EECS 545: Machine Learning

Lecture 21. Decision Tree and Ensemble Methods

Honglak Lee, Yiwei Lyu

03/31/2025

UNIVERSITY OF MICHIGAN

Logistics

- HW5 due 4/1 (tomorrow)
- Exam Review on 4/7 lecture time
- Midterm Exam on 4/9 6-9 pm. Room will be announced via canvas

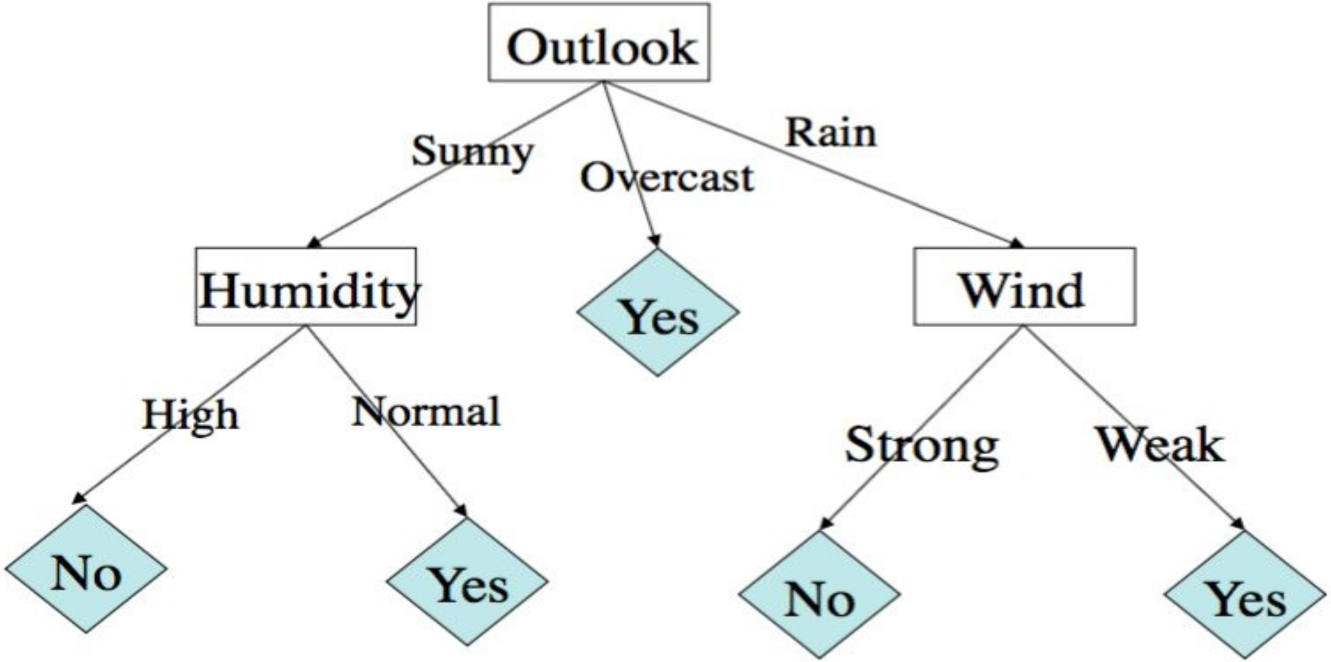
Outline

- **Decision tree**
- Overview of ensemble methods
  - Bagging
  - Boosting
- AdaBoost

Data on "Play Tennis"

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

A Possible Decision Tree



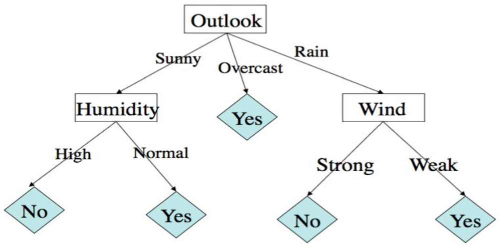
Intuitions

- A decision tree emulates a human’s decision making process.

- A decision is made by examining (not necessarily all) features sequentially.

- A feature will not be examined twice in the decision making process. This implies that a feature will not show up more than once in a path from the root node to the leaf node.

Training



- In the training process, we want to grow the decision tree from root to leaf node in an “optimal” way
- Concretely, we wish to decide
  - the features attached to the internal nodes
  - the realization of a feature in an edge
  - the label of a leaf node
- This is done by maximizing the “information gain”

Entropy

- The entropy of a random variable X is defined as

$$\mathrm{H}(X) = -\sum_{j=1}^{M} p(x_j) \log_2 p(x_j)$$

$$\text{where} \quad p(x_j) = P(X = x_j)$$

- Intuitively, the entropy of a random variable measures how uncertain we are about it
- The entropy is maximized when the realizations are equi-probable, namely

$$p(x_1) = p(x_2) = \cdots = p(x_M) = \frac{1}{M}$$

Entropy

- If X follows the following multinomial distribution

P(X=a)	1/4
P(X=b)	1/4
P(X=c)	1/4
P(X=d)	1/4

- The entropy of X is

$$\mathrm{H}(X) = -\left[\frac{1}{4}\log\left(\frac{1}{4}\right) + \frac{1}{4}\log\left(\frac{1}{4}\right) + \frac{1}{4}\log\left(\frac{1}{4}\right) + \frac{1}{4}\log\left(\frac{1}{4}\right)\right] = 2$$

Entropy

- If X follows the following multinomial distribution

P(X=a)	1/2
P(X=b)	1/4
P(X=c)	1/8
P(X=d)	1/8

- The entropy of X is

$$\mathrm{H}(X) = -\left[\frac{1}{2}\log\left(\frac{1}{2}\right) + \frac{1}{4}\log\left(\frac{1}{4}\right) + \frac{1}{8}\log\left(\frac{1}{8}\right) + \frac{1}{8}\log\left(\frac{1}{8}\right)\right] = 1.75$$

Entropy

- If X follows the following multinomial distribution

P(X=a)	1
P(X=b)	0
P(X=c)	0
P(X=d)	0

- The entropy of X is

$$\mathrm{H}(X) = -\left[1 \log 1 + 0 \log 0 + 0 \log 0 + 0 \log 0\right] = 0$$

In the above calculation, we defined $0 \log 0 := 0$. Note: $\lim_{x \to 0} x \log(x) = 0$

Conditional Entropy

- The conditional entropy of *X* given *Y* (another random variable) is defined as

$$\begin{aligned}
\mathrm{H}(X|Y) &= -\sum_{i=1}^{N}\sum_{j=1}^{M} p(x_j, y_i)\log p(x_j|y_i) \\
&= \sum_{i=1}^{N} p(y_i) \sum_{j=1} -p(x_j|y_i)\log p(x_j|y_i) \\
&= \sum_{i=1}^{N} p(y_i)\mathrm{H}(X|y_i)
\end{aligned}$$

