations T

1: Initialize a vector of n uniform weights $\mathbf{w}_1 = \left[\frac{1}{n}, \ldots, \frac{1}{n}\right]$

2: **for** $t = 1, \ldots, T$

3:     Train model h_t on X, y with instance weights $\mathbf{w}_t$

4:     Compute the weighted training error rate of h_t:
$$\epsilon_t = \sum_{i:y_i \neq h_t(\mathbf{x}_i)} w_{t,i}$$

5:     Choose $\beta_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$

6:     Update all instance weights:
$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right) \quad \forall i = 1, \ldots, n$$

7:     Normalize $\mathbf{w}_{t+1}$ to be a distribution:
$$w_{t+1,i} = \frac{w_{t+1,i}}{\sum_{j=1}^{n} w_{t+1,j}} \quad \forall i = 1, \ldots, n$$

> Make $\mathbf{w}_{t+1}$ sum to 1

8: **end for**

9: **Return** the hypothesis
$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x})\right)$$

AdaBoost

INPUT: training data $X, y = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$,
the number of iterations T

1: Initialize a vector of n uniform weights $\mathbf{w}_1 = \left[\frac{1}{n}, \ldots, \frac{1}{n}\right]$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with instance weights $\mathbf{w}_t$
4: Compute the weighted training error rate of h_t:
$$\epsilon_t = \sum_{i:y_i \neq h_t(\mathbf{x}_i)} w_{t,i}$$
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
6: Update all instance weights:
$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right) \quad \forall i = 1, \ldots$$
7: Normalize $\mathbf{w}_{t+1}$ 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
$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^T \beta_t h_t(\mathbf{x})\right)$$

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 part of the instance space
  - Instances with the highest weight are often outliers

AdaBoost and Overfitting

- The VC (Vapnik-Chervonenkis) theory originally predicted that AdaBoost would always overfit as T grew large
    - Hypothesis keeps growing more complex
- In practice, AdaBoost often did not overfit, contradicting the VC 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

AdaBoost on OCR data with C4.5 as the base learner
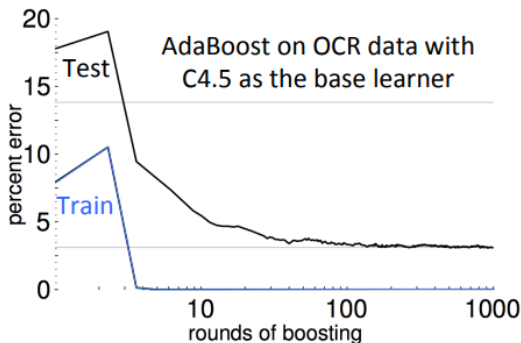
AdaBoost in Practice

- Strengths:
  - Fast and simple to program
  - No parameters to tune (besides T)
  - No assumption 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 Tree

- Boosted decision trees are one of the best "off-the-shelf" classifiers
  - i.e. no parameter tunning
- Limit member hypothesis complexity by limiting tree depth

> **Quote**
>
> "AdaBoost with trees is the best off-the-shelf classifier in the world"
> - Breiman

- Also, see results by Caruana & Niculescu-Mizil, ICML 2006