Welcome!!

Ensemble Learning

Dr. Sugata Ghosal

sugata.ghosal@pilani.bits-pilani.ac.in

BITS Pilani
Pilani Campus

Contents

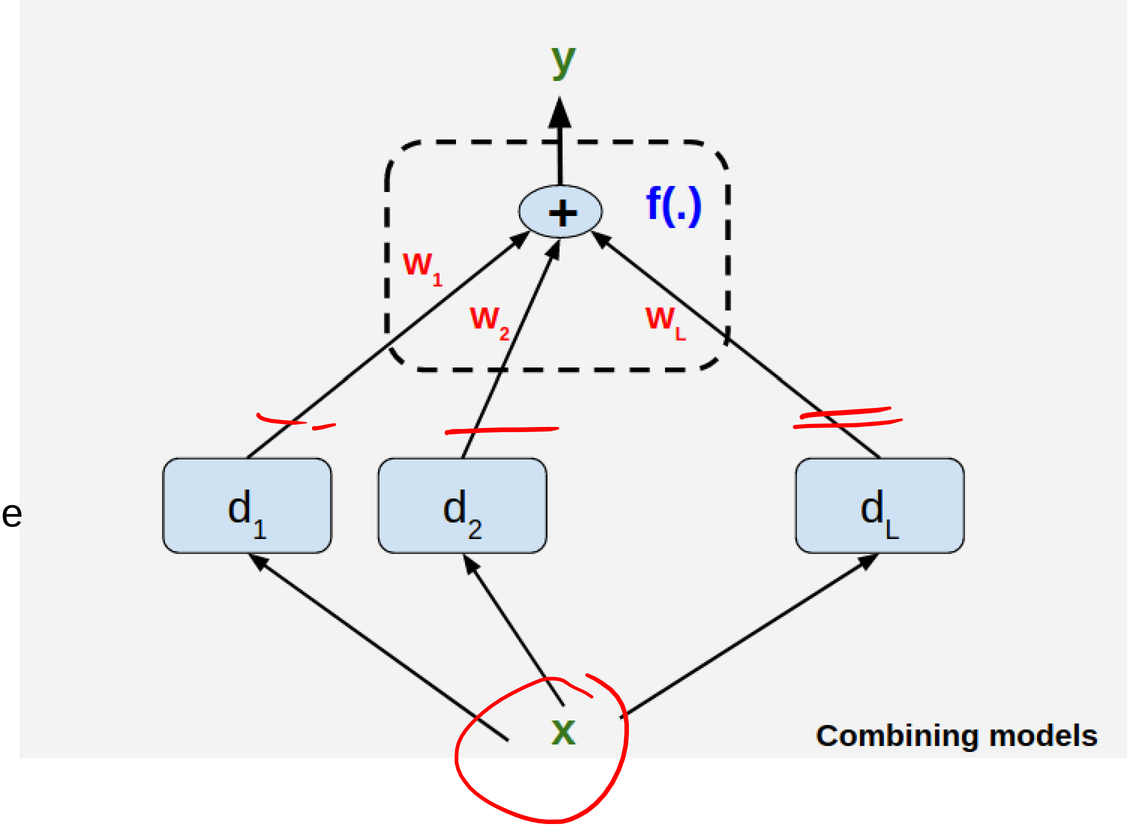
- Combining classifiers
- Bagging
- Boosting
- Random Forest Algorithm
- AdaBoost Algorithm
- Gradient Boosting

Getting Started

- No Free Lunch Theorem: There is no algorithm that is always the most accurate
- Each learning algorithm dictates a certain model that comes with a set of assumptions
  - Each algorithm converges to a different solution and fails under different circumstances
    - The best tuned learners could miss some examples and there could be other learners which works better on (may be only) those !
  - In the absence of a single expert (*a superior model*) , a committee (*combinations of models*) can do better !
    - A committee can work in many ways ...

Committee of Models

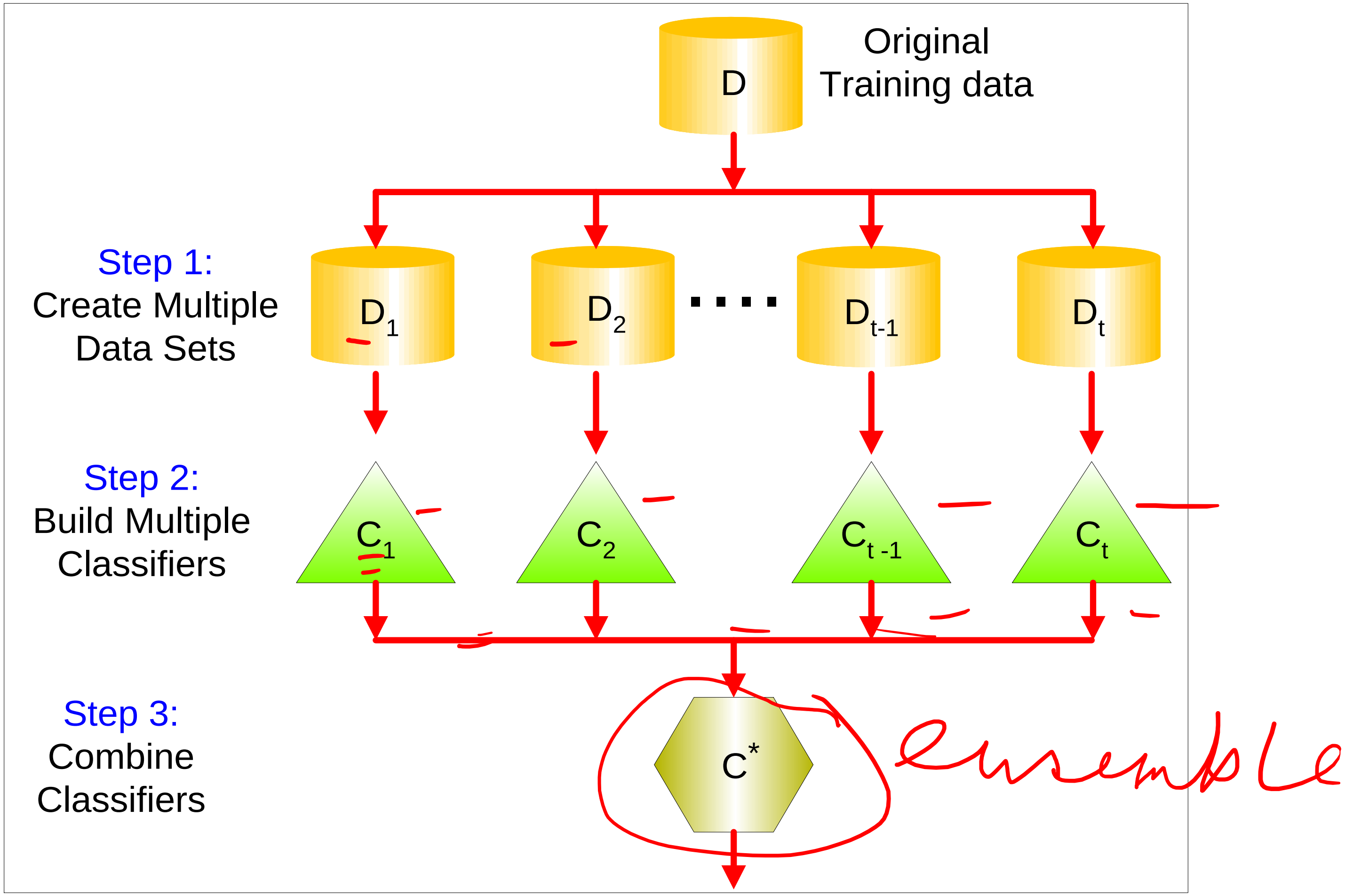
- Committee Members are base learners !
- Major challenges dealing with this committee
  - Expertise of each of the members (Does it help / not?)
  - Combining the results from the members for better performance



Ensemble Methods

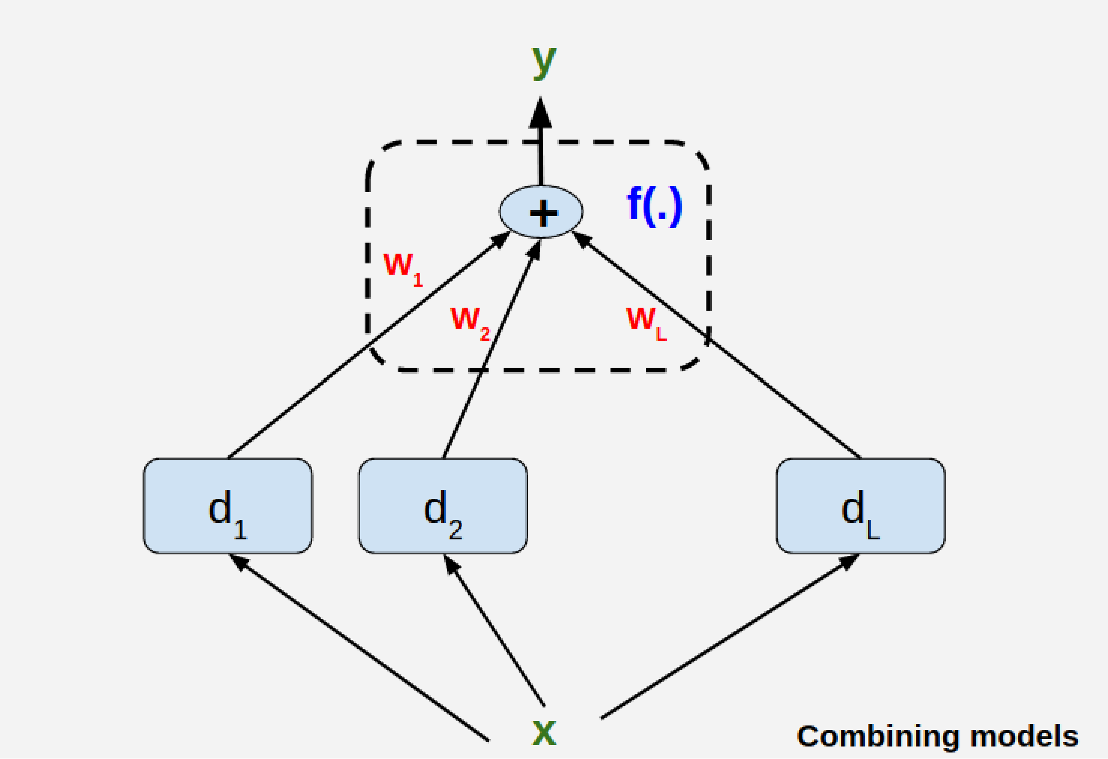
- **Ensemble methods** use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone
- Construct a set of classifiers from the training data
- Predict class label of test records by combining the predictions made by multiple classifiers
- Tend to reduce problems related to over-fitting of the training data.

General Approach



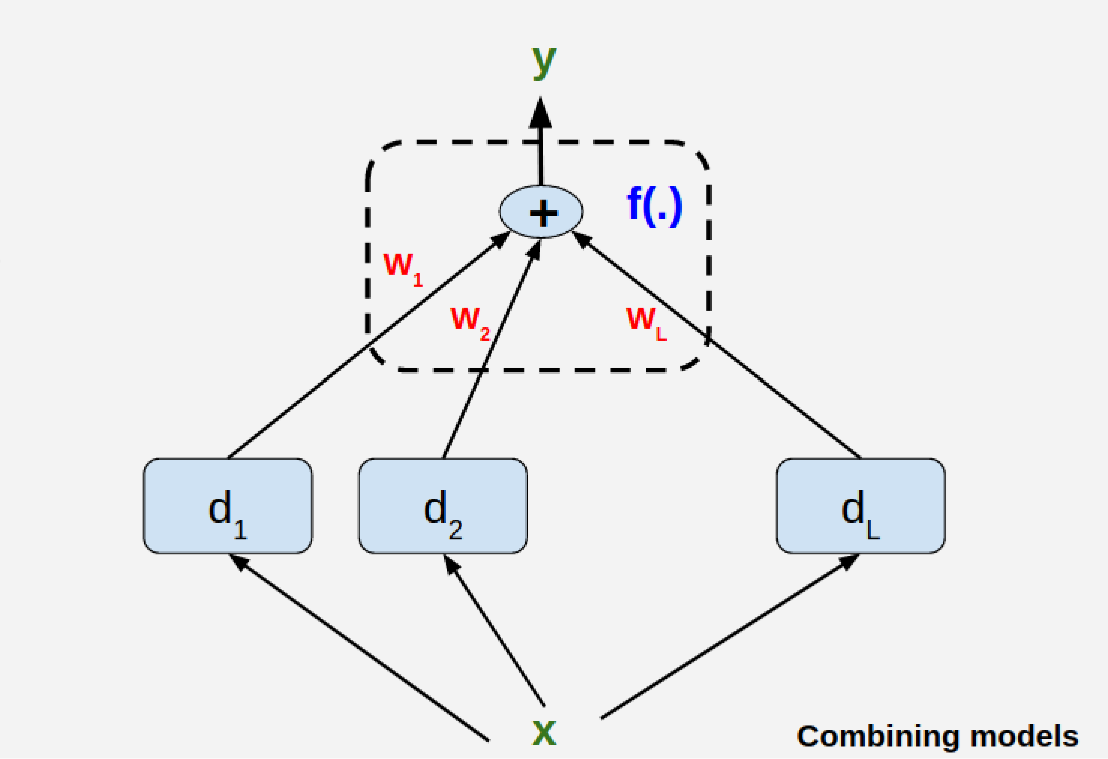
Issue 1 : On the members (Base Learners)

- It does not help if all learners are good/bad at roughly same thing
  - Need Diverse Learners



Issue 1 : On the members (Base Learners)

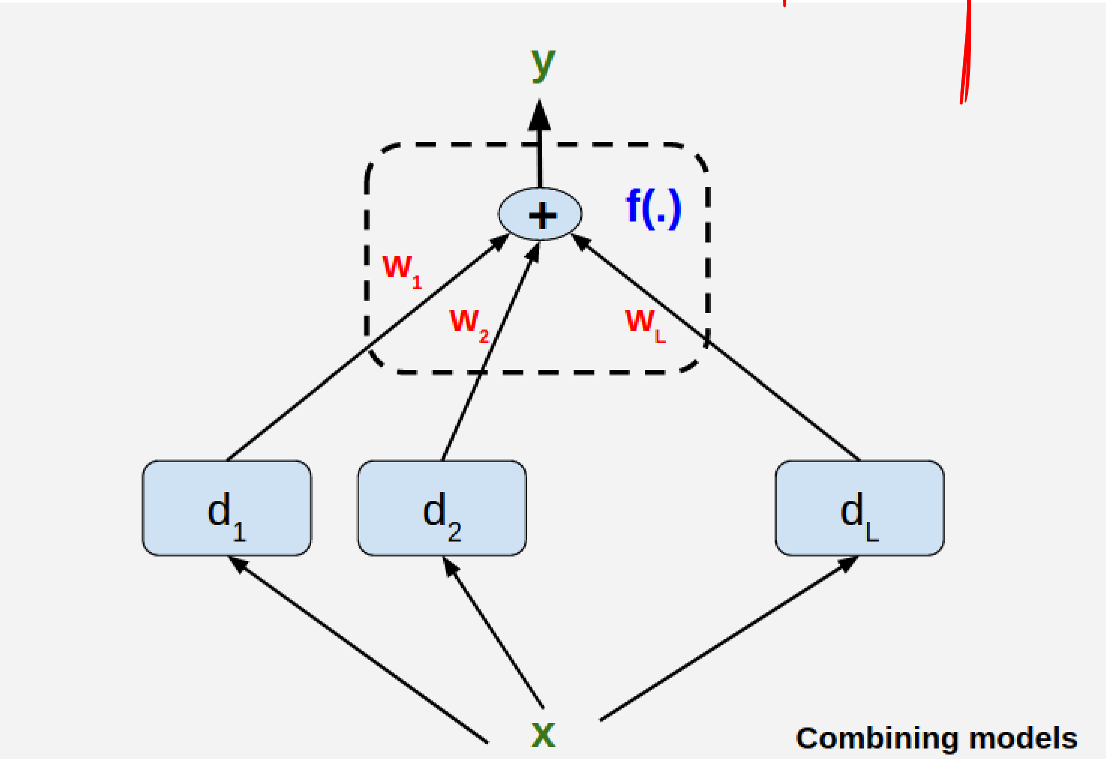
- Use Different Algorithms
  - Different algorithms make different assumptions
- Use Different Hyperparameters, that is ,
  - vary the structure of neural nets



Combining models

Issue 1 : On the members (Base Learners)

- Different input representations
  - Uttered words + video information of speakers clips
  - image + text annotations
- Different training sets
  - Draw different random samples of data
  - Partition data in the input space and have learners specialized in those spaces (mixture of experts)