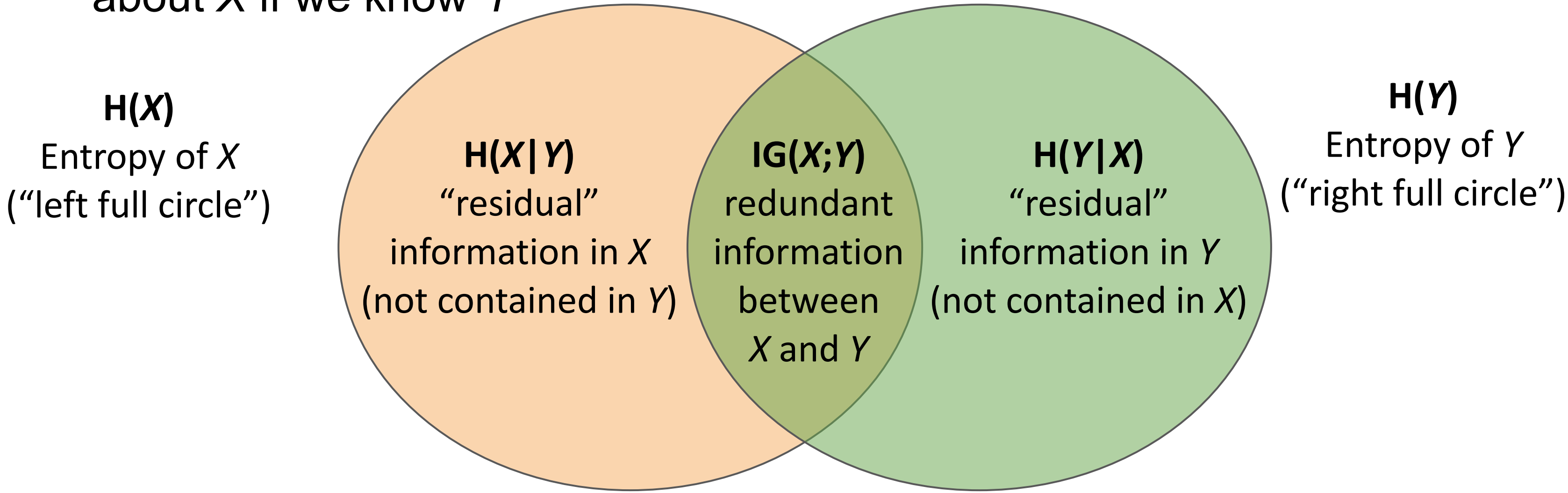
- Intuitively it is the expected entropy of *X* given different realizations of *Y*

Information Gain & Node Splitting

- The information gain (also known as mutual information) is defined as:

$$
\begin{aligned}
IG(X,Y) &= \mathrm{H}(X) - \mathrm{H}(X|Y) \\
&= \mathrm{H}(Y) - \mathrm{H}(Y|X)
\end{aligned}
$$

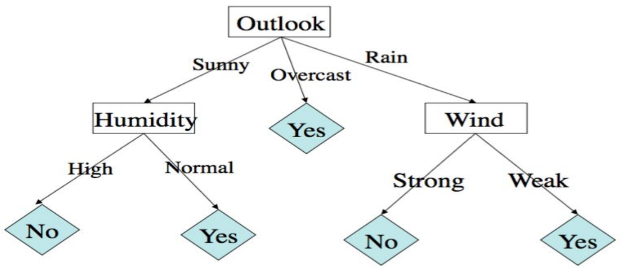
- Intuitively, the information gain measures how much more certain we are about *X* if we know *Y*



Note: IG can be calculated for particular splits of the dataset corresponding to an intermediate node in the decision tree. If *X* and *Y* are just two random variables with full range of values, IG is also called mutual information.

Training of Decision Tree

- When deciding which feature to use in an internal node of a decision tree, we wish to use the one that maximizes the information gain.
- As we go deeper with the hierarchies, we only consider the subset of data points that satisfy the conditions along the path.



Decision Tree on Playing Tennis

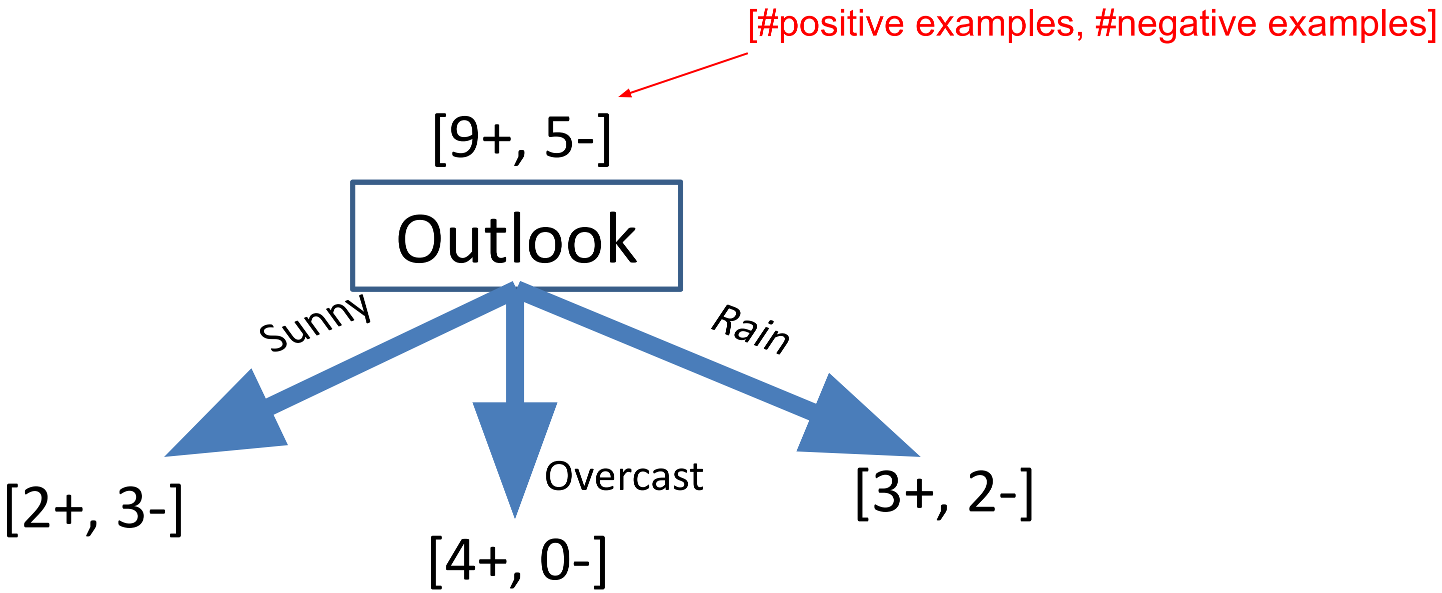
- Originally, in the whole dataset, the person played tennis on 9 out of the 14 days. Hence, the entropy of PlayTennis (*P*) is

$$\mathrm{H}(P) = -\left[\frac{9}{14}\log\frac{9}{14} + \frac{5}{14}\log\frac{5}{14}\right]$$
$$= 0.94$$

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Decision Tree on Playing Tennis

- Suppose we use outlook (0) as the root node, then the data will be split as



Decision Tree on Playing Tennis

- The conditional entropy of PlayTennis (P) given Outlook (O) is

$$
\begin{aligned}
\mathrm{H}(P|O) =& p(O=\text{sunny})\mathrm{H}(P|O=\text{sunny}) \\
& + p(O=\text{overcast})\mathrm{H}(P|O=\text{overcast}) \\
& + p(O=\text{rain})\mathrm{H}(P|O=\text{rain}) \\
=& \frac{2+3}{14}\left[-\frac{2}{5}\log\frac{2}{5}-\frac{3}{5}\log\frac{3}{5}\right] \\
& + \frac{4+0}{14}\left[-\frac{4}{4}\log\frac{4}{4}-\frac{0}{4}\log\frac{0}{4}\right] \\
& + \frac{3+2}{14}\left[-\frac{3}{5}\log\frac{3}{5}-\frac{2}{5}\log\frac{2}{5}\right] \\
=& 0.694
\end{aligned}
$$

- The information gain is therefore:

$$
\begin{aligned}
IG(P,O) &= \mathrm{H}(P) - \mathrm{H}(P|O) \\
&= 0.94 - 0.694 = 0.246
\end{aligned}
$$

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Decision Tree on Playing Tennis

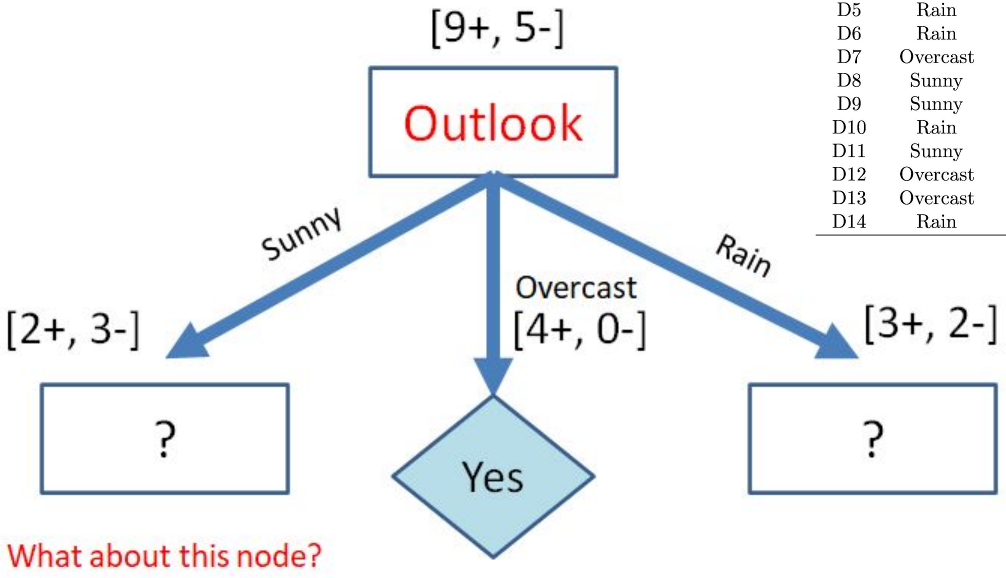
- Similarly, we can compute the following:
    - IG(PlayTennis, Wind) = 0.048
    - IG(PlayTennis, Outlook) = 0.246
    - IG(PlayTennis, Humidity) = 0.151
    - IG(PlayTennis, Temperature) = 0.029

- Therefore outlook is selected as the root node attribute because it produces the max information gain

Tree Grown So Far

- Now we have decided outlook as the root node

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No



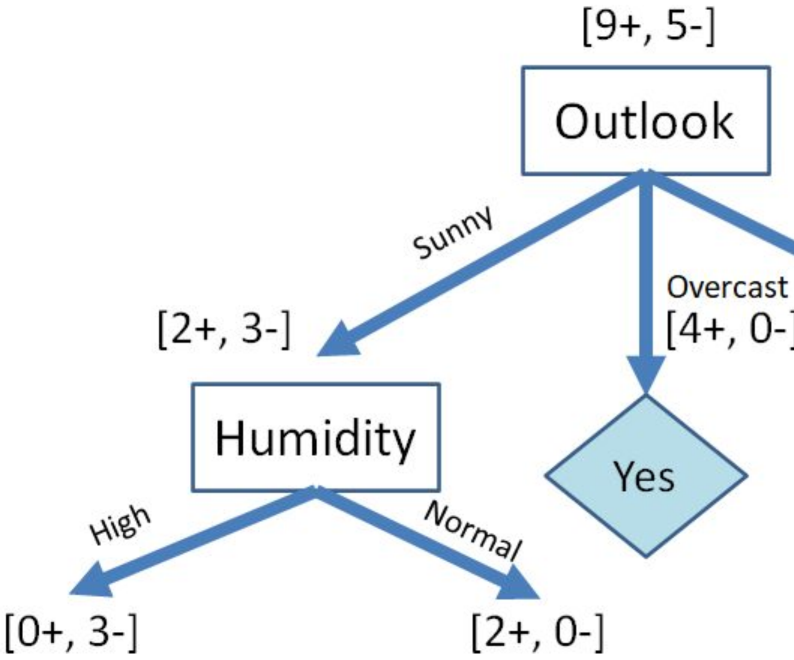
Decision Tree on Playing Tennis

- Given the outlook is sunny, the conditional entropy is

$$\begin{aligned}\mathrm{H}(P|O = \text{sunny}) &= -\frac{2}{2+3}\log\frac{2}{5} - \frac{3}{2+3}\log\frac{3}{5}\\ &= 0.97\end{aligned}$$

Decision Tree on Playing Tennis

- If we choose humidity for the next internal node



Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Decision Tree on Playing Tennis

- The conditional entropy of PlayTennis (*P*) given Humidity (*Hm*) and outlook being sunny is:

$$\mathrm{H}(P|Hm, O = \text{sunny}) = \frac{3}{5}\left[-\frac{0}{3}\log\frac{0}{3} - \frac{3}{3}\log\frac{3}{3}\right]$$
$$\frac{2}{5}\left[-\frac{2}{2}\log\frac{2}{2} - \frac{0}{2}\log\frac{0}{2}\right]$$
$$= 0$$

Note:

- Zero conditional entropy means “no uncertainty”.