Combining models

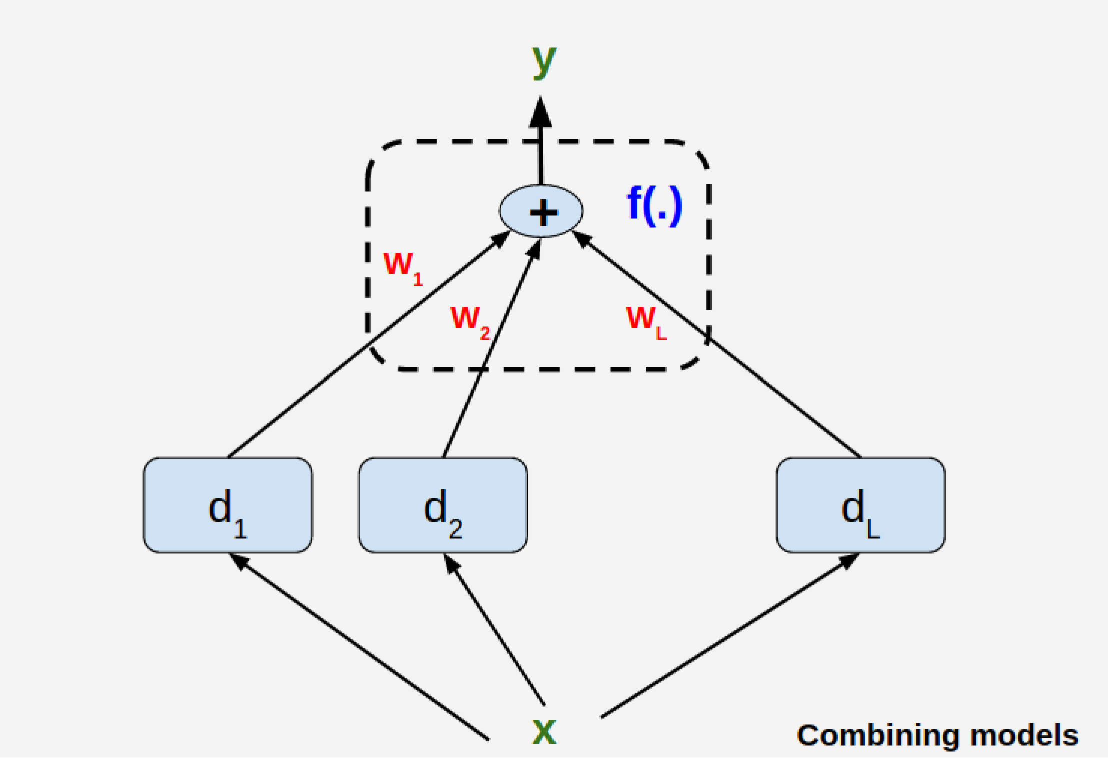
Issue -2 : Combining Results

$$y = f(d_1, d_2, \ldots, d_L | \Phi)$$

A Simple Combination Scheme:

$$y = \sum_{j=1}^{L} w_j d_j$$

$$w_j \geq 0 \text{ and } \sum_{j=1}^{L} w_j = 1$$

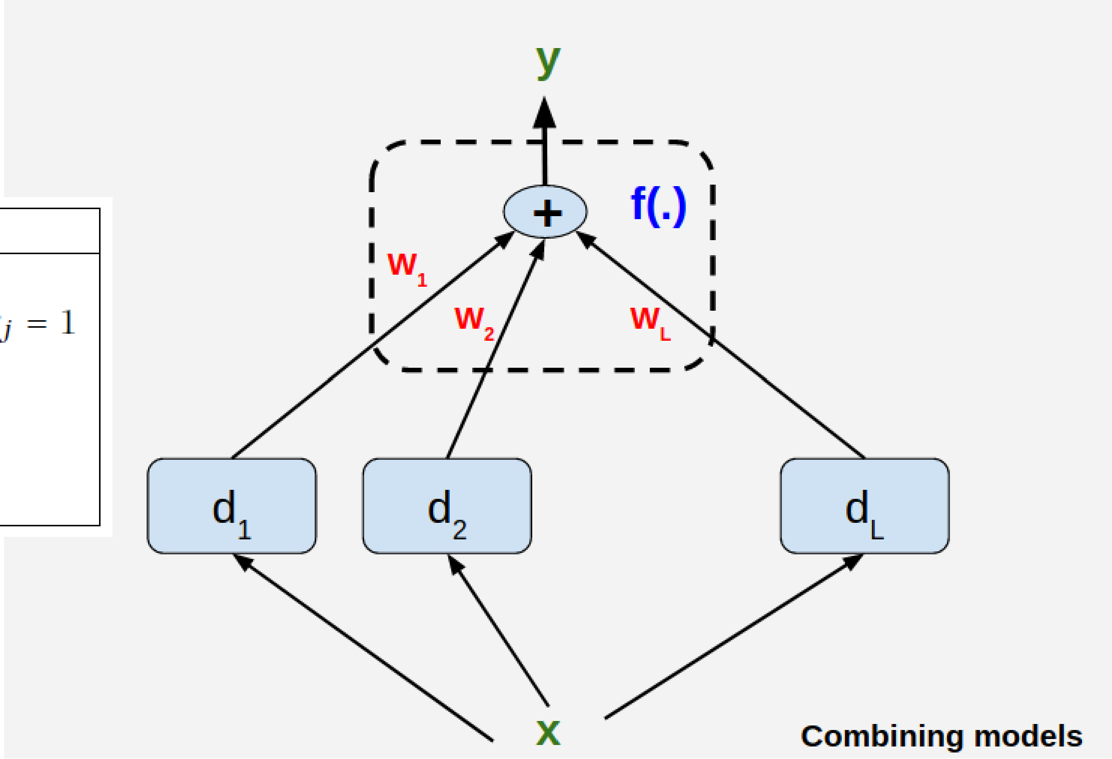


Combining models

Issue -2 : Combining Results

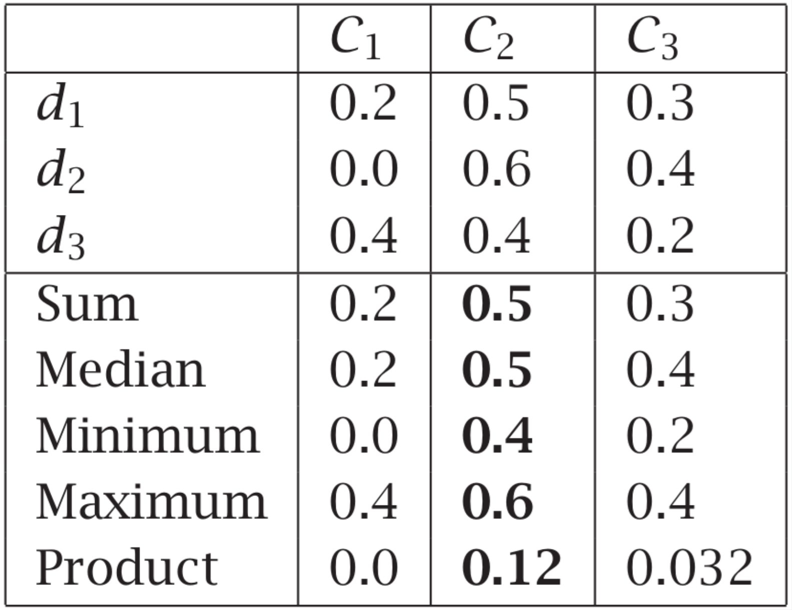
$$y = f(d_1, d_2, \ldots, d_L | \Phi)$$

Rule	Fusion function $f(\cdot)$
Sum	$y_i = \frac{1}{L} \sum_{j=1}^{L} d_{ji}$
Weighted sum	$y_i = \sum_j w_j d_{ji}, w_j \geq 0, \sum_j w_j = 1$
Median	$y_i = \text{median}_j d_{ji}$
Minimum	$y_i = \min_j d_{ji}$
Maximum	$y_i = \max_j d_{ji}$
Product	$y_i = \prod_j d_{ji}$

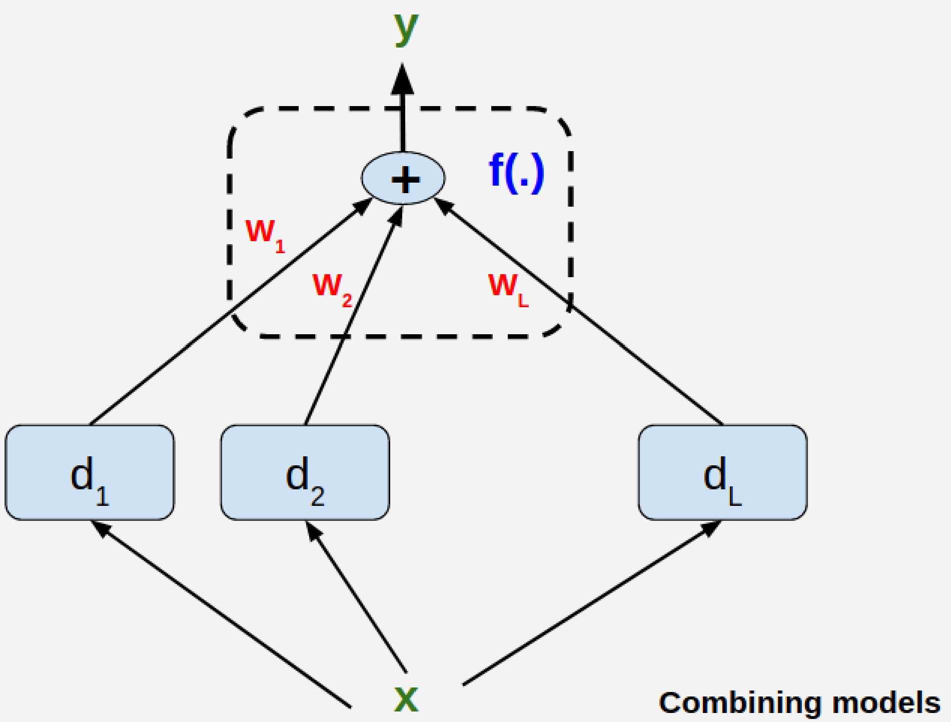


Combining models

Issue -2 : Combining Results



	C_1	C_2	C_3
d_1	0.2	0.5	0.3
d_2	0.0	0.6	0.4
d_3	0.4	0.4	0.2
Sum	0.2	**0.5**	0.3
Median	0.2	**0.5**	0.4
Minimum	0.0	**0.4**	0.2
Maximum	0.4	**0.6**	0.4
Product	0.0	**0.12**	0.032



Combining models

When does Ensemble work?

- Ensemble classifier performs better than the base classifiers when error rate is smaller than 0.5
- Necessary conditions for an ensemble classifier to perform better than a single classifier:
  - Base classifiers should be independent of each other
  - Base classifiers should do better than a classifier that performs random guessing

Why Majority Vote?

- assume *n* independent classifiers with a base error rate ϵ
- here, independent means that the errors are uncorrelated
- assume a binary classification task
- assume the error rate is better than random guessing (i.e., lower than 0.5 for binary classification)

$$\forall \epsilon_i \in \{\epsilon_1, \epsilon_2, \ldots, \epsilon_n\}, \epsilon_i \quad < 0.5$$

Why Majority Vote?

- assume *n* independent classifiers with a base error rate ϵ
- here, independent means that the errors are uncorrelated
- assume a binary classification task
- assume the error rate is better than random guessing (i.e., lower than 0.5 for binary classification)

$k =$

$\boxed{n = 11}$

6 of them makes error

$^nC_k =$

$$\forall \epsilon_i \in \{\epsilon_1, \epsilon_2, \ldots, \epsilon_n\}, \epsilon_i < 0.5$$

$\epsilon_i = \epsilon$

The probability that we make a wrong prediction via the ensemble if *k* classifiers predict the same class label

^nC_k

$$P(k) = \binom{n}{k} \epsilon^k (1-\epsilon)^{\boxed{n-k}}$$

$$k > \lceil n/2 \rceil$$

(Probability mass func. of a binomial distr.)

Why Majority Vote?

The probability that we make a wrong prediction via the ensemble if *k* classifiers predict the same class label

$$P(k) = \binom{n}{k} \epsilon^k (1-\epsilon)^{n-k} \qquad k > \lceil n/2 \rceil$$

$$\sum_{k=6}^{n=11} \binom{n}{k} \epsilon^k (1-\epsilon)^{n-k}$$

Ensemble error:

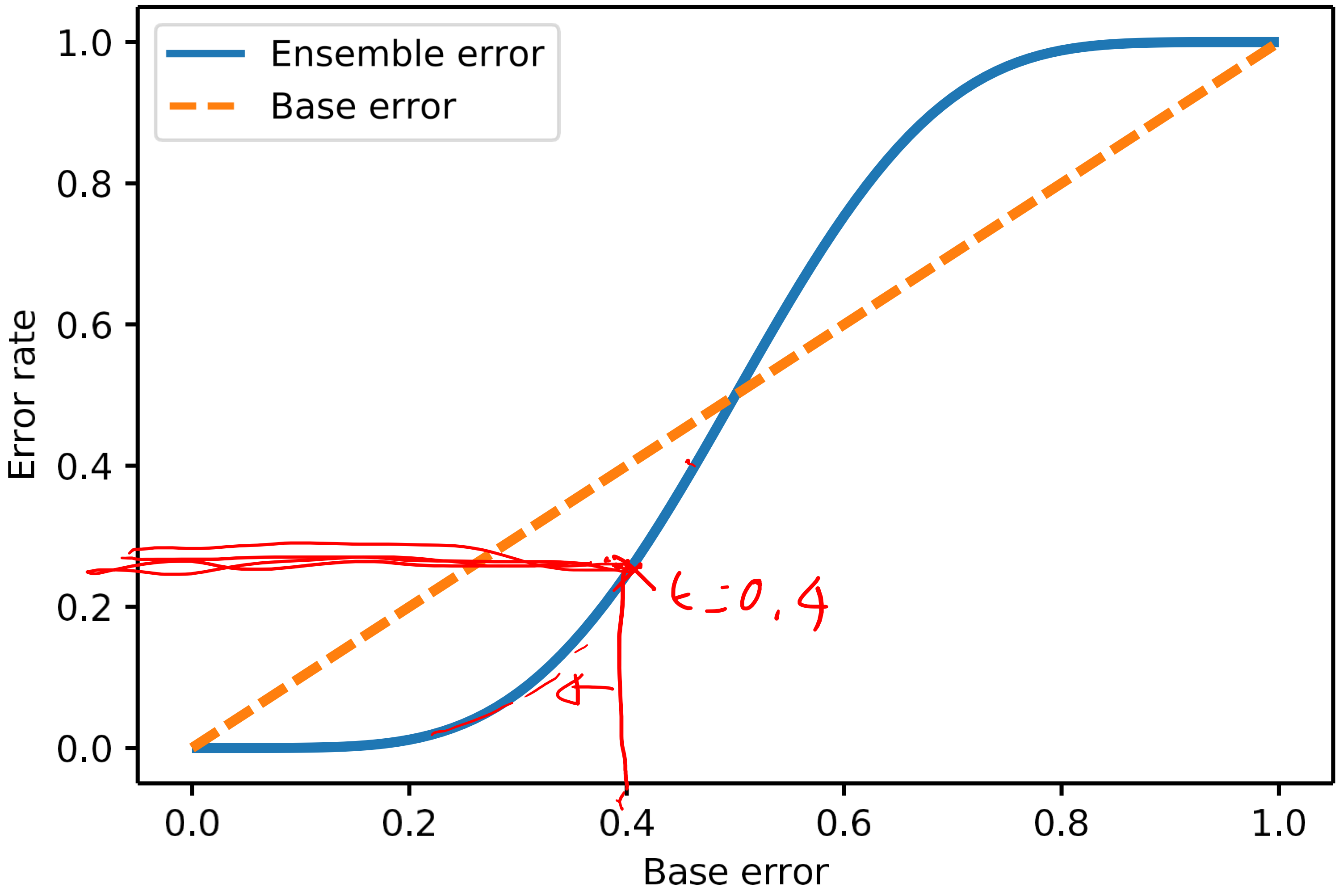
when $n=11$

$$\epsilon_{ens} = \sum_{k}^{n} \binom{n}{k} \epsilon^k (1-\epsilon)^{n-k}$$

$k \geq 6$

$$\epsilon_{ens} = \sum_{k=6}^{11} \binom{11}{6} 0.25^k (1-0.25)^{11-k} = 0.034$$

$$\epsilon_{\text{ens}} = \sum_{k}^{n} \binom{n}{k} \epsilon^{k} (1-\epsilon)^{n-k}$$



Sebastian Raschka

Types of Ensemble Methods

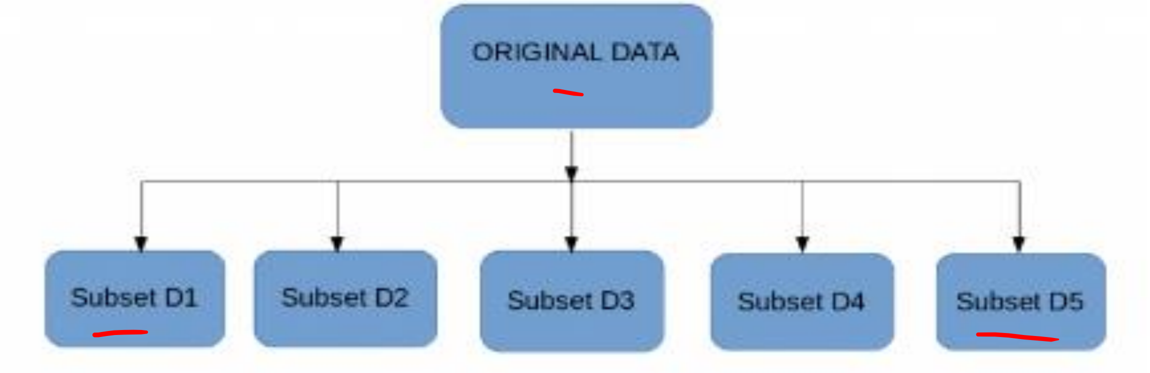
- Manipulate data distribution
  - Example: bagging, boosting
- Manipulate input features
  - Example: random forests

→ (1, 7, 6, 9) → (1, 5, 6, 8)

Bagging (Bootstrap Aggregating)

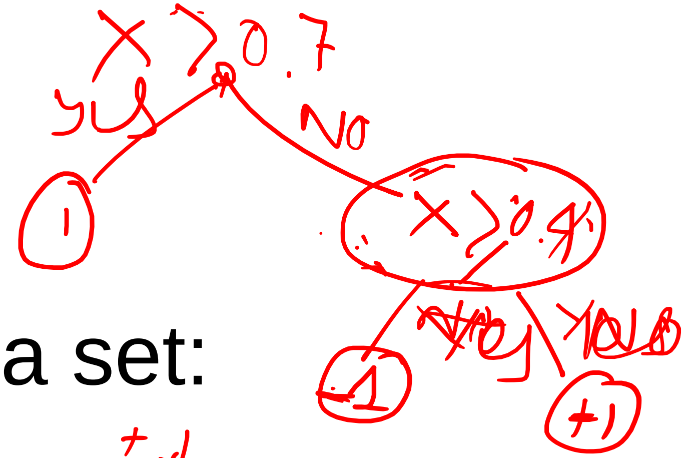
- Technique uses these subsets (bags) to get a fair idea of the distribution (complete set).
- The size of subsets created for bagging may be less than the original set.
- Bootstrapping is a sampling technique in which we create subsets of observations from the original dataset, **with replacement**.
- When you sample with replacement, items are independent. One item does not affect the outcome of the other. You have 1/7 chance of choosing the first item and a 1/7 chance of choosing the second item.
- If the two items are **dependent**, or linked to each other. When you choose the first item, you have a 1/7 probability of picking a item. Assuming you don’t replace the item, you only have six items to pick from. That gives you a 1/6 chance of choosing a second item.