- In other words, when the outlook is sunny, humidity determines PlayTennis(*P*).

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Decision Tree on Playing Tennis

- The information gain is

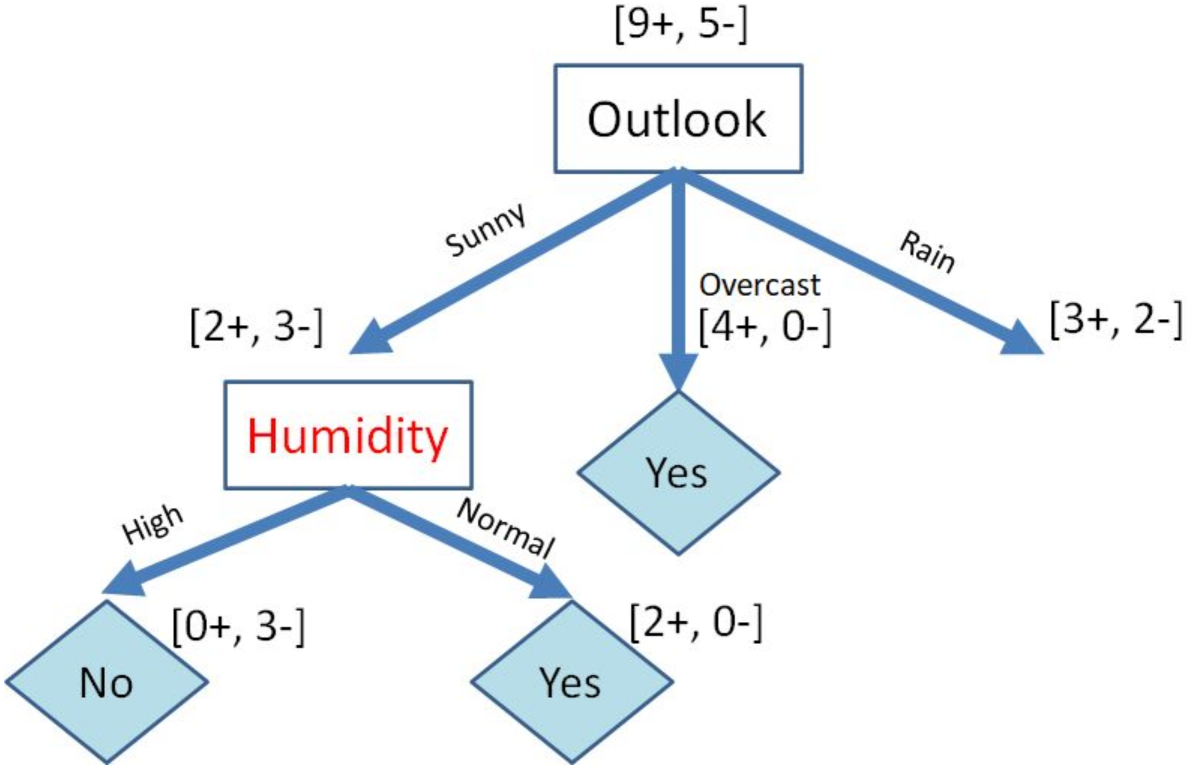
$$IG(P|O = \text{sunny}, Hm) = 0.97 - 0 = 0.97$$

- Similarly

$$IG(P|O = \text{sunny}, Temp) = 0.57$$
$$IG(P|O = \text{sunny}, Wind) = 0.019$$

- And so humidity gets chosen for this node

Decision Tree on Playing Tennis



Decision Tree on Playing Tennis

- Recursively use information gain to grow the tree until
  - A leaf node appears that represents only a single pure class (all yes or all no) or no feature can be used
  - Early stopping condition met

- Early stopping: stop growing the tree on this branch if the current node represents no more than K (a hyper-parameter) data points. This can reduce the risk of over-fitting.

Pruning

- It’s another way to avoid overfitting.
- Pruning a node means removing subtree rooted at that node and then assigning to that node the majority class from the data in that subtree
- Partition the data into training and validation set.
- Consider each of the decision-nodes in the tree as candidates for pruning
- Prune a node if the resulting tree performs no worse on the validation data than the original tree

Outline

- Decision tree
- **Overview of ensemble methods**
    - Bagging
    - Boosting
- AdaBoost

Learning with Ensembles

Idea behind ensemble learning:

- Train multiple models (classifiers, etc.) and combine them to improve the prediction.

Well-known examples of ensemble methods:

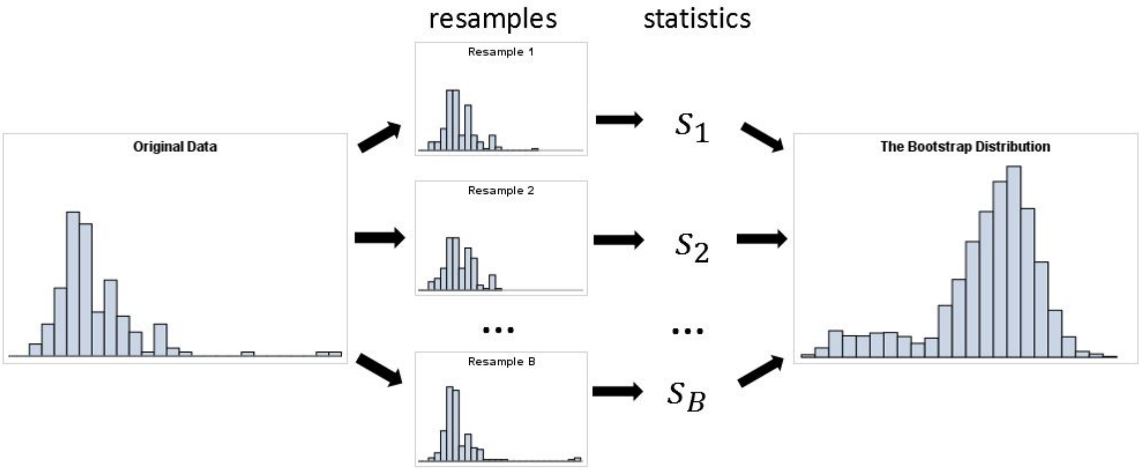
- Bootstrapping
- Bagging (Breiman 1996)
- Boosting (Schapire 1989; Kearns & Valiant 1989)
- Adaboost (Freund and Schapire 1995)

History

- Kearns & Valiant (1989)
  - Proved the astonishing fact that learners, each performing only slightly better than random, can be combined to form an arbitrarily good ensemble hypothesis
- Schapire (1990)
  - First to provide a provably polynomial time boosting algorithm
- Schapire et al. (1993)
  - First application of boosting to real-world OCR task
- Schapire et al. (1994-1996)
  - Adaboost Algorithm

Bootstrap Estimation

- Repeatedly draw *n* samples from *D*
- For each set of samples, estimate a statistic
  - E.g., mean, variance, skewness, kurtosis, etc.
- The bootstrap estimate is the mean of the individual estimates
- Bootstrapping is also used to estimate the distribution of the statistic

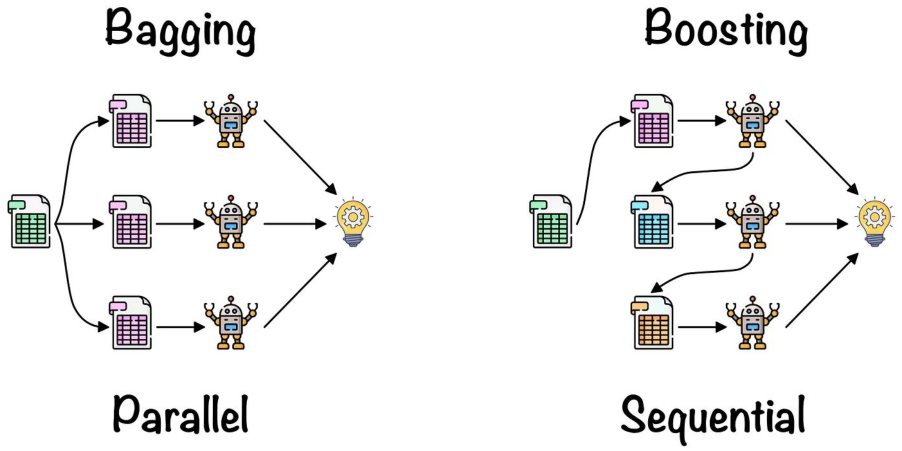


Ensemble Learning: framework

- Data: $\{(\mathbf{x}^{(n)}, y^{(n)}) : n = 1, \ldots, N\}$ where $\mathbf{x}^{(n)} \in \mathbb{R}^d, y \in \{+1, -1\}$
- Model
  - Hypothesis class: $\mathcal{H} = \{h | h : \mathbb{R}^d \rightarrow \{+1, -1\}\}$
- Objective: $\widehat{h} = \arg\min_{h \in \mathcal{H}} \frac{1}{N} \sum_{n=1}^{N} \mathbb{I}\left[y^{(n)} \neq h(\mathbf{x}^{(n)})\right]$
- Note
  - Most learning algorithms we consider assume a specific hypothesis space (in boosting, we use “weak learners”).
  - Depending on training data, you will get different learners

Outline

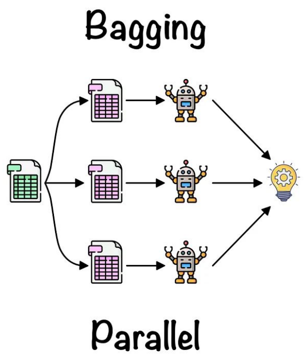
- Decision tree
- Overview of ensemble methods
  - **Bagging**
  - **Boosting**
- AdaBoost



Bagging - Aggregate Bootstrapping

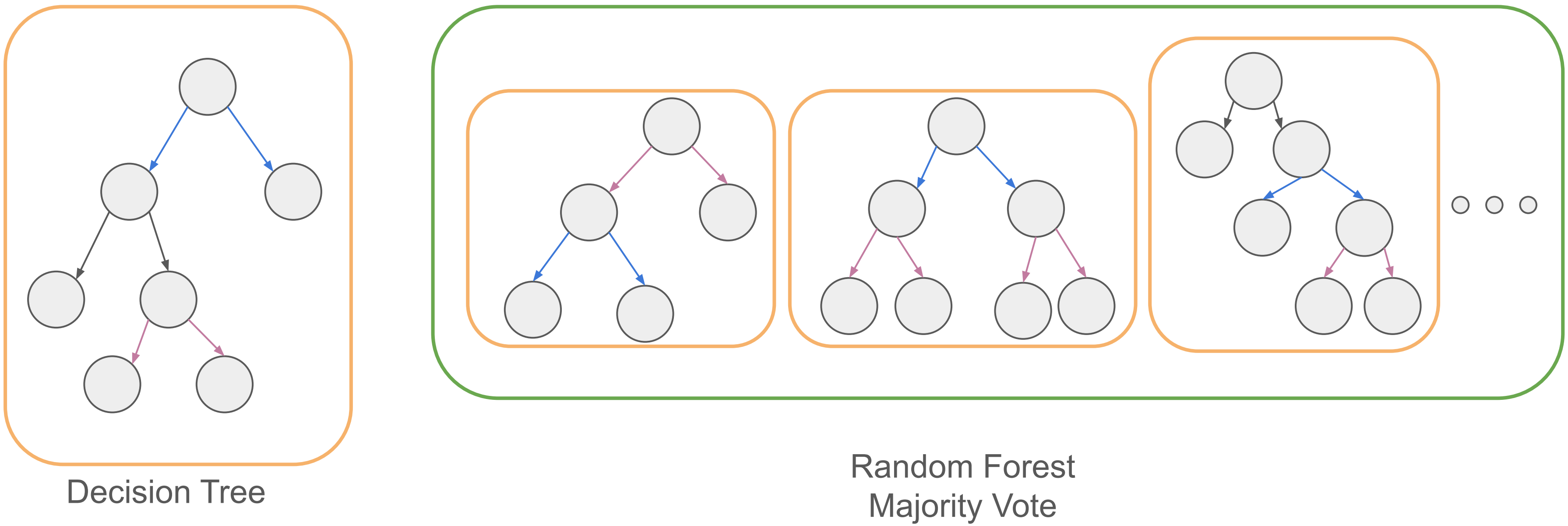
- For $m = 1 \dots M$
  - Draw $n^* < n$ samples from D (training data) with replacement
  - Learn classifier h_m
- Final classifier is a (majority) vote of $h_1 \dots h_M$
  - (Note: For the case of regression, the final ensemble output is just the average of individual regression outputs from M models.)

- Bagging reduces variance (Increases classifier stability)
- However, bagging does not reduce bias.



Random Forest

- Decision Trees: overfitting and generalization issue
  - Decision Trees can be combined with bagging or boosting.
- Random Forest: Ensemble of decision trees with **reduced correlation** among trees
  - For splitting of a (sub-)tree, random subset of features are considered as candidates.
  - After learning multiple decision trees, typically bagging is used for ensembling.