Bagging



- Multiple subsets are created from the original dataset, selecting observations with replacement.
- A base model (weak model) is created on each of these subsets.
- The models run in parallel and are independent of each other.
- The final predictions are determined by combining the predictions from all the models.

Bagging Example



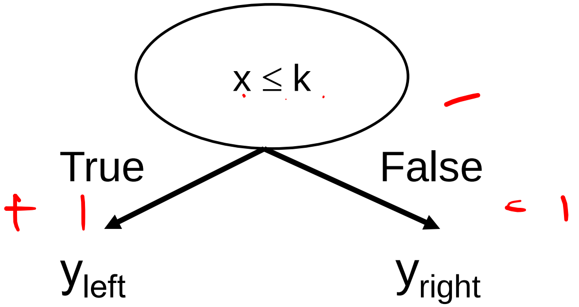
- Consider 1-dimensional data set:

-1 +1

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Classifier is a decision stump
  - – Decision rule: x ≤ k versus x > k
  - – Split point k is chosen based on entropy



Bagging Example

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

x 0.05

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.5	0.9	1	1	1
y	1	1	1	-1	-1	-1	1	1	1	1

X < = 0.0~~1~~5 -> y= -1
X > 0.0~~1~~5 -> y= 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ➔ y = 1
x > 0.3 ➔ y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Example

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ➔ y =-1
x > 0.05 ➔ y = 1

Bagging Example

- Summary of Training sets:

Round	Split Point	Left Class	Right Class
1	0.35	1	-1
2	0.7	-1	1
3	0.35	1	-1
4	0.3	1	-1
5	0.35	1	-1
6	0.75	-1	1
7	0.75	-1	1
8	0.75	-1	1
9	0.75	-1	1
10	0.05	1	1

Bagging Example

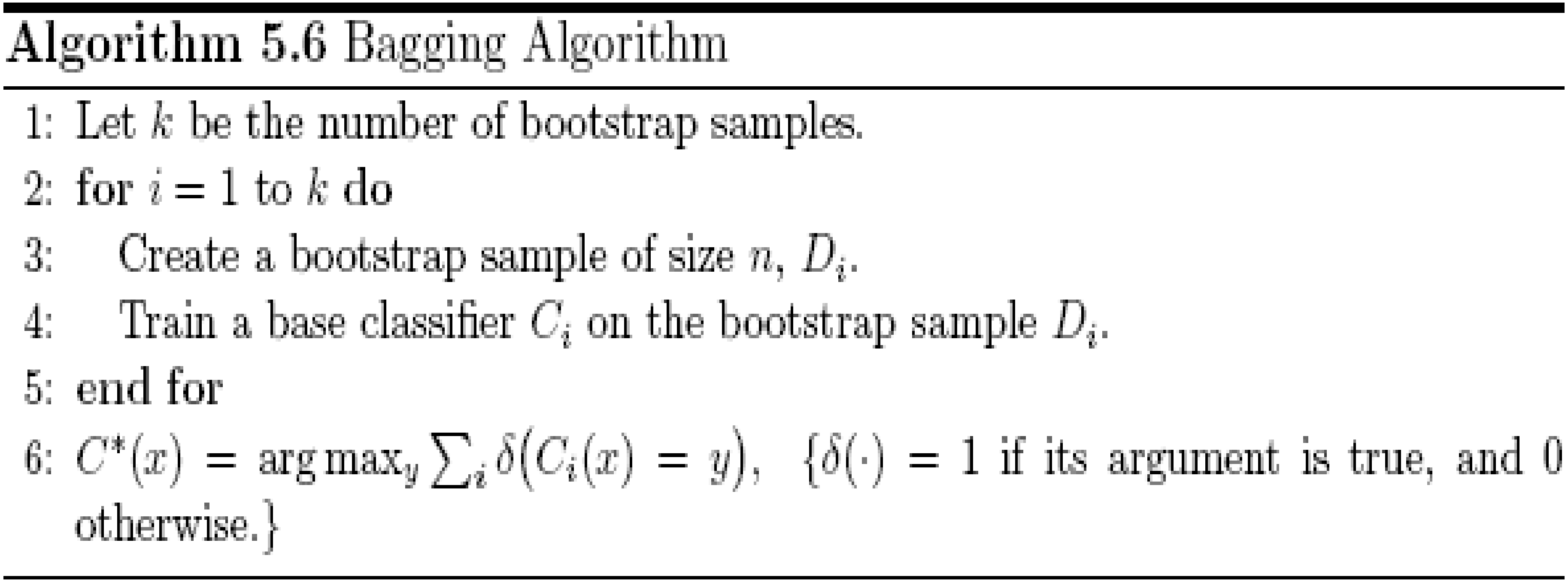
- Assume test set is the same as the original data
- Use majority vote to determine class of ensemble classifier

0.35

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

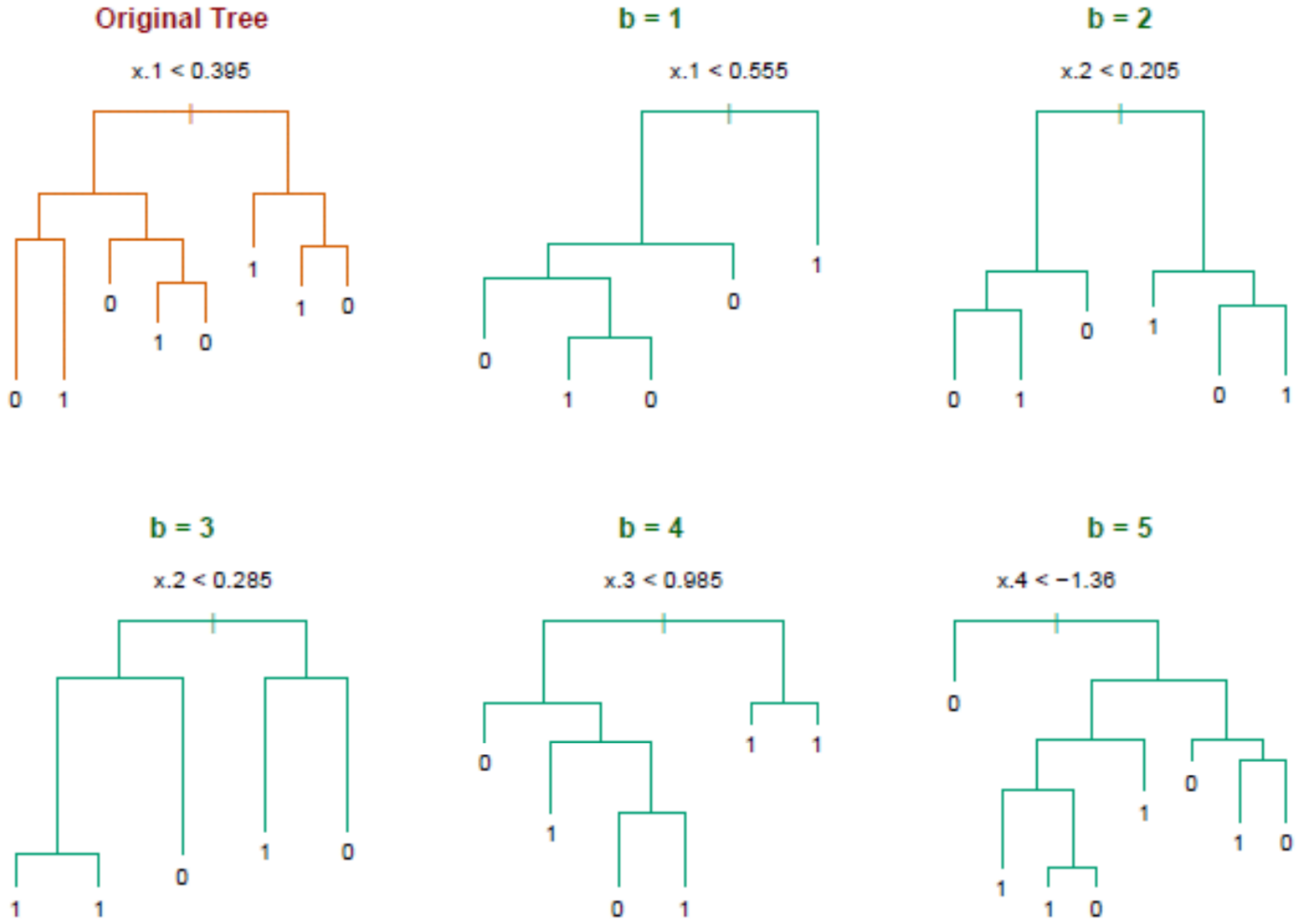
Bagging Algorithm



Algorithm 5.6 Bagging Algorithm

```
Let $k$ be the number of bootstrap samples.
for $i = 1$ to $k$ do
    Create a bootstrap sample of size $n$, $D_i$.
    Train a base classifier $C_i$ on the bootstrap sample $D_i$.
end for
$C^*(x) = \arg\max_y \sum_i \delta\big(C_i(x) = y\big)$, $\{\delta(\cdot) = 1$ if its argument is true, and 0
  otherwise.$\}$
```

Bagging decision trees



Hastie et al.,"The Elements of Statistical Learning: Data Mining, Inference, and Prediction", Springer (2009)

Out-of-Bag Error Estimation

- No cross validation?
- Remember, in bootstrapping we sample with replacement, and therefore **not all observations are used for each bootstrap sample**. On average 1/3 of them are not used!
- We call them out-of-bag samples (OOB)
- We can predict the response for the *i-th* observation using each of the trees in which that observation was OOB and do this for *n* observations
- Calculate overall OOB MSE or classification error

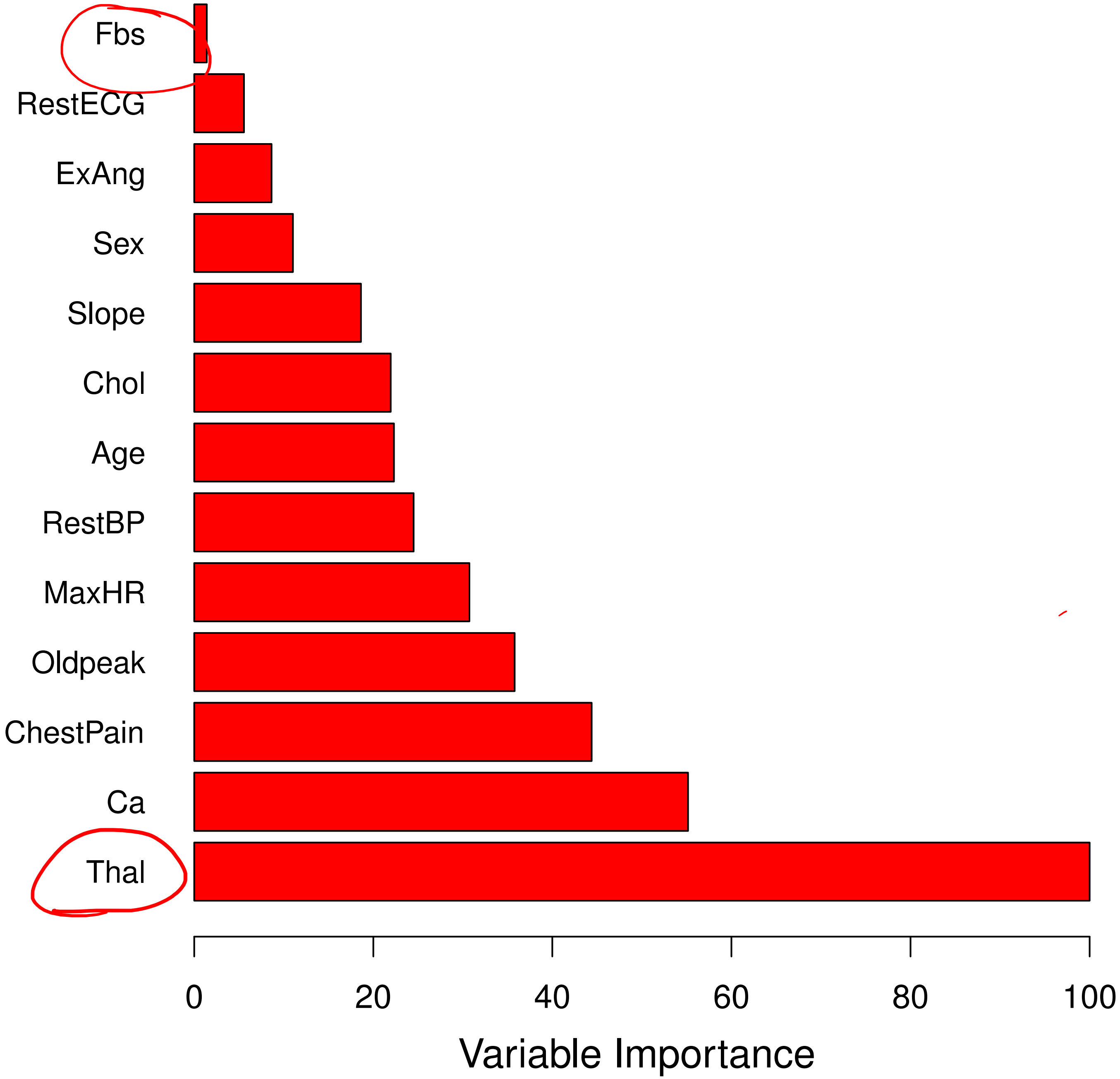
Bagging

- Reduces overfitting (variance)
- Normally uses one type of classifier
- Decision trees are popular
- Easy to parallelize

Variable Importance Measures

- Bagging results in improved accuracy over prediction using a single tree
- Unfortunately, difficult to interpret the resulting model. Bagging improves prediction accuracy at the expense of interpretability.

Calculate the total amount that the RSS or entropy is decreased due to splits over a given predictor, averaged over all B trees.


Fbs
RestECG
ExAng
Sex
Slope
Chol
Age
RestBP
MaxHR
Oldpeak
ChestPain
Ca
Thal
0
20
40
60
80
100
Variable Importance

Bagging - issues

Each tree is identically distributed (i.d.)

➔ the expectation of the average of *B* such trees is the same as the expectation of any one of them

➔the bias of bagged trees is the same as that of the individual trees

i.d. and not i.i.d

Bagging - issues

An average of *B* i.i.d. random variables, each with variance σ², has variance*: σ²/B*

If i.d. (identical but not independent) and pair correlation ρ is present, then the variance is:

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

As *B* increases the second term disappears but the first term remains

Why does bagging generate correlated trees?

Bagging - issues

Suppose that there is one very strong predictor in the data set, along with a number of other moderately strong predictors.

Then all bagged trees will select the strong predictor at the top of the tree and therefore all trees will look similar.

How do we avoid this?

Bagging - issues

Remember we want i.i.d such as the bias to be the same and variance to be less?

Other ideas?

What if we consider only a subset of the predictors at each split?

We will still get correlated trees unless ….

we **randomly** select the subset !

Boosting

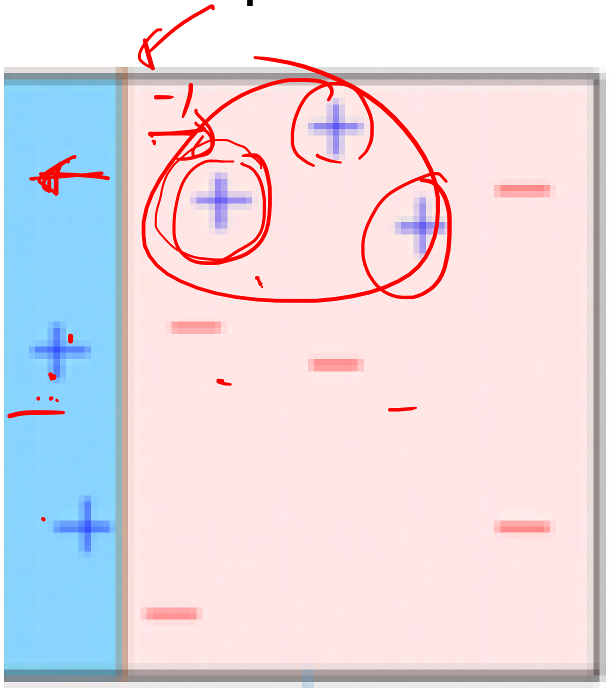
- What if a data point is incorrectly predicted by the first model, and then the next (probably all models), will combining the predictions provide better results? Such situations are taken care of by boosting.
- Boosting is a sequential process, where each subsequent model attempts to correct the errors of the previous model.
- The succeeding models are dependent on the previous model.

Boosting

- An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records
  - Initially, all N records are assigned equal weights
  - Unlike bagging, weights may change at the end of each boosting round

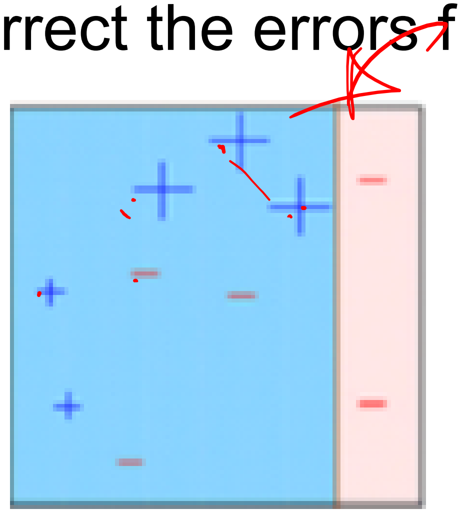
Boosting

- A subset is created from the original dataset.
- Initially, all data points are given equal weights.
- A base model is created on this subset.
- This model is used to make predictions on the whole dataset.



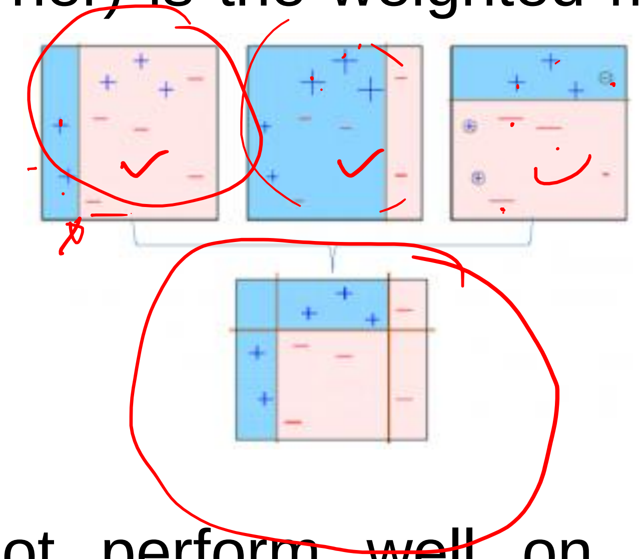
Boosting

- Errors are calculated using the actual values and predicted values.
- The observations which are incorrectly predicted, are given higher weights. (Here, the three misclassified blue-plus points will be given higher weights)
- Another model is created and predictions are made on the dataset. (This model tries to correct the errors from the previous model)



Boosting

- Similarly, multiple models are created, each correcting the errors of the previous model.
- The final model (strong learner) is the weighted mean of all the models (weak learners).