Random Forests

Random Forest is a modified form of bagging that creates ensembles of independent decision trees.

To de-correlate the trees, we:

1. train each tree on a separate bootstrap sample of the full training set (same as in bagging)

2. for each tree, at each split, we ***randomly*** select a set of J' predictors from the full set of predictors.

From amongst the J' predictors, we select the optimal predictor and the optimal corresponding threshold for the split.

Slide credit: Harvard CS109A

Tuning Random Forests

Random forest models have multiple hyper-parameters to tune:

1. the number of predictors to randomly select at each split

2. the total number of trees in the ensemble

3. the maximum depth or minimum leaf node size

In theory, each tree in the random forest is full, but in practice this can be computationally expensive (and added redundancies in the model), thus, imposing a minimum node size or max_depth is not unusual.

Slide credit: Harvard CS109A

Out of bag error: estimating generalization error

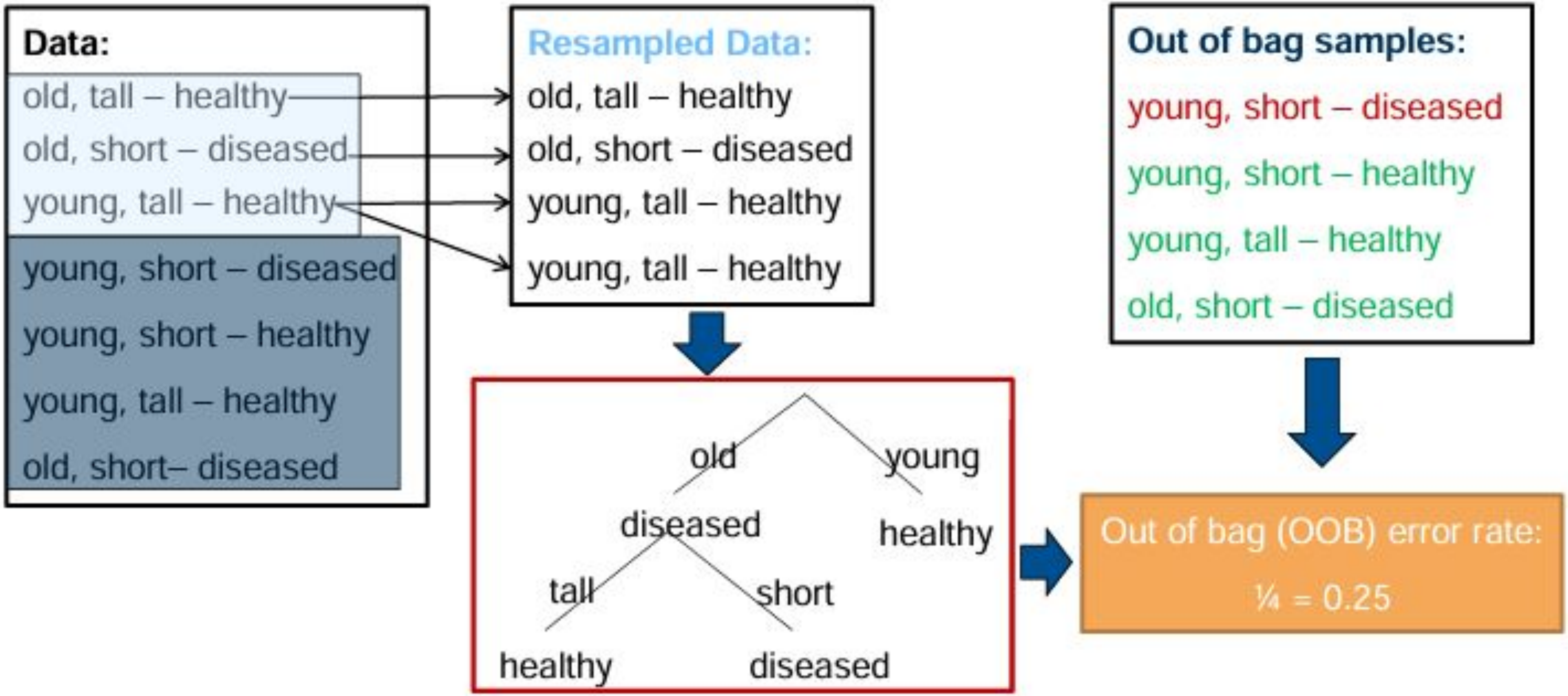


Image credit: https://stat.ethz.ch/education/semesters/ss2012/ams/slides/v10.2.pdf

Out of bag error: estimating generalization error

- Given a random forest, OOB can be computed over aggregation of multiple trees
    - For example, if a data point is used for training tree 1 but not for tree 2, tree 3 and tree 4, its OOB prediction is a majority vote over tree 2, tree 3 and tree 4.
- Advantage: provides meaningful generalization estimation while all data points can be used as training data for the random forest (i.e. no held-out set needed).

Outline

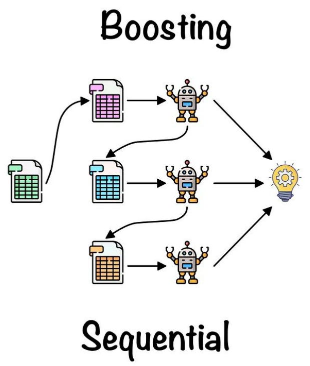
- Decision tree
- Overview of ensemble methods
    - Bagging
    - **Boosting**
- AdaBoost

Boosting: overview

- Boosting is a way to effectively combining multiple weak learners to construct a strong learner (e.g., classifier).

- Finding many weak rules of thumb is easier than finding a single, highly accurate prediction rule

- Use “feedbacks” from previously learned weak learners to inform training of next weak learners.



Boosting (Schapire 1989)

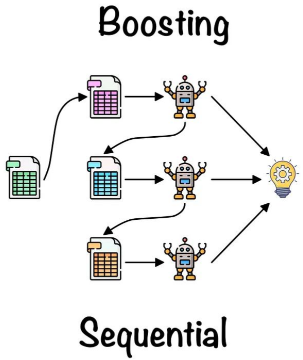
- Randomly select $n_1 < n$ samples from D to obtain D_1
- Train weak learner h_1
- Select $n_2 < n$ samples from D with half of the samples misclassified by h_1 to obtain D_2
- Train weak learner h_2
- Select all samples from D that h_1 and h_2 disagree on
- Train weak learner h_3
- Final classifier is vote of weak learners

Outline

- Decision tree
- Overview of ensemble methods
  - Bagging
  - Boosting
- **AdaBoost**

Adaboost - Adaptive Boosting

- Instead of sampling training data, re-weight the training data
  - Previous weak learner has only >50% accuracy over new distribution
- Can be used to learn weak classifiers (e.g. decision trees)
  - Strong classifier like neural network may overfit
- Final classification based on weighted vote of weak classifiers



What’s Good about Adaboost

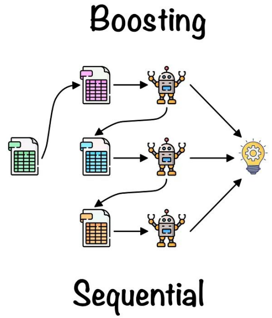
- Improves classification accuracy
- Can be used with many different (weak) classifiers
- Commonly used in many areas
- Simple to implement
- Robust to overfitting

Learning framework

- Data: $\{(\mathbf{x}^{(n)}, y^{(n)}) : n = 1, \ldots, N\}$ where $\mathbf{x}^{(n)} \in \mathbb{R}^d, y \in \{+1, -1\}$
- Model
  - Hypothesis class: $\mathcal{H} = \{h | h : \mathbb{R}^d \rightarrow \{+1, -1\}\}$
  - Objective: $\widehat{h} = \arg\min_{h \in \mathcal{H}} \frac{1}{N} \sum_{n=1}^{N} \mathbb{I}\left[y^{(n)} \neq h(x^{(n)})\right]$
- Goal: with many learners $h_1, \ldots, h_M$, combine them to get a strong classifier

$$h(\mathbf{x}) = \operatorname{sign}\left(\sum_{m}^{M} \alpha_m h_m(\mathbf{x})\right)$$

AdaBoost: Illustration



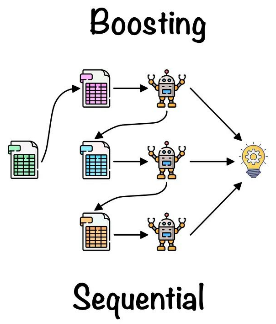
Weighted examples

$\{w_1^{(n)}\}$

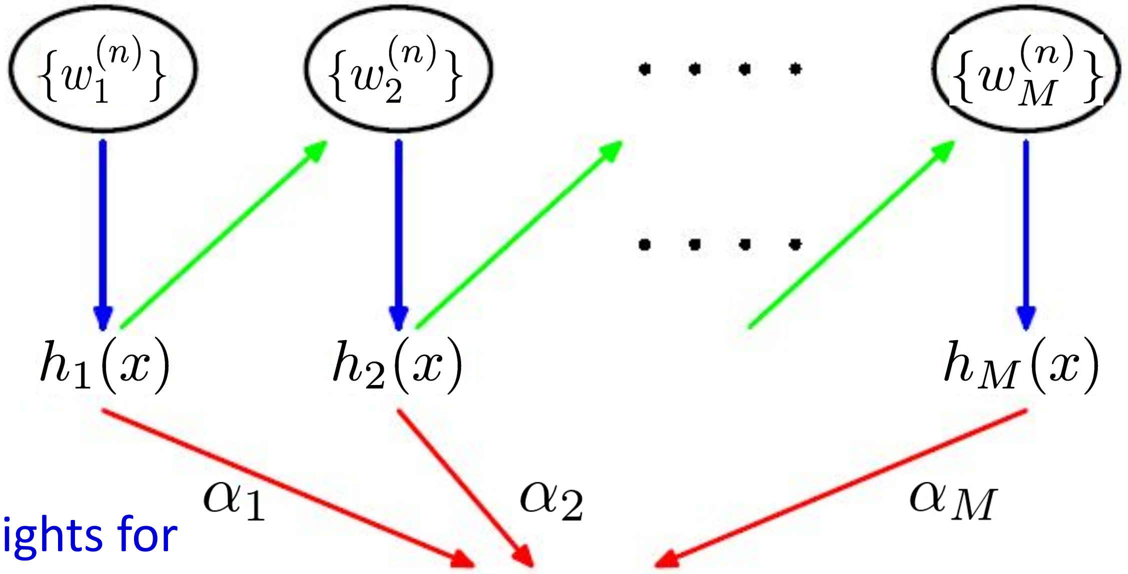
Weak learners

Final classifier

AdaBoost: Illustration



Weighted examples



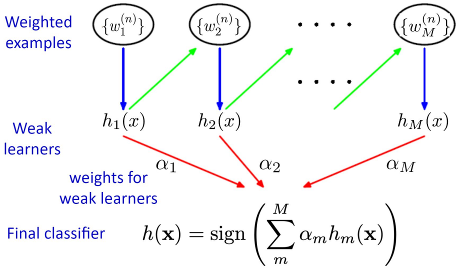
Weak learners

weights for weak learners

Final classifier

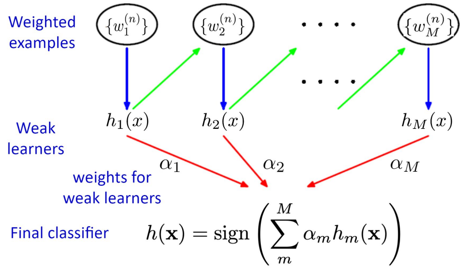
$$h(\mathbf{x}) = \operatorname{sign}\left(\sum_{m}^{M} \alpha_m h_m(\mathbf{x})\right)$$

AdaBoost Algorithm



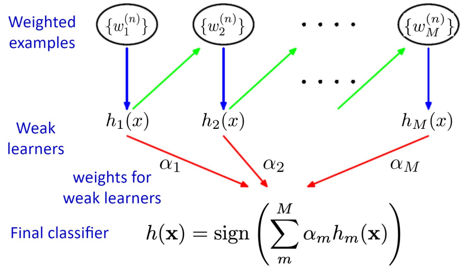
- Initialize data weights: $w_1^{(n)} = 1/N$ $\quad n = 1, \ldots, N.$

AdaBoost Algorithm



- Initialize data weights: $w_1^{(n)} = 1/N$ $\quad n = 1, \ldots, N.$
- For *m*=1,…,*M* (m^{th} boosting round)
  - Train a classifier $J_m = \sum_{n=1}^{N} w_m^{(n)} \mathbb{I}\left(h_m\left(\mathbf{x}^{(n)}\right) \neq y^{(n)}\right)$ to minimize weighted error function