- Individual models would not perform well on the entire dataset, but they work well for some part of the dataset. Thus, each model actually boosts the performance of the ensemble.

Algorithms based on Bagging and Boosting

Bagging algorithms:

- Random forest — + featur randomization

Boosting algorithms:

- AdaBoost
- Gradient Boosting

Random Forest

- Random Forest is ensemble machine learning algorithm that follows the bagging technique.
- The base estimators in random forest are decision trees.
- Random forest randomly selects a set of features which are used to decide the best split at each node of the decision tree.

feature randomization

reducing

Random Forest

data

- Random subsets are created from the original dataset (bootstrapping).
- At each node in the decision tree, only a random set of features are considered to decide the best split.
- A decision tree model is fitted on each of the subsets.
- The final prediction is calculated by averaging the predictions from all decision trees.

Random Forests Algorithm

- For b = 1 to B:
  - (a) Draw a bootstrap sample Z∗ of size *N* from the training data.
  - (b) (b) Grow a random-forest tree to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size $\boldsymbol{n_{min}}$ is reached.

    i. Select ***m*** variables at random from the *p* variables.

    ii. Pick the best variable/split-point among the *m*.

    iii. Split the node into two daughter nodes.
- Output the ensemble of trees.
- To make a prediction at a new point *x* we do:
  - For regression: average the results
  - For classification: majority vote

Random Forests Tuning

The inventors make the following recommendations:

- For classification, the default value for *m* is √*p* and the minimum node size is one.
- For regression, the default value for m is *p/3* and the minimum node size is five.

In practice the best values for these parameters will depend on the problem, and they should be treated as tuning parameters.

Like with Bagging, we can use OOB and therefore RF can be fit in one sequence, with cross-validation being performed along the way. Once the OOB error stabilizes, the training can be terminated.

Advantages of Random Forest

- Algorithm can solve both type of problems i.e. classification and regression
- Power to handle large data set with higher dimensionality.
- It can handle thousands of input variables and identify most significant variables so it is considered as one of the dimensionality reduction methods.
- Model outputs **Importance of variable,** which can be a very handy feature (on some random data set).

RF: Variable Importance Measures

Record the prediction accuracy on the oob samples for each tree

Randomly permute the data for column *j* in the oob samples the record the accuracy again.

The decrease in accuracy as a result of this permuting is averaged over all trees, and is used as a measure of the importance of variable j in the random forest.

Disadvantages of Random Forest

- May over-fit data sets that are particularly noisy.
- Random Forest can feel like a black box approach for statistical modelers – you have very little control on what the model does. You can at best – try different parameters and random seeds!

Random Forests Issues

When the number of variables is large, but the fraction of relevant variables is small, random forests are likely to perform poorly when *m* is small

Why?

Because:

At each split the chance can be small that the relevant variables will be selected

For example, with 3 relevant and 100 not so relevant variables the probability of any of the relevant variables being selected at any split is ~0.25

Can RF overfit?

Random forests “cannot overfit” the data wrt to number of trees.

Why?

The number of trees, *B* does not mean increase in the flexibility of the model

AdaBoost

- Adaptive boosting or AdaBoost is one of the simplest boosting algorithms. Usually, decision trees are used for modelling. Multiple sequential models are created, each correcting the errors from the last model.
- AdaBoost assigns weights to the observations which are incorrectly predicted and the subsequent model works to predict these values correctly.

AdaBoost Algorithm

- Initially, all observations (n) in the dataset are given equal weights (1/n).
- A model is built on a subset of data.
- Using this model, predictions are made on the whole dataset.
- Errors are calculated by comparing the predictions and actual values.
- While creating the next model, higher weights are given to the data points which were predicted incorrectly.

Adaboost Algorithm

- Weights can be determined using the error value. For instance, higher the error more is the weight assigned to the observation.
- This process is repeated until the error function does not change, or the maximum limit of the number of estimators is reached.

AdaBoost

$$\begin{tikzcd} D \arrow[r] & \boxed{C_1} \arrow[r] & \boxed{W_j\text{'s}} \arrow[r] & \boxed{C_2} \end{tikzcd}$$

- Base classifiers C_i: C_1, C_2, …, C_T

W_j

- Error rate:
  - N input samples

$$\boxed{\varepsilon_i} = \frac{1}{N}\sum_{j=1}^{N} \boxed{w_j}\,\delta\left(C_i(x_j) \neq y_j\right)$$

- Importance of a classifier:

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

https://en.wikipedia.org/wiki/AdaBoost#Choosing_αt

AdaBoost: Weight Update

$w_i^{(j+1)} \downarrow$ if correctly $e^{-\alpha_j}$ $e^{-1.2}$ by the previous

Weight Update:

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j}\begin{cases}\exp^{-\alpha_j} & \text{if } C_j(x_i) = y_i\\ \exp^{\alpha_j} & \text{if } C_j(x_i) \neq y_i\end{cases}$$ <- Eqn:5.88

i th sample

where Z_j is the normalizat ion factor

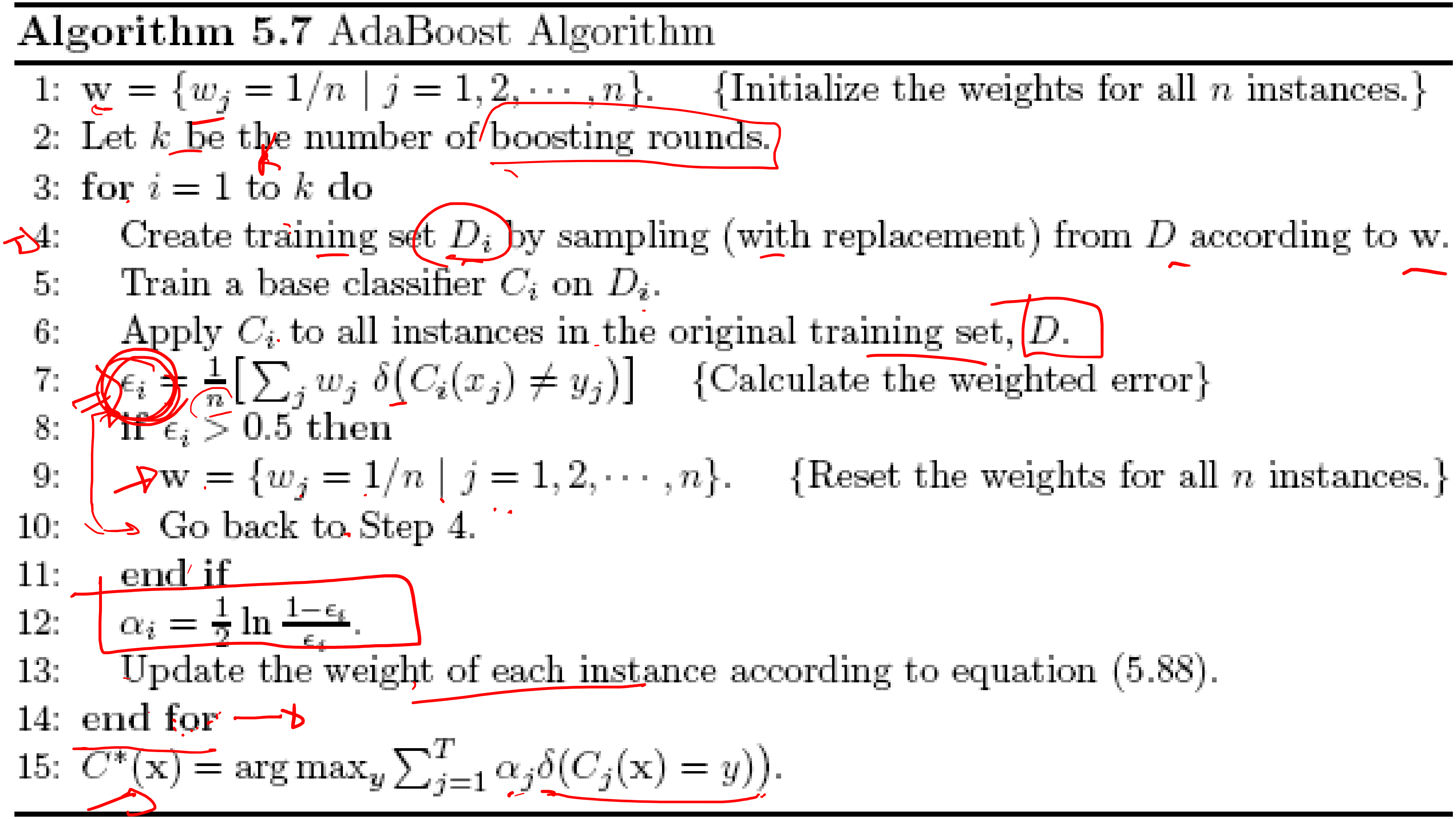
$\sum_i W_i^{(j+1)} = 1$

$Z_j = \sum W_i^{(j+1)}$

$$C^*(x) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta\left(C_j(x) = y\right)$$

- Reduce weight if correctly classified else increase
- If any intermediate rounds produce error rate higher than 50%, the weights are reverted back to 1/n and the resampling procedure is repeated

AdaBoost Algorithm

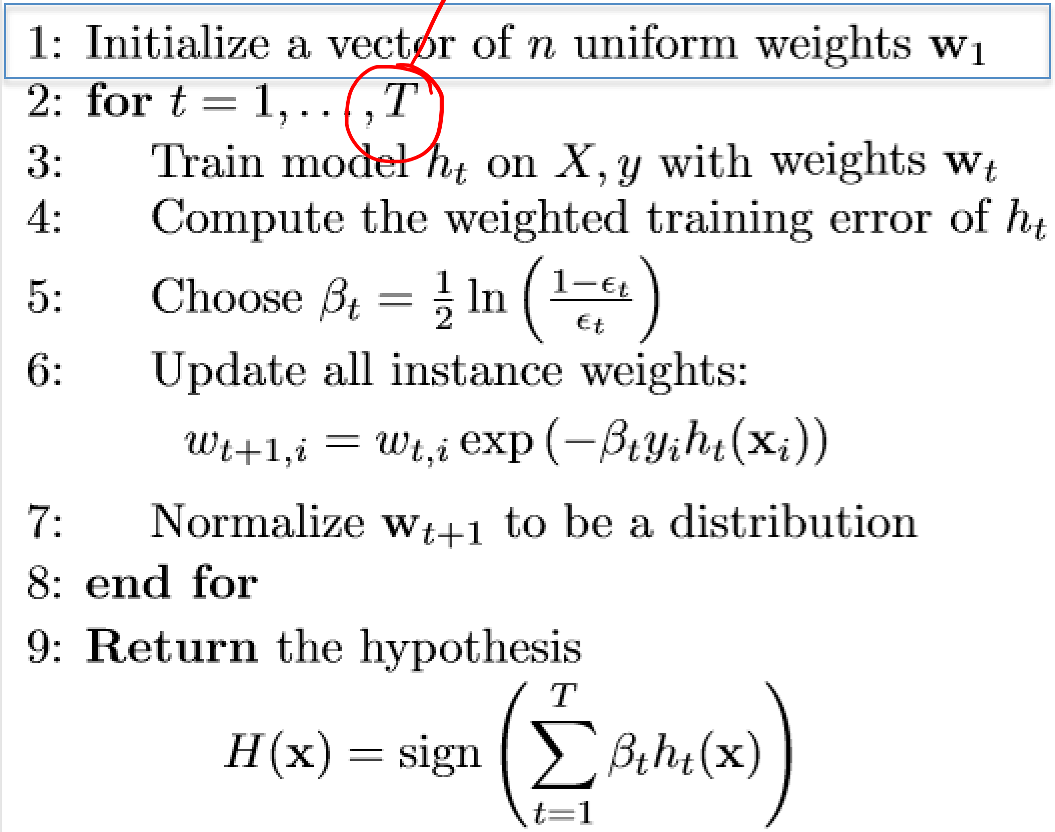


Algorithm 5.7 AdaBoost Algorithm

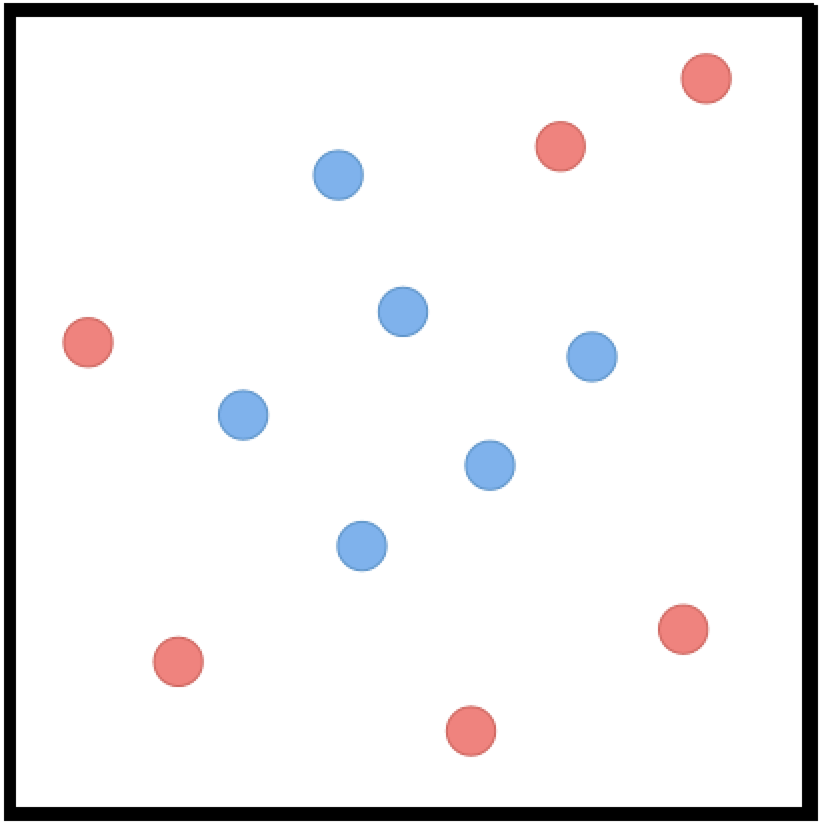
```
1:  $\mathbf{w} = \{w_j = 1/n \mid j = 1, 2, \cdots, n\}$.   {Initialize the weights for all $n$ instances.}
2:  Let $k$ be the number of boosting rounds.
3:  for $i = 1$ to $k$ do
4:      Create training set $D_i$ by sampling (with replacement) from $D$ according to $\mathbf{w}$.
5:      Train a base classifier $C_i$ on $D_i$.
6:      Apply $C_i$ to all instances in the original training set, $D$.
7:      $\epsilon_i = \frac{1}{n}\left[\sum_j w_j \, \delta(C_i(x_j) \neq y_j)\right]$   {Calculate the weighted error}
8:      if $\epsilon_i > 0.5$ then
9:          $\mathbf{w} = \{w_j = 1/n \mid j = 1, 2, \cdots, n\}$.   {Reset the weights for all $n$ instances.}
10:         Go back to Step 4.
11:     end if
12:     $\alpha_i = \frac{1}{2} \ln \frac{1-\epsilon_i}{\epsilon_i}$.
13:     Update the weight of each instance according to equation (5.88).
14: end for
15: $C^*(\mathbf{x}) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta(C_j(\mathbf{x}) = y)$.
```

AdaBoost Algorithm

max # of_classifiers



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

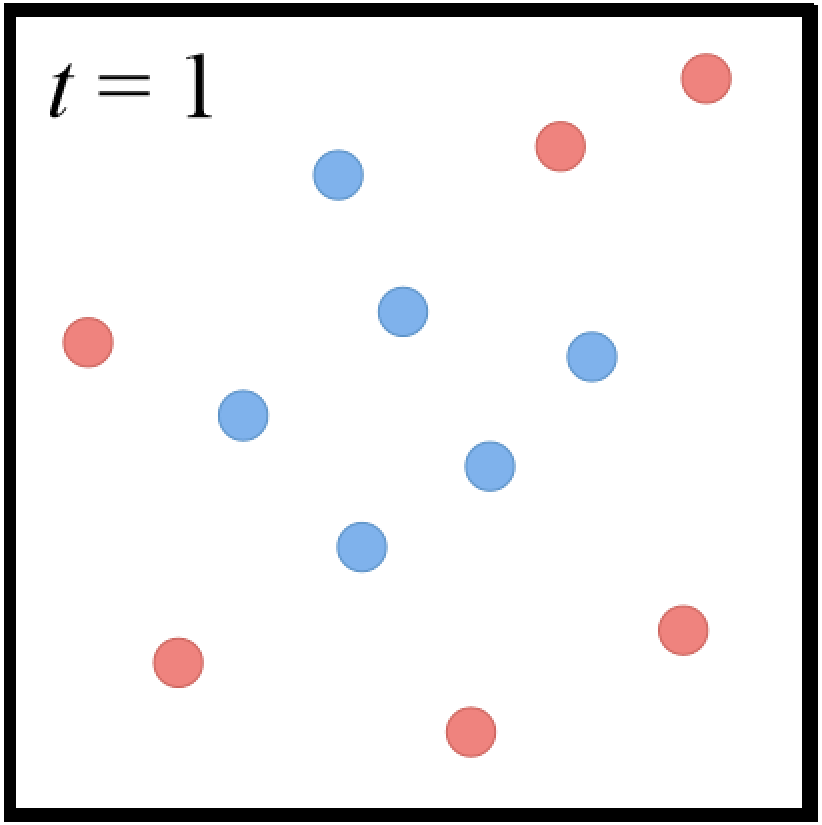


- Size of point represents the instance’s weight

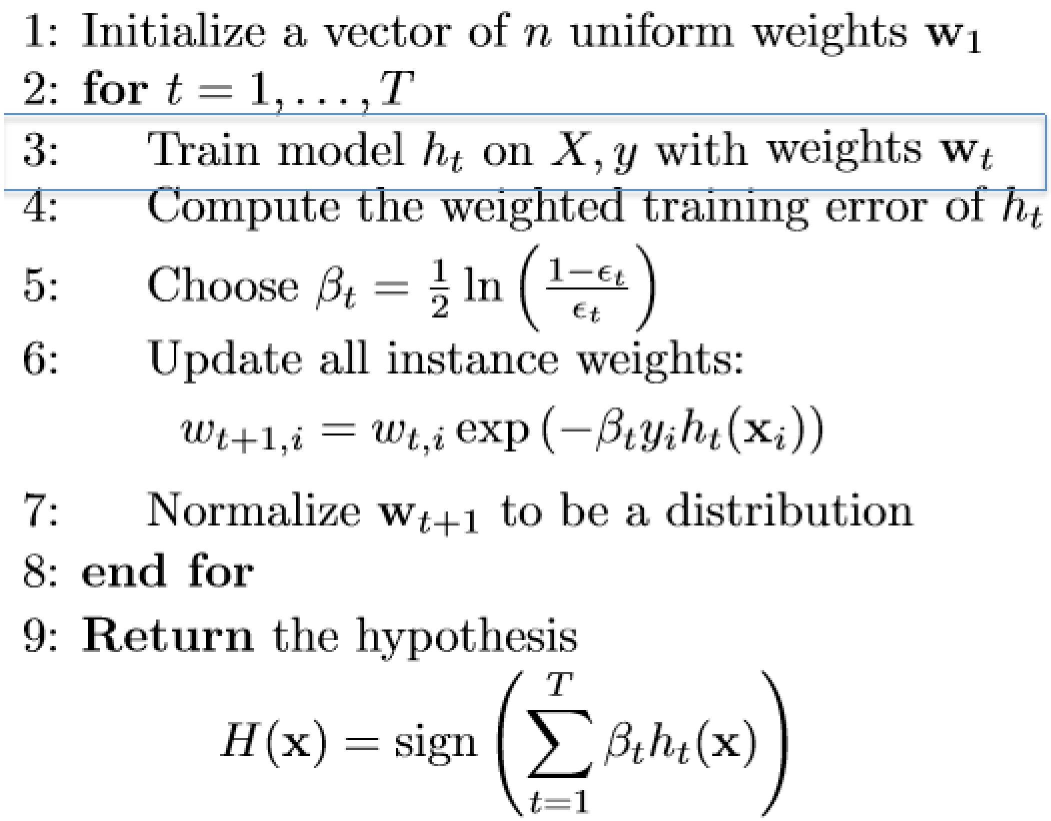
α *in earlier slide same as* $\boldsymbol{\beta}$ = *weight of class*

AdaBoost Algorithm

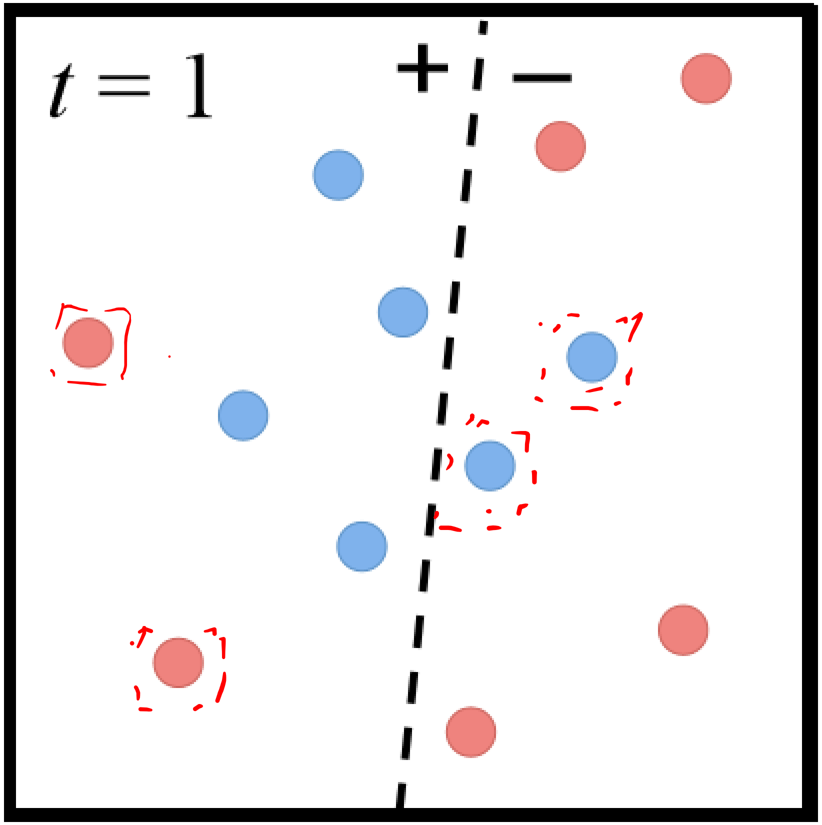
```
1: Initialize a vector of $n$ uniform weights $\mathbf{w}_1$
2: for $t = 1, \ldots, T$
3:     Train model $h_t$ on $X, y$ with weights $\mathbf{w}_t$
4:     Compute the weighted training error of $h_t$
5:     Choose $\beta_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
6:     Update 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 Algorithm

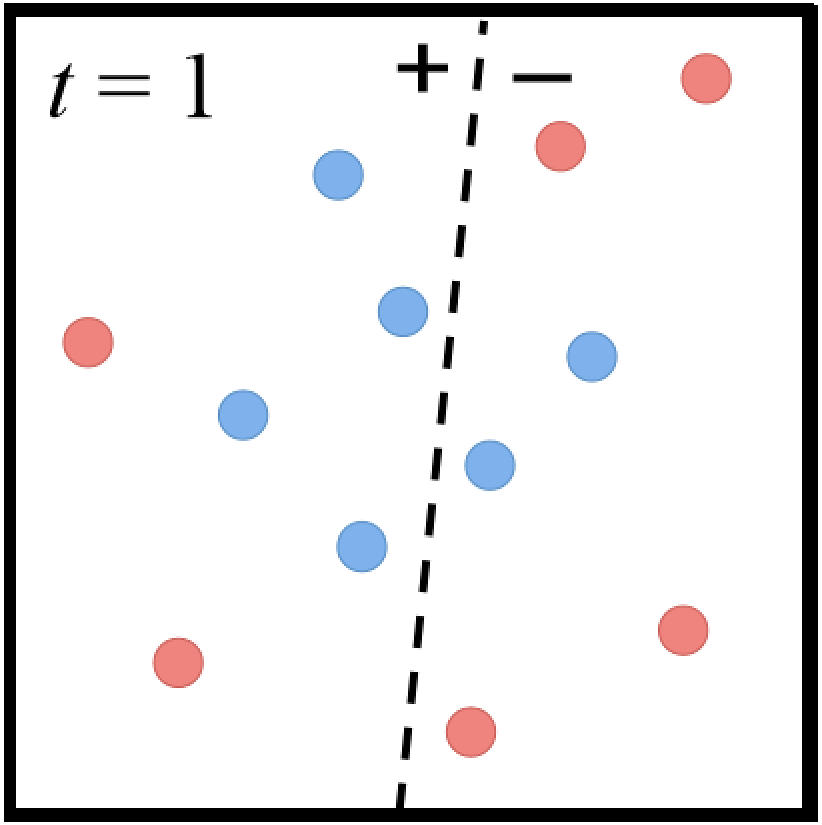


```
Initialize a vector of $n$ uniform weights $\mathbf{w}_1$
for $t = 1, \ldots, T$
    Train model $h_t$ on $X, y$ with weights $\mathbf{w}_t$
    Compute the weighted training error of $h_t$
    Choose $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$
    Update all instance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



AdaBoost Algorithm

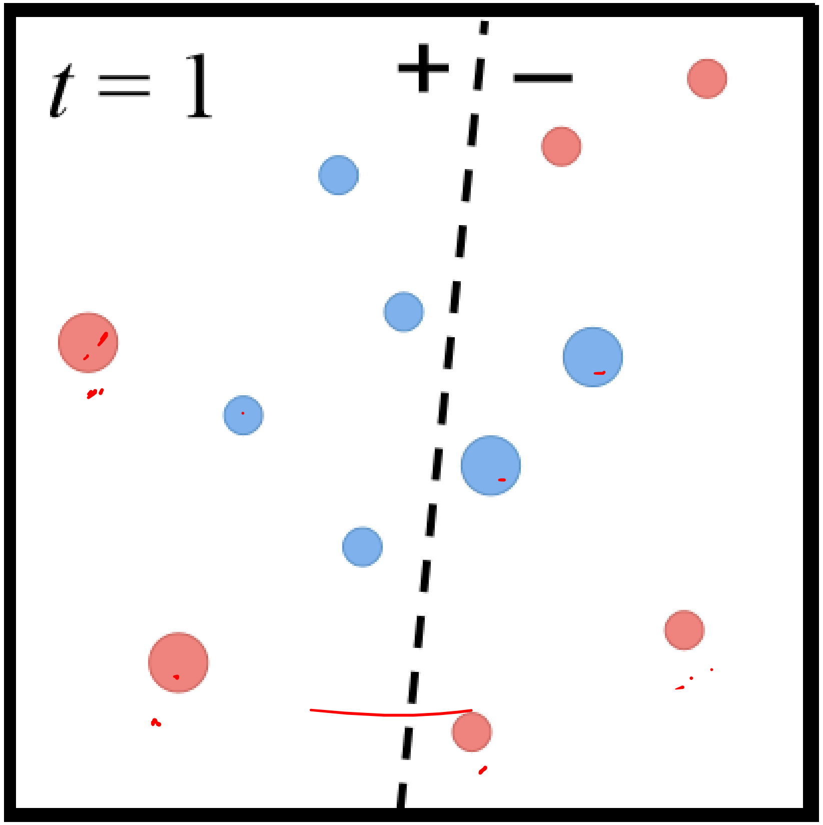
```
Initialize a vector of $n$ uniform weights $\mathbf{w}_1$
for $t = 1, \ldots, T$
    Train model $h_t$ on $X, y$ with weights $\mathbf{w}_t$
    Compute the weighted training error of $h_t$
    Choose $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$
    Update all instance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



- 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 Algorithm

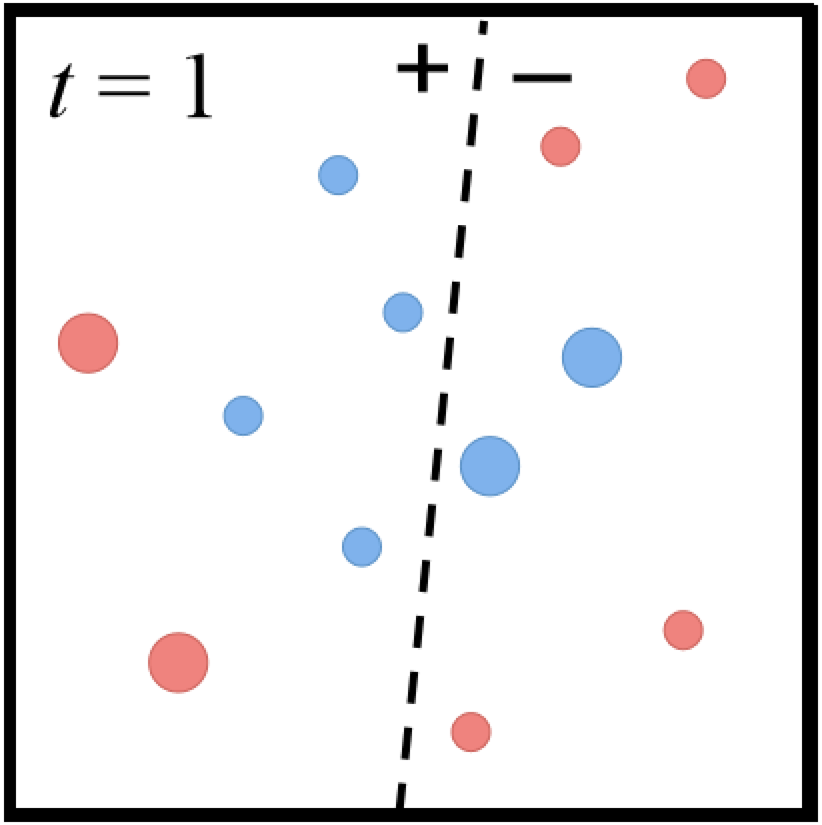
```
Initialize a vector of $n$ uniform weights $\mathbf{w}_1$
for $t = 1, \ldots, T$
    Train model $h_t$ on $X, y$ with weights $\mathbf{w}_t$
    Compute the weighted training error of $h_t$
    Choose $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$
    Update all instance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



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

AdaBoost Algorithm

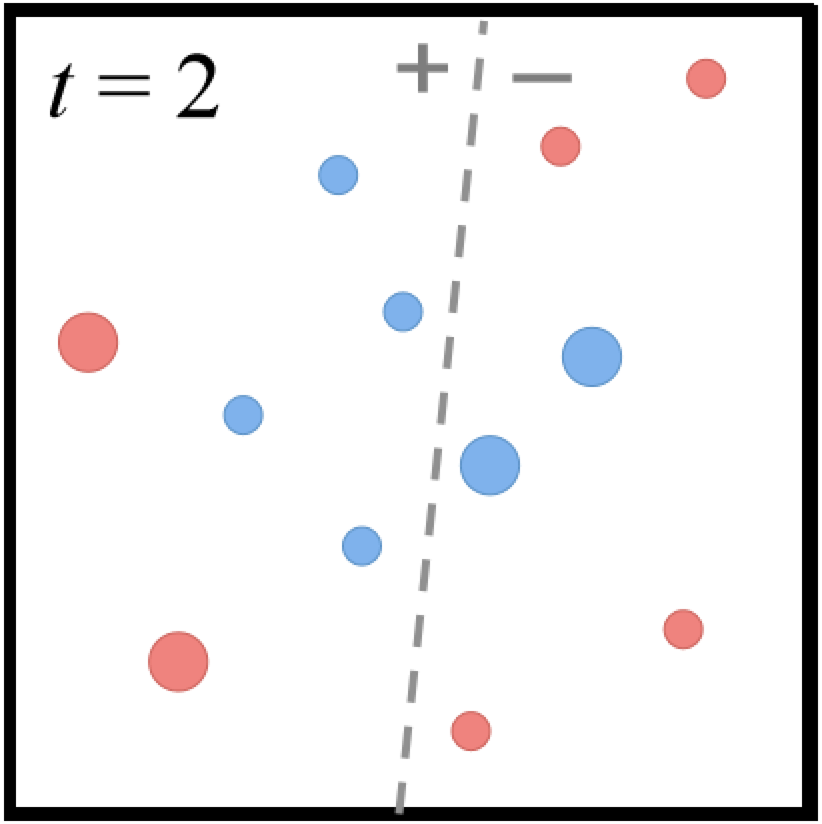
```
1: Initialize a vector of $n$ uniform weights $\mathbf{w}_1$
2: for $t = 1, \ldots, T$
3:     Train model $h_t$ on $X, y$ with weights $\mathbf{w}_t$
4:     Compute the weighted training error of $h_t$
5:     Choose $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$
6:     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
         $H(\mathbf{x}) = \text{sign}\left( \sum_{t=1}^{T} \beta_t h_t(\mathbf{x}) \right)$
```



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

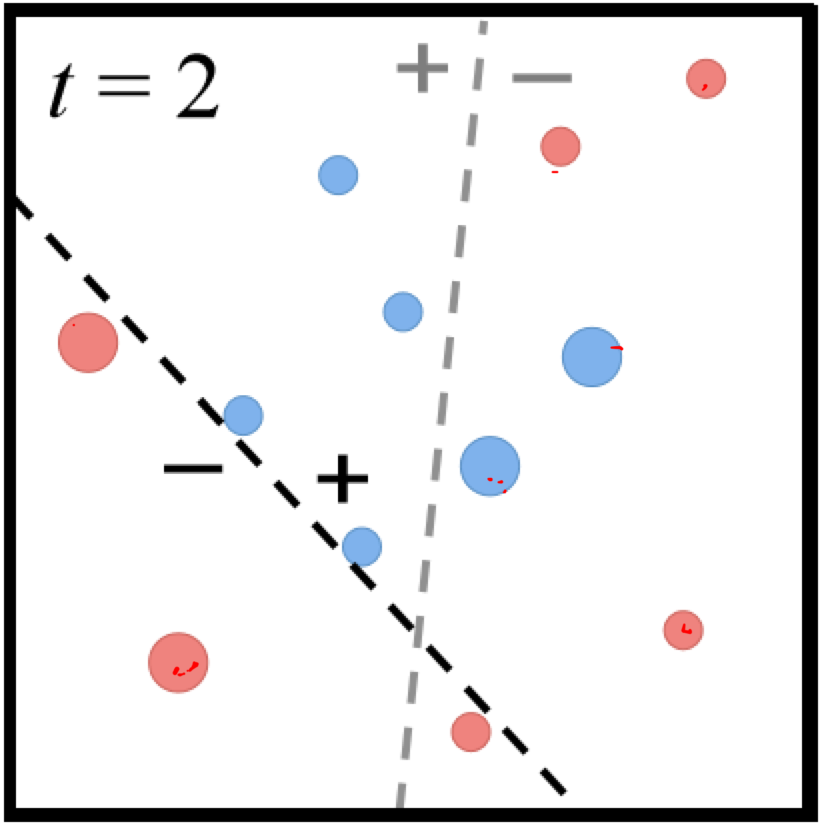
AdaBoost Algorithm

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

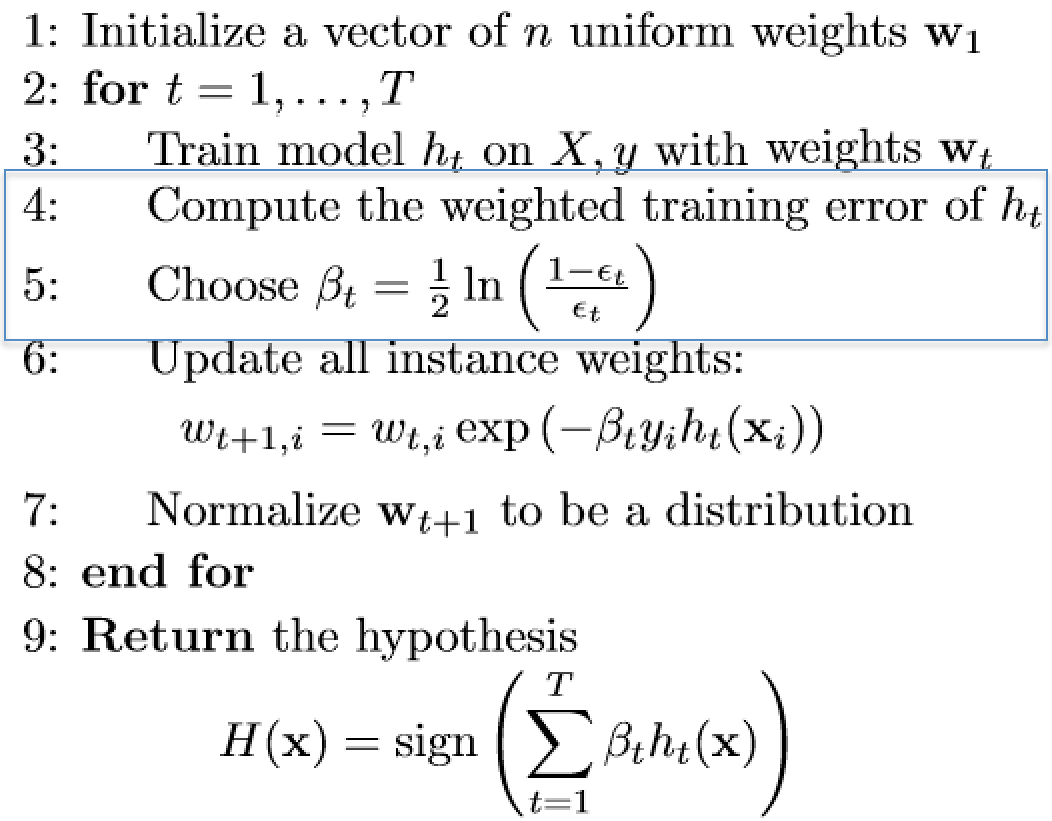


AdaBoost Algorithm

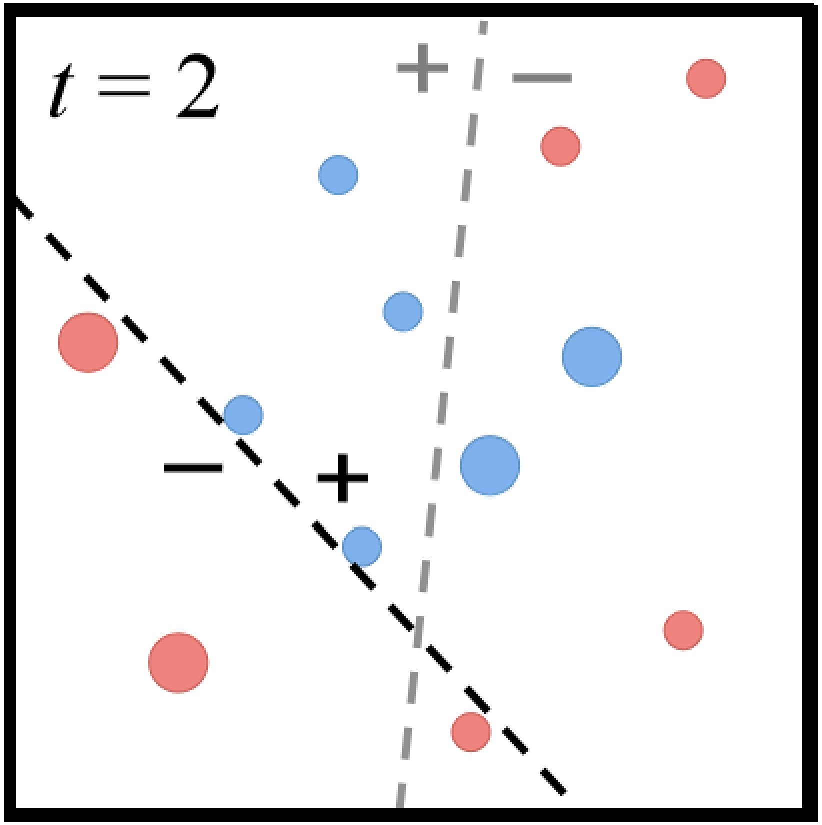
```
Initialize a vector of $n$ uniform weights $\mathbf{w}_1$
for $t = 1, \ldots, T$
    Train model $h_t$ on $X, y$ with weights $\mathbf{w}_t$
    Compute the weighted training error of $h_t$
    Choose $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$
    Update all instance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



AdaBoost Algorithm



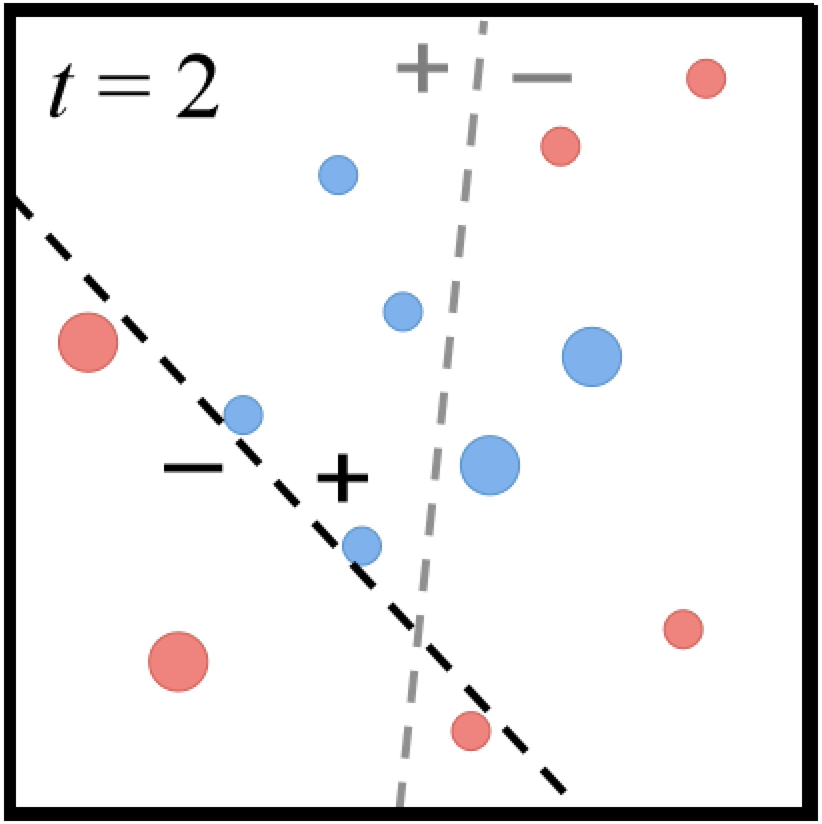
```
Initialize a vector of $n$ uniform weights $\mathbf{w}_1$
for $t = 1, \ldots, T$
    Train model $h_t$ on $X, y$ with weights $\mathbf{w}_t$
    Compute the weighted training error of $h_t$
    Choose $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$
    Update all instance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



- 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 Algorithm

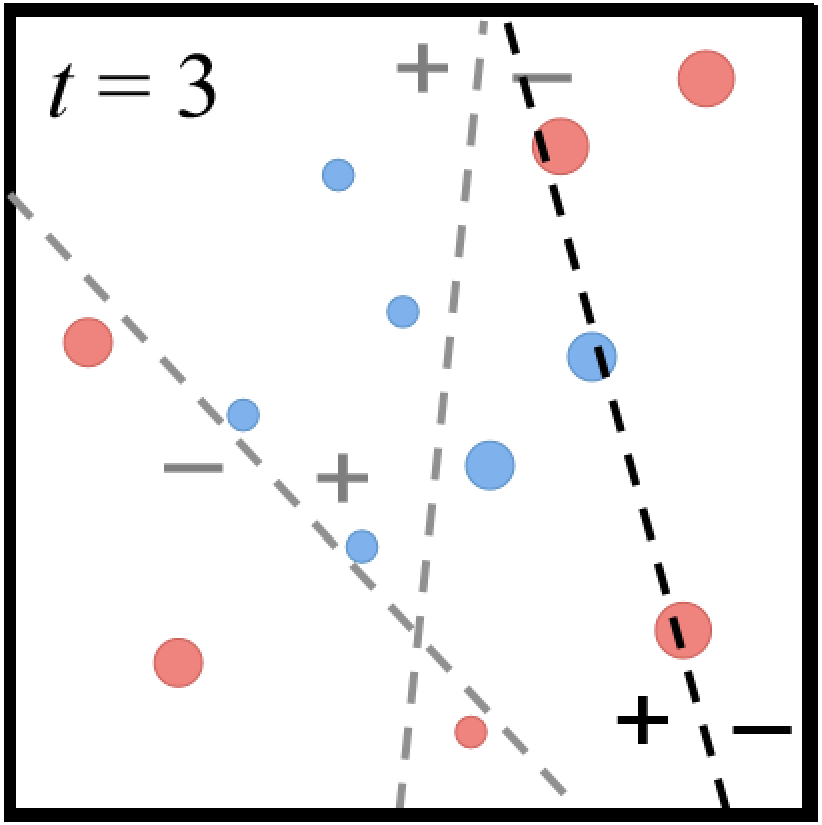
```
Initialize a vector of $n$ uniform weights $\mathbf{w}_1$
for $t = 1, \ldots, T$
    Train model $h_t$ on $X, y$ with weights $\mathbf{w}_t$
    Compute the weighted training error of $h_t$
    Choose $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$
    Update all instance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



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

AdaBoost Algorithm

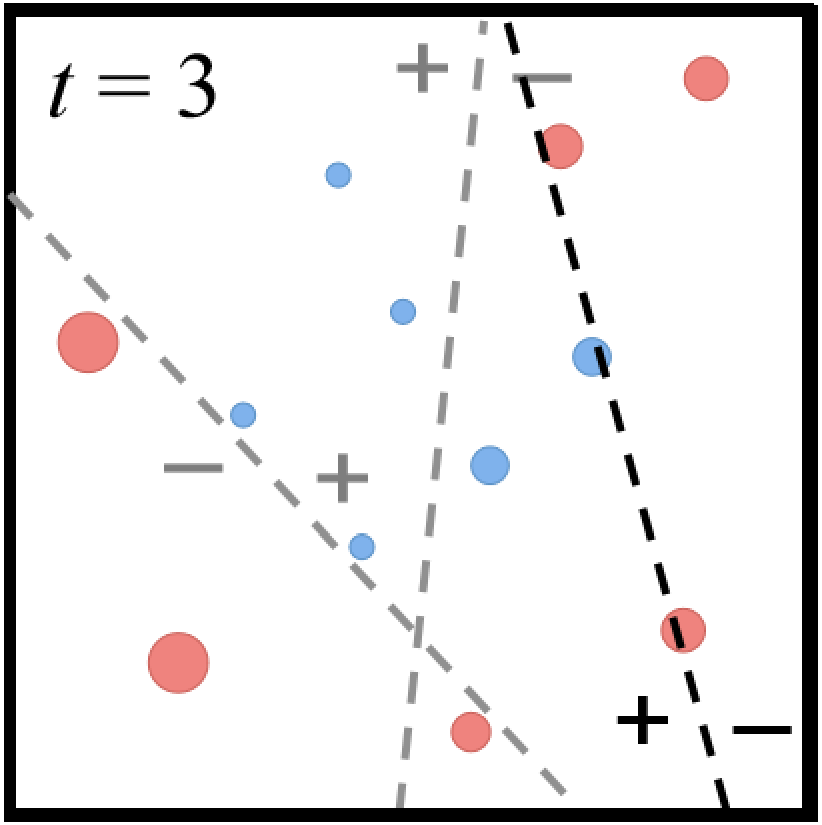
```
Initialize a vector of $n$ uniform weights $\mathbf{w}_1$
for $t = 1, \ldots, T$
    Train model $h_t$ on $X, y$ with weights $\mathbf{w}_t$
    Compute the weighted training error of $h_t$
    Choose $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$
    Update all instance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



- 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)

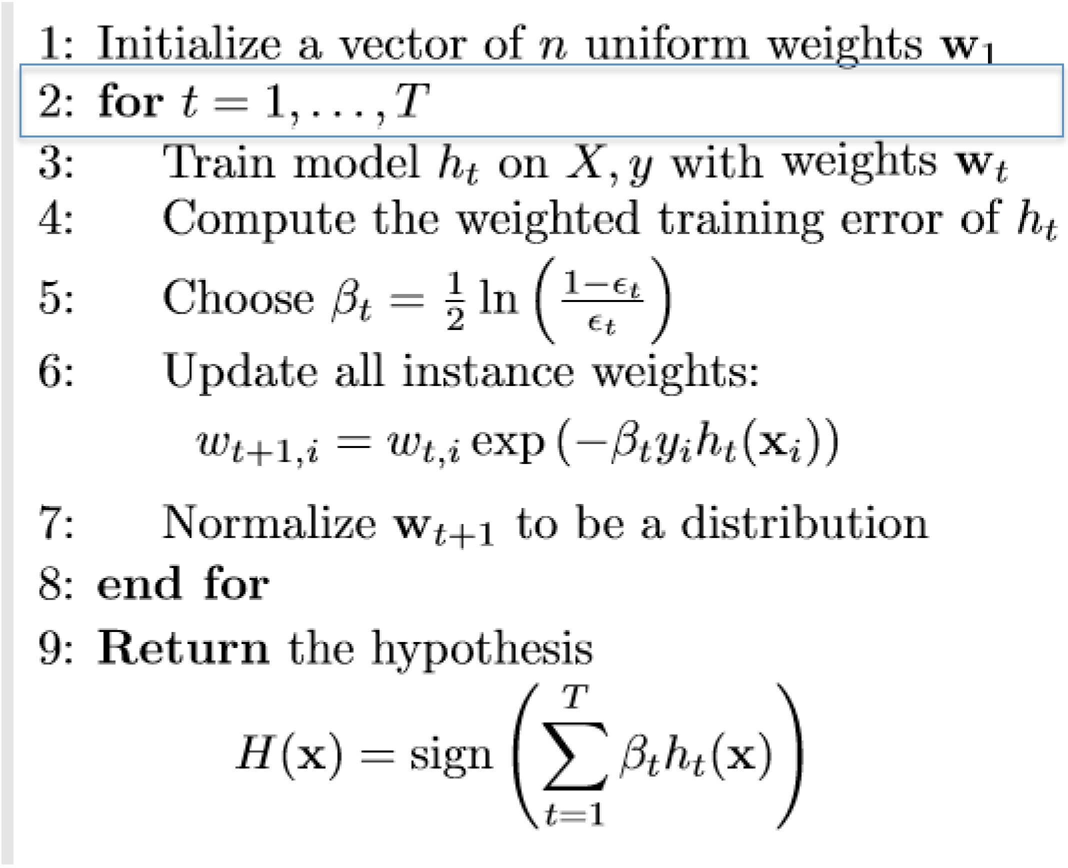
AdaBoost Algorithm

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

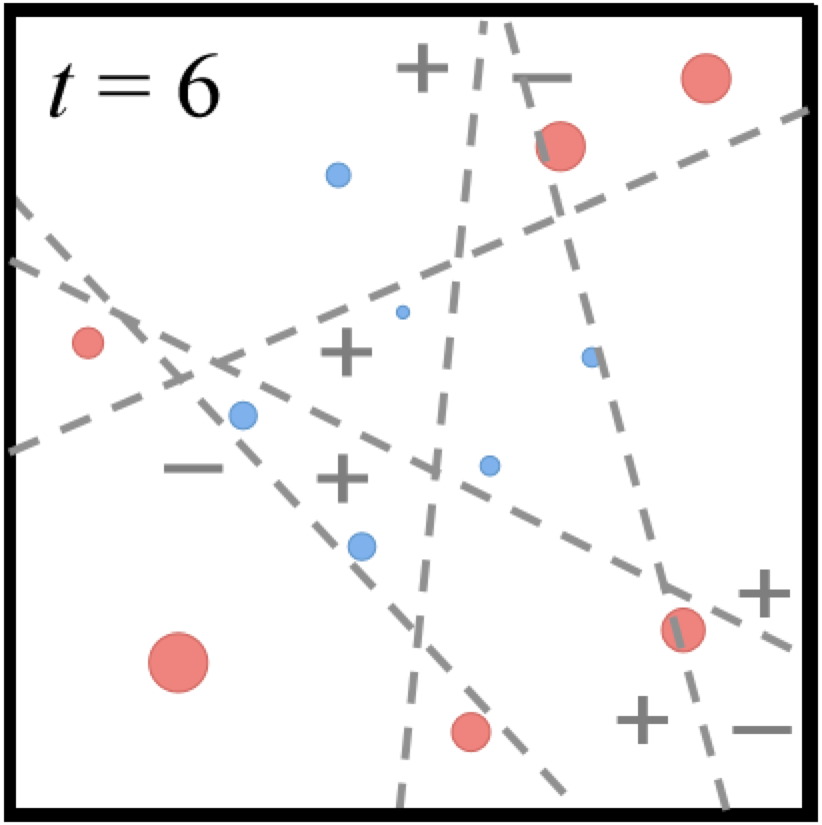


- Weights of correct predictions are multiplied by $e^{-\beta_t} \leq 1$
- Weights of incorrect predictions are multiplied by $e^{\beta_t} \geq 1$

AdaBoost Algorithm

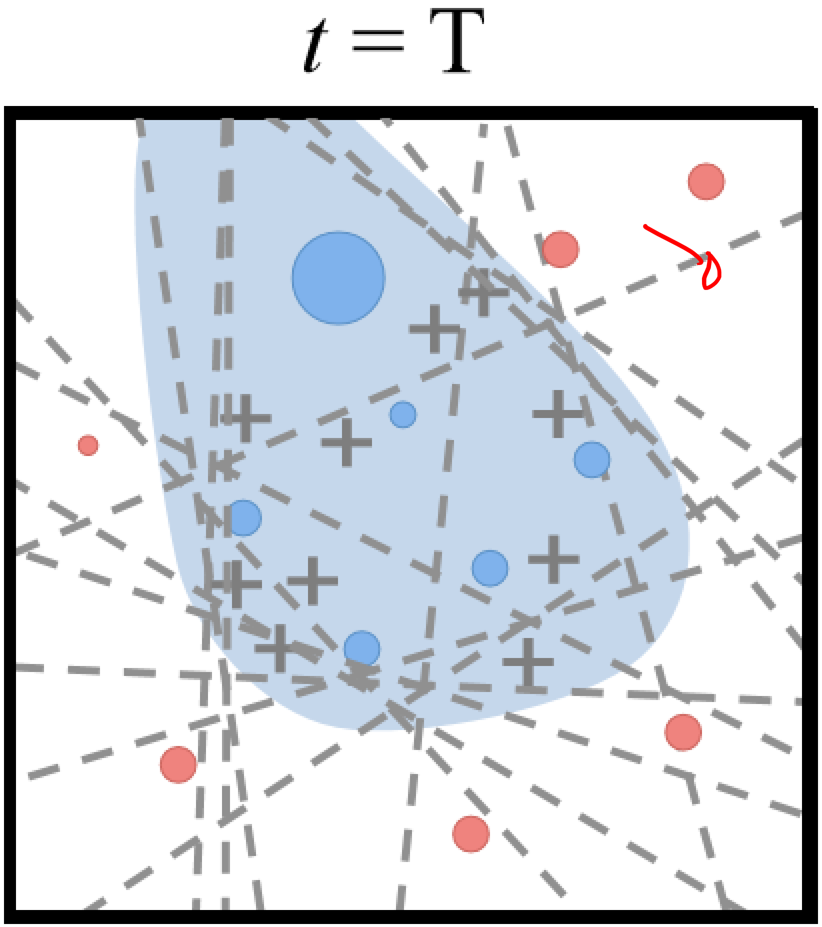


```
Initialize a vector of $n$ uniform weights $\mathbf{w}_1$
for $t = 1, \ldots, T$
    Train model $h_t$ on $X, y$ with weights $\mathbf{w}_t$
    Compute the weighted training error of $h_t$
    Choose $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$
    Update all instance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



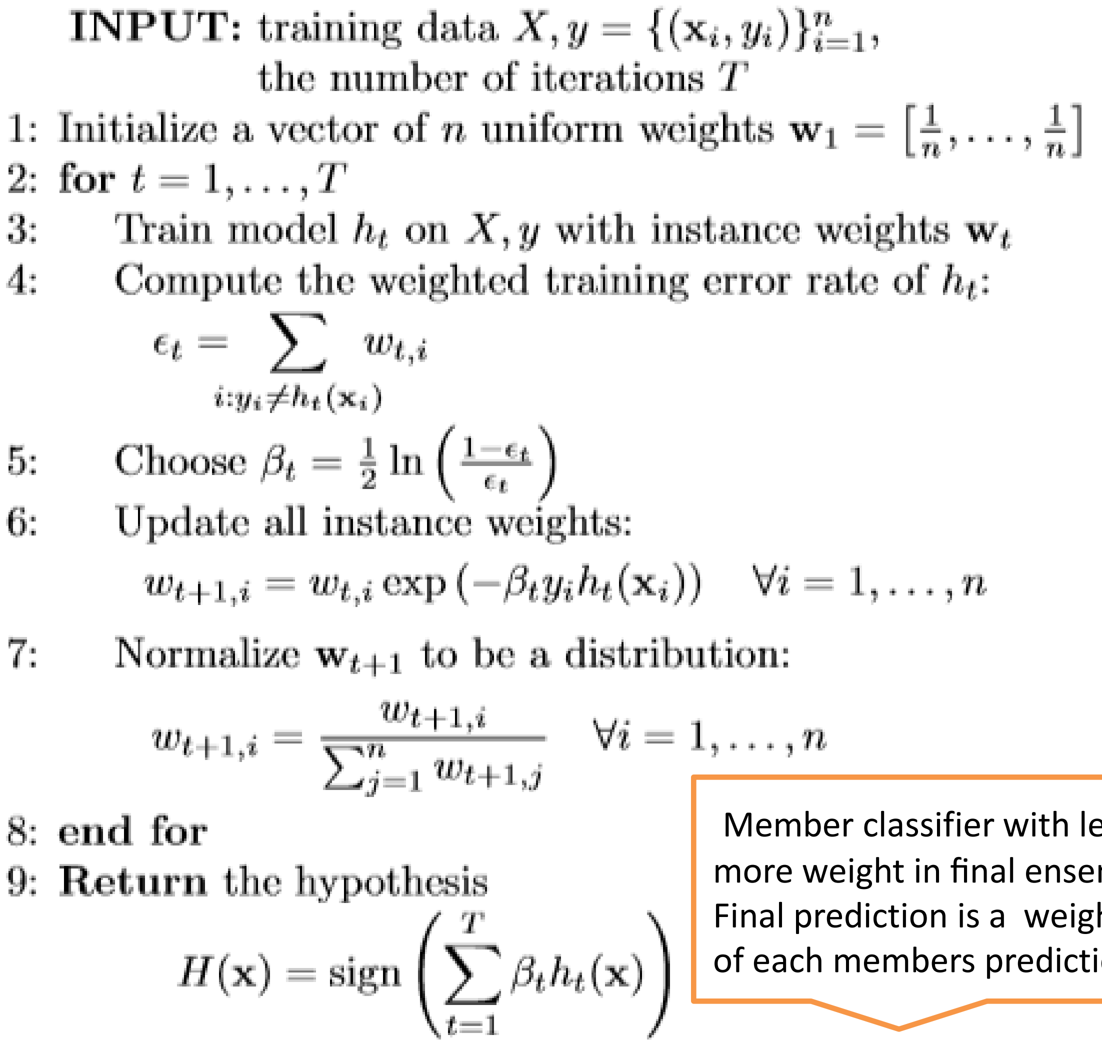
AdaBoost Algorithm

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



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

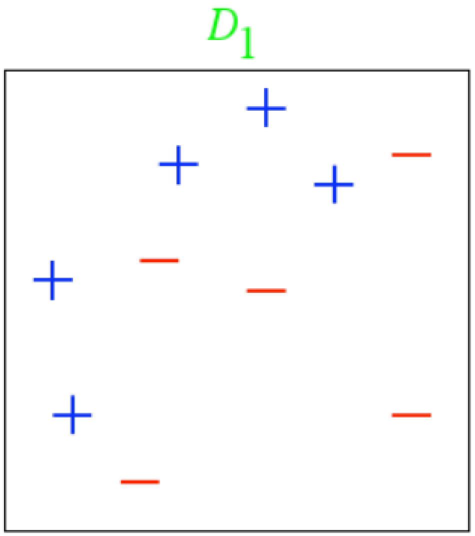
AdaBoost Algorithm



```
INPUT: training data $X, y = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$,
       the number of iterations $T$
Initialize a vector of $n$ uniform weights $\mathbf{w}_1 = \left[\frac{1}{n}, \ldots, \frac{1}{n}\right]$
for $t = 1, \ldots, T$
    Train model $h_t$ on $X, y$ with instance weights $\mathbf{w}_t$
    Compute the weighted training error rate of $h_t$:
        $\epsilon_t = \sum_{i : y_i \neq h_t(\mathbf{x}_i)} w_{t,i}$
    Choose $\beta_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
    Update all instance weights:
        $w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right) \quad \forall i = 1, \ldots, n$
    Normalize $\mathbf{w}_{t+1}$ to be a distribution:
        $w_{t+1,i} = \frac{w_{t+1,i}}{\sum_{j=1}^{n} w_{t+1,j}} \quad \forall i = 1, \ldots, n$
end for
Return the hypothesis
    $H(\mathbf{x}) = \operatorname{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x})\right)$
```

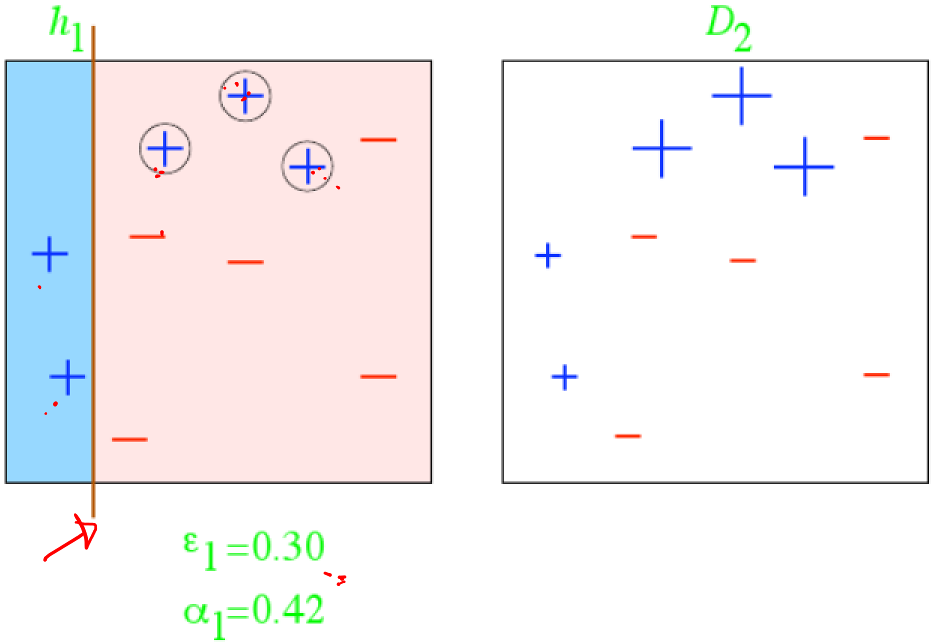
Member classifier with less error are given more weight in final ensemble hypothesis. Final prediction is a weighted combination of each members prediction

Example



From, L´eon Bottou

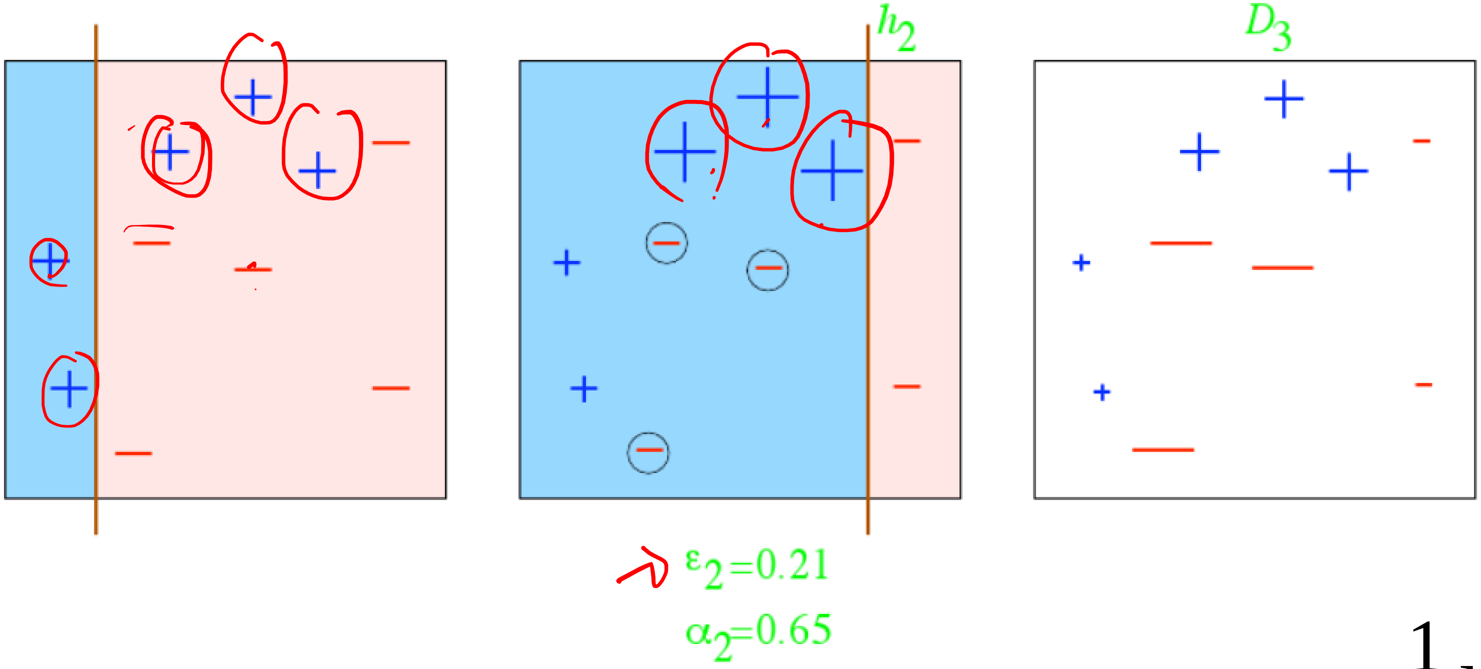
Example



From, L´eon Bottou

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right) \qquad \alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

Example

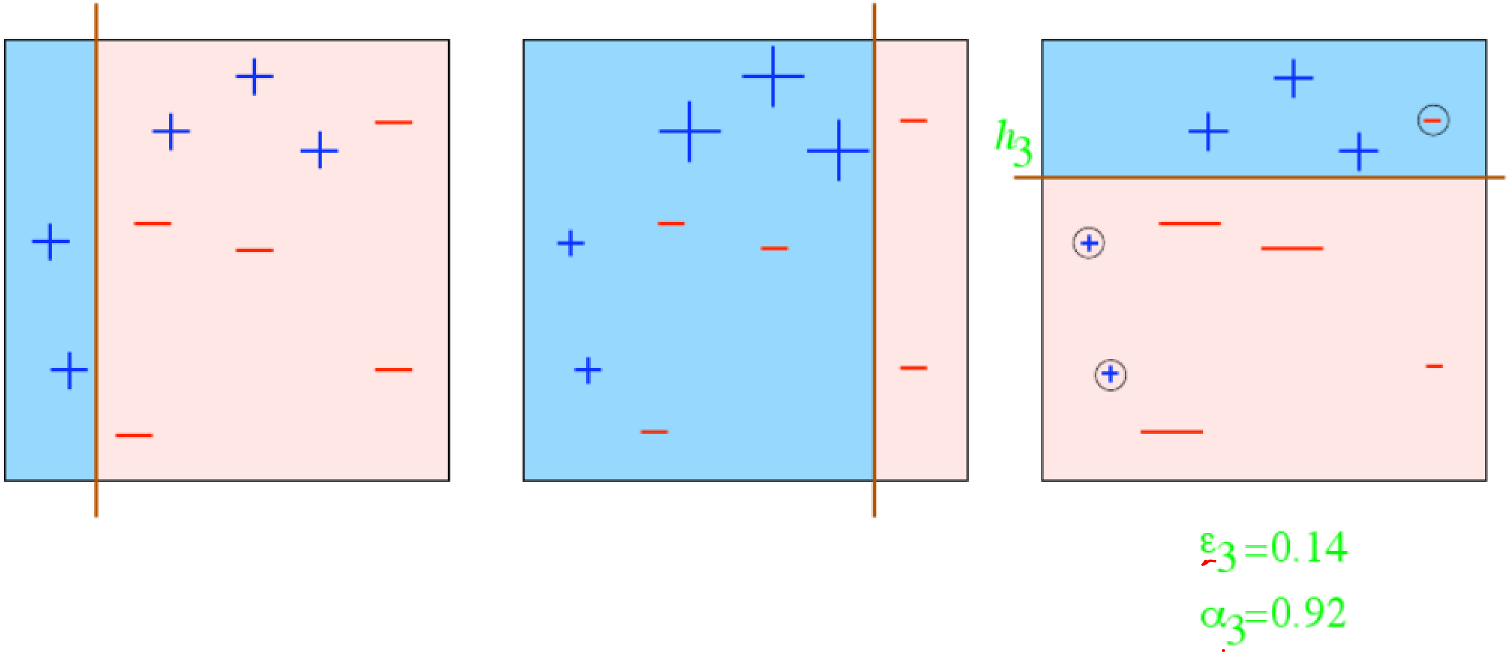


From, L´eon Bottou

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

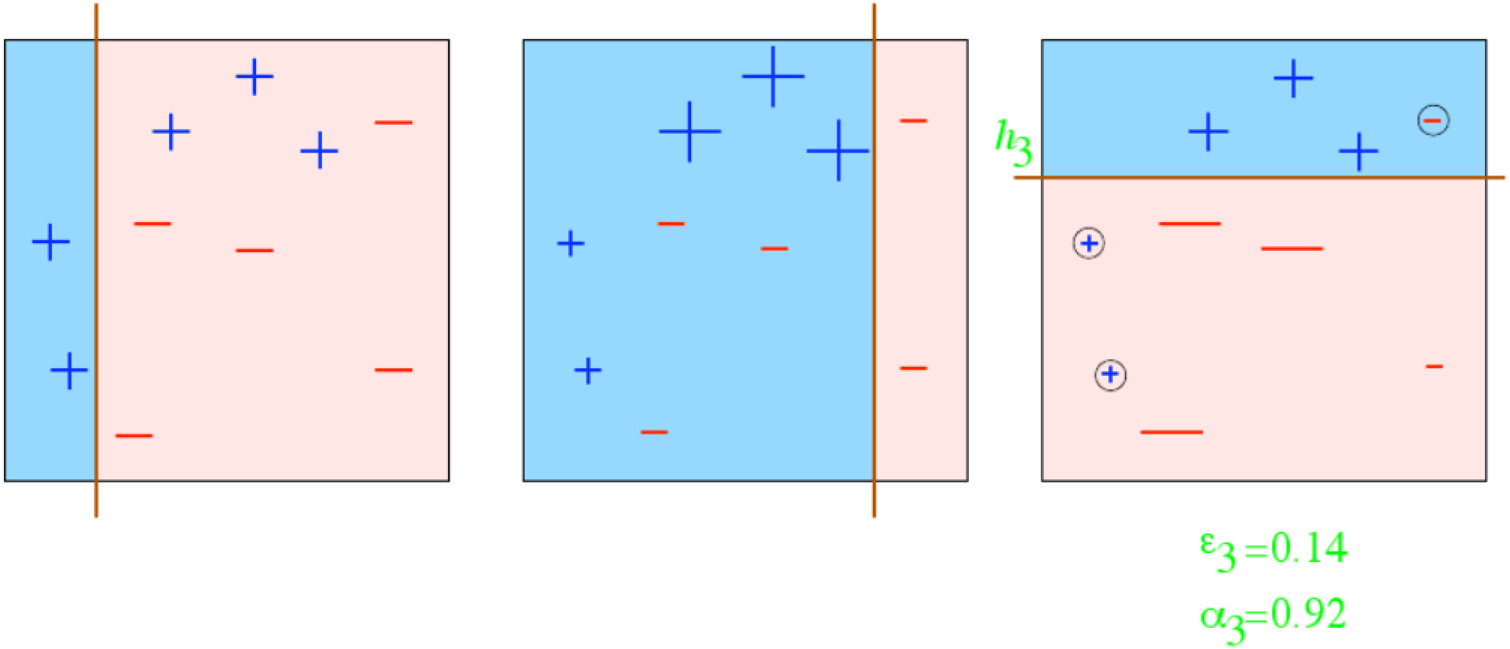
Example



From, L´eon Bottou

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

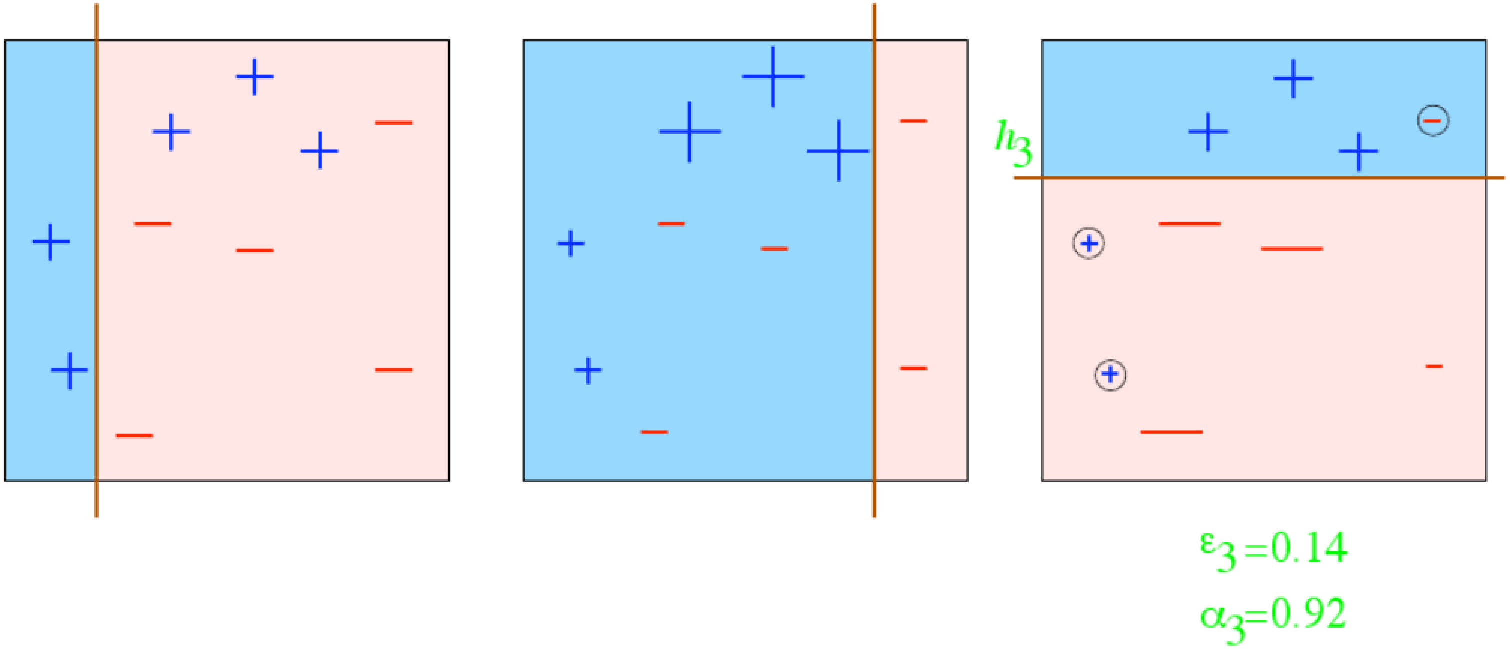
Example



From, L´eon Bottou

Example

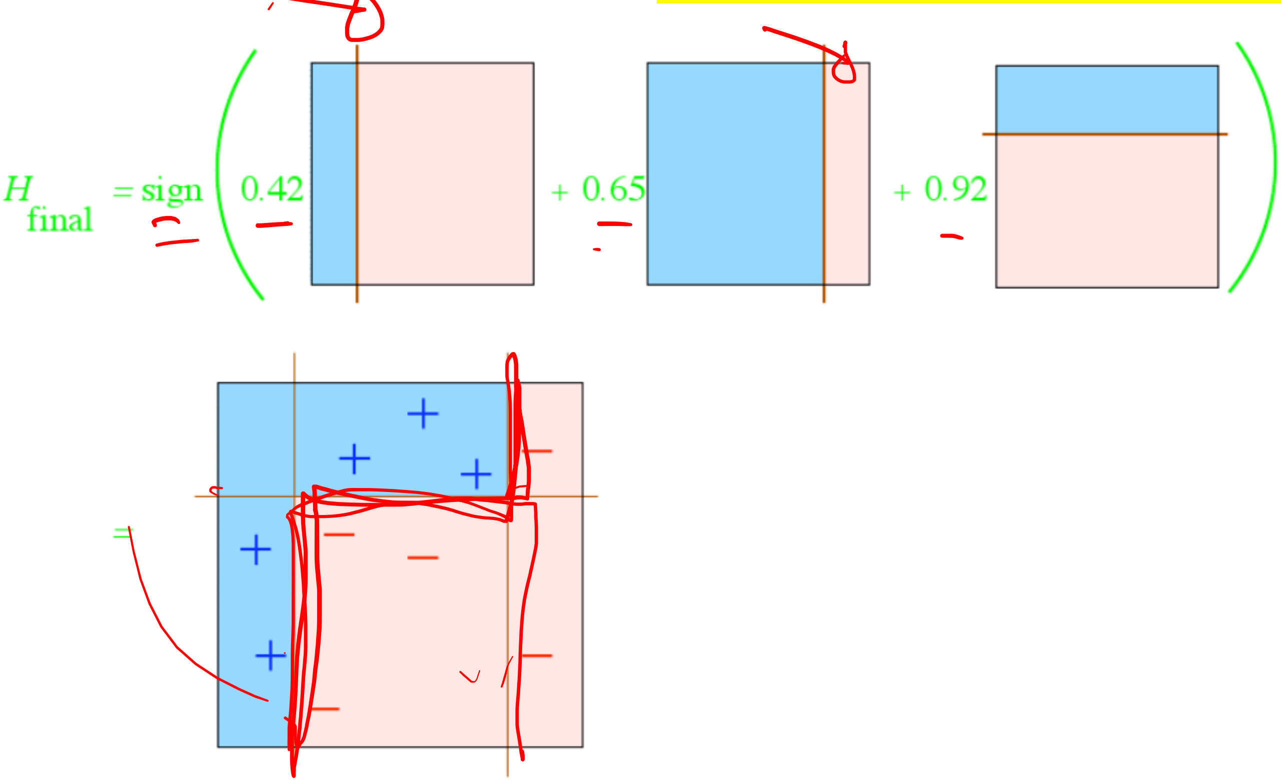
How do we combine the results now?



From, L´eon Bottou

Example

How do we combine the results now?


$H_{\text{final}} = \text{sign}\left(0.42 \; + 0.65 \; + 0.92\right)$
+
+
+
+
+
−
−
−
−
−

From, L´eon Bottou

AdaBoost Example

0	.1	.2	0.3	0.4	.5	.6	.7	.8	.9	1
	1	1	1	+1	-1	-1	-1	1	1	1

W_i's $= \frac{1}{10}$

$\epsilon_1 = \frac{1}{10}\left(\frac{1}{10} \times 3\right) = \frac{3}{100} = 0.03$

- Training sets for the first 3 boosting rounds:

0.75 = -1 0.75 +1

ϵ_i

α_i

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

$\ln$

$\frac{1}{2}\left(\frac{1-0.03}{0.03}\right)$

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

- Summary:

Round	Split Point	Left Class	Right Class	alpha
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

AdaBoost Example

- Weights

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	0.029	0.029	0.029	0.228	0.228	0.228	0.228	0.009	0.009	0.009

- Classification

	Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
	1	-1	-1	-1	-1	-1	-1	-1	1	1	1
	2	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	-1	-1	-1	-1	-1	-1	-1
	Sum	5.16	5.16	5.16	-3.08	-3.08	-3.08	-3.08	0.397	0.397	0.397
Predicted Class	Sign	1	1	1	-1	-1	-1	-1	1	1	1

AdaBoost error function takes into account the fact that only the sign of the final result is used, thus sum can be far larger than 1 without increasing error

AdaBoost base learners

- 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 in practice

Strengths:

- 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

Fine Tuning Ensembles

- Model combination does not always guaranteed to decrease error, unless
  - base-learners are diverse and accurate
- Ignore poor base learners
  - Use accuracy as a cut-off
  - Introduce some pruning with which at each iteration remove poor learners / learners whose absence lead to improvement (if any)
    - Modify iterations to allow both additions / deletions of learners
  - Discarding appropriately leads to better performance

Gradient Boosting

- In Gradient Boosting, "shortcomings” are identified by gradients.
- Recall that, in Adaboost, “shortcomings” are identified by high-weight data points.
- Both high-weight data points and gradients tell us how to improve our model.

Gradient Boosting

- Gradient Boosting for Different Problems Difficulty: regression ===> classification ===> ranking

Gradient Boosting

- You are given (x1, y1),(x2, y2), ...,(xn, yn), and the task is to fit a model F(x) to minimize square loss
- There are some mistakes: F(x1) = 0.8, while y1 = 0.9, and F(x2) = 1.4 while y2 = 1.3... How can you improve this model?
- Rules:
  - You are not allowed to remove anything from F or change any parameter in F.
  - You can add an additional model (regression tree) h to F, so the new prediction will be F(x) + h(x).

Gradient Boosting

- You wish to improve the model such that
  - F(x1) + h(x1) = y1
  - F(x2) + h(x2) = y2 ...
  - F(xn) + h(xn) = yn

Or, equivalently, you wish

h(x1) = y1 − F(x1)

h(x2) = y2 − F(x2) ...

h(xn) = yn − F(xn)

Fit a regression tree h to data

(x1, y1 − F(x1)),(x2, y2 − F(x2)), ...,(xn, yn − F(xn))

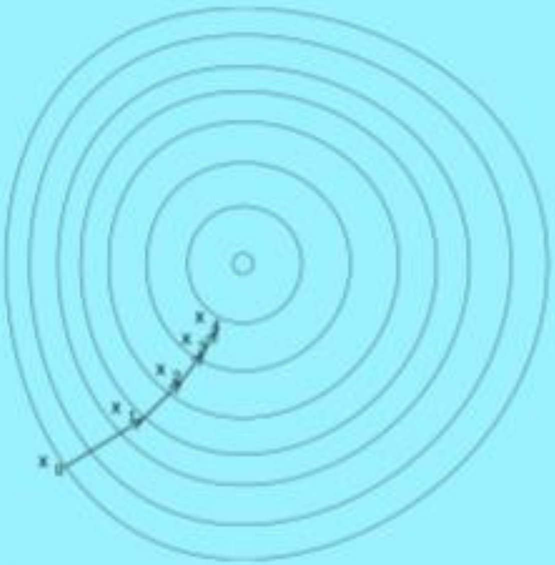
Gradient Boosting

- Simple solution: yi − F(xi) are called residuals. These are the parts that existing model F cannot do well.
- The role of h is to compensate the shortcoming of existing model F.
- If the new model F + h is still not satisfactory, we can add another regression tree...
- We are improving the predictions of training data, is the procedure also useful for test data?

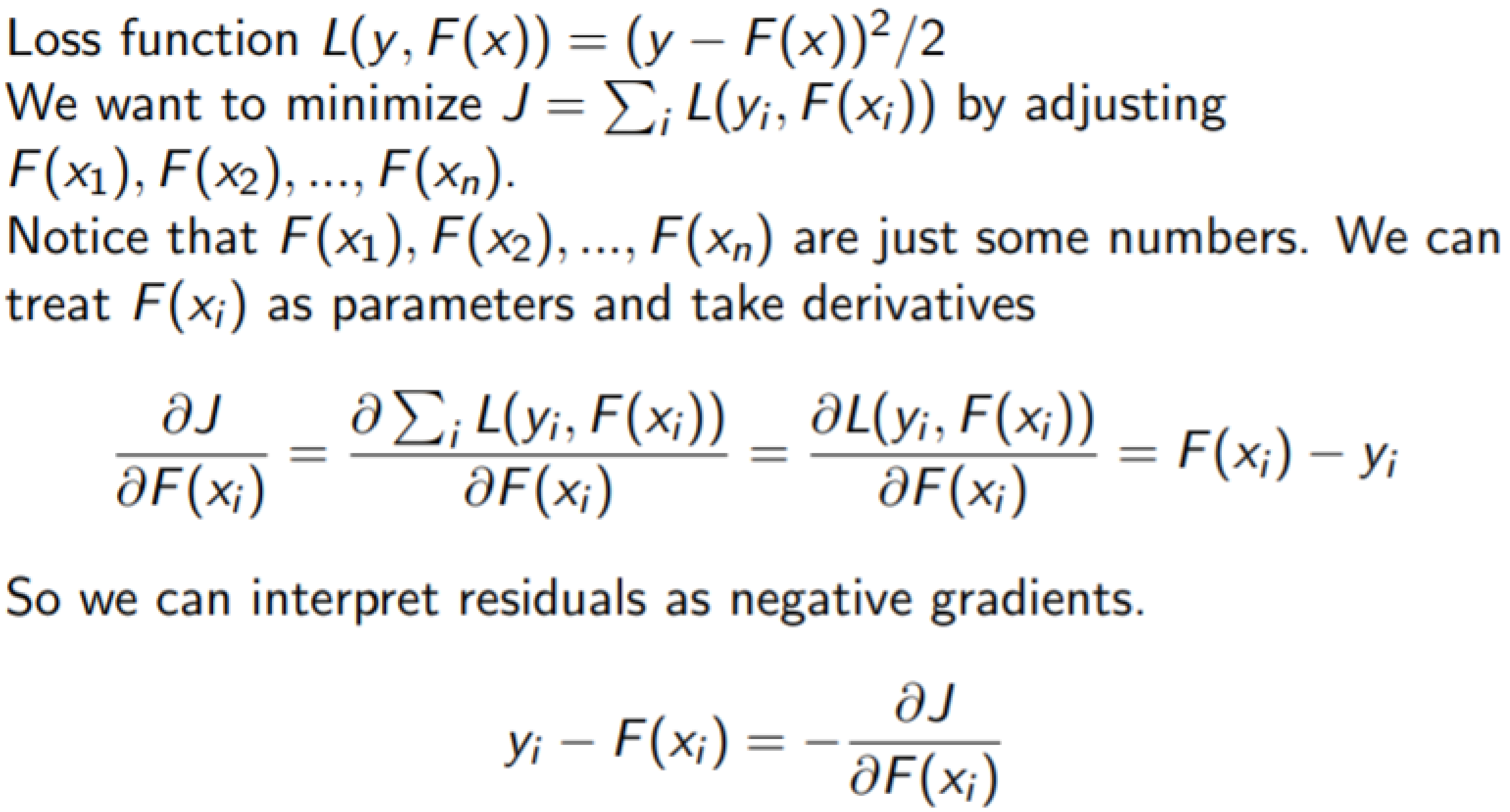
Gradient Descent

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

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



Gradient Boosting for regression



Loss function $L(y, F(x)) = (y - F(x))^2/2$
We want to minimize $J = \sum_i L(y_i, F(x_i))$ by adjusting $F(x_1), F(x_2), ..., F(x_n)$.
Notice that $F(x_1), F(x_2), ..., F(x_n)$ are just some numbers. We can treat $F(x_i)$ as parameters and take derivatives

$$\frac{\partial J}{\partial F(x_i)} = \frac{\partial \sum_i L(y_i, F(x_i))}{\partial F(x_i)} = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = F(x_i) - y_i$$

So we can interpret residuals as negative gradients.

$$y_i - F(x_i) = -\frac{\partial J}{\partial F(x_i)}$$

$$F(x_i) := F(x_i) + h(x_i)$$
$$F(x_i) := F(x_i) + y_i - F(x_i)$$
$$F(x_i) := F(x_i) - 1\frac{\partial J}{\partial F(x_i)}$$
$$\theta_i := \theta_i - \rho\frac{\partial J}{\partial \theta_i}$$

For regression with **square loss**,

residual ⇔ negative gradient

fit h to residual ⇔ fit h to negative gradient

update F based on residual ⇔ update F based on negative gradient

So we are actually updating our model using **gradient descent**!

Gradient Boosting Algorithm

- It involves three elements
  - A loss function to be optimized (minimizes expected value)

$$\hat{F} = \arg\min_{F} \mathbb{E}_{x,y}[L(y, F(x))]$$

  - Approximation of F(x) in terms of weighted sum of base(weak) learners $h_i(x)$ to make

$$\hat{F}(x) = \sum_{i=1}^{M} \gamma_i h_i(x) + \text{const}$$

  - An additive model to minimize the loss function, starting with $F_0(x)$ and incrementally expanding in greedy fashion

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma),$$

$$F_m(x) = F_{m-1}(x) + \arg\min_{h_m \in \mathcal{H}} \left[\sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + h_m(x_i)) \right]$$

Why XGBoost so popular?

- **Speed** : faster than other ensemble classifiers.
- **Core algorithm is parallelizable**: harness the power of multi-core computers and networks of computers enabling to train on very large datasets **Consistently outperforms other algorithm methods** : It has shown better performance on a variety of machine learning benchmark datasets.
- **Wide variety of tuning parameters** : cross-validation, regularization, missing values, tree parameters, etc
- XGBoost (Extreme Gradient Boosting) uses the gradient boosting (GBM) framework at its core.

References

The-Morgan-Kaufmann-Series-in-Data-Management-Systems-Jiawei-Han-Micheline-Kamber-Jian-Pei-Data-Mining.-Concepts-and-Techniques-3rd-Edition-Morgan-Kaufmann-2011

Bishop - Pattern Recognition And Machine Learning - Springer 2006

A Gentle Introduction to Gradient Boosting
Cheng Li chengli@ccs.neu.edu College of Computer and Information Science Northeastern University

https://www.youtube.com/watch?time_continue=647&v=LsK-xG1cLYA&feature=emb_logo

Thank You!