AdaBoost Algorithm

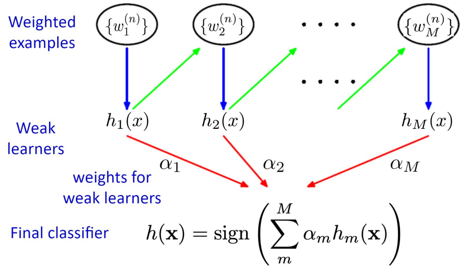


- Initialize data weights: $w_1^{(n)} = 1/N$ $\quad n = 1, \ldots, N.$
- For m=1,…,M (m^{th} boosting round)
  - Train a classifier $J_m = \sum_{n=1}^{N} w_m^{(n)} \mathbb{I}\left(h_m\left(\mathbf{x}^{(n)}\right) \neq y^{(n)}\right)$ to minimize weighted error function

$$\epsilon_m = \frac{\sum_{n=1}^N w_m^{(n)} I\left(h_m\left(\mathbf{x}^{(n)}\right) \neq y^{(n)}\right)}{\sum_{n=1}^N w_m^{(n)}} \qquad \alpha_m = \ln\left\{\frac{1-\epsilon_m}{\epsilon_m}\right\}.$$

  - Evaluate weighted error and set the <u>classifier weight</u>

AdaBoost Algorithm



- Initialize data weights: $w_1^{(n)} = 1/N$ $\quad n = 1, \ldots, N.$
- For *m*=1,…,*M* (*m*th boosting round)
  - Train a classifier $J_m = \sum_{n=1}^{N} w_m^{(n)} \mathbb{I}\left(h_m\left(\mathbf{x}^{(n)}\right) \neq y^{(n)}\right)$ to minimize weighted error function

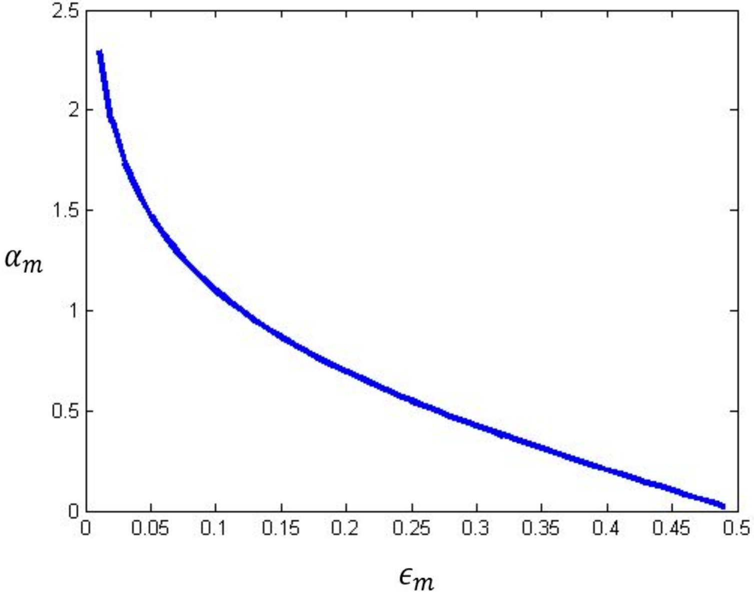
$$\epsilon_m = \frac{\sum_{n=1}^{N} w_m^{(n)} I\left(h_m\left(\mathbf{x}^{(n)}\right) \neq y^{(n)}\right)}{\sum_{n=1}^{N} w_m^{(n)}} \qquad \alpha_m = \ln\left\{\frac{1-\epsilon_m}{\epsilon_m}\right\}.$$

  - Evaluate weighted error and set the classifier weight
  - update data weights $w_{m+1}^{(n)} = w_m^{(n)} \exp\left\{\alpha_m I\left(h_m\left(\mathbf{x}^{(n)}\right) \neq y^{(n)}\right)\right\}$
- Make final prediction

$$h(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})\right)$$

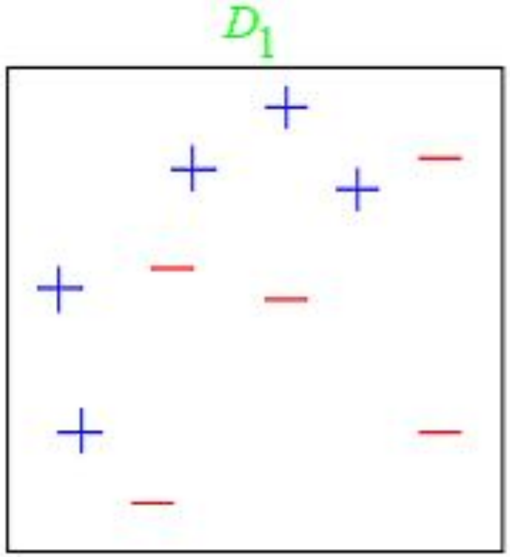
Relationship between weighted error and classifier weights

$$\alpha_m = \ln \left\{ \frac{1-\epsilon_m}{\epsilon_m} \right\}$$

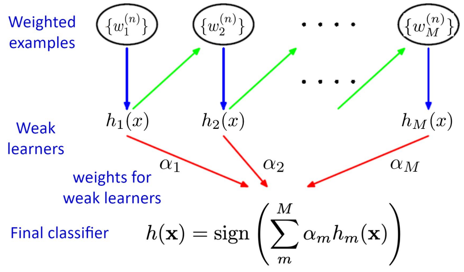


Example

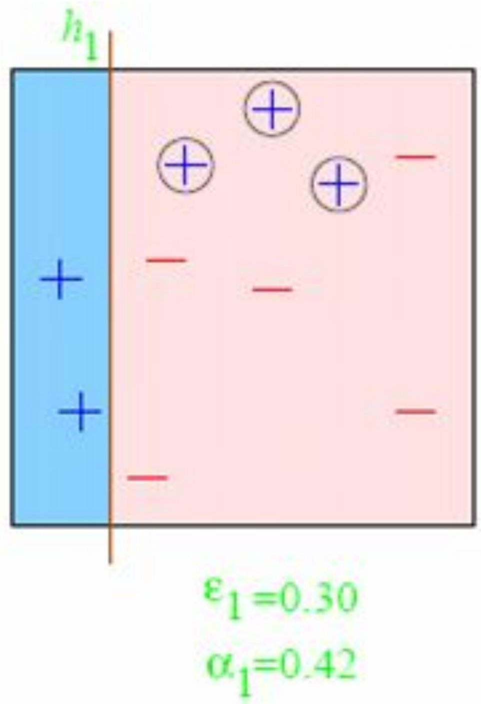
- Assume that the weak classifiers are decision stumps (vertical or horizontal half-planes):



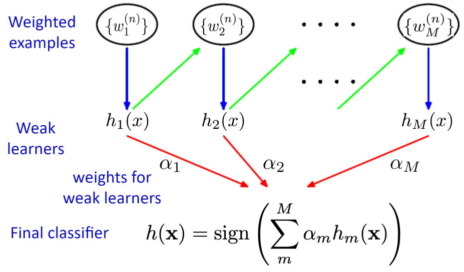
Example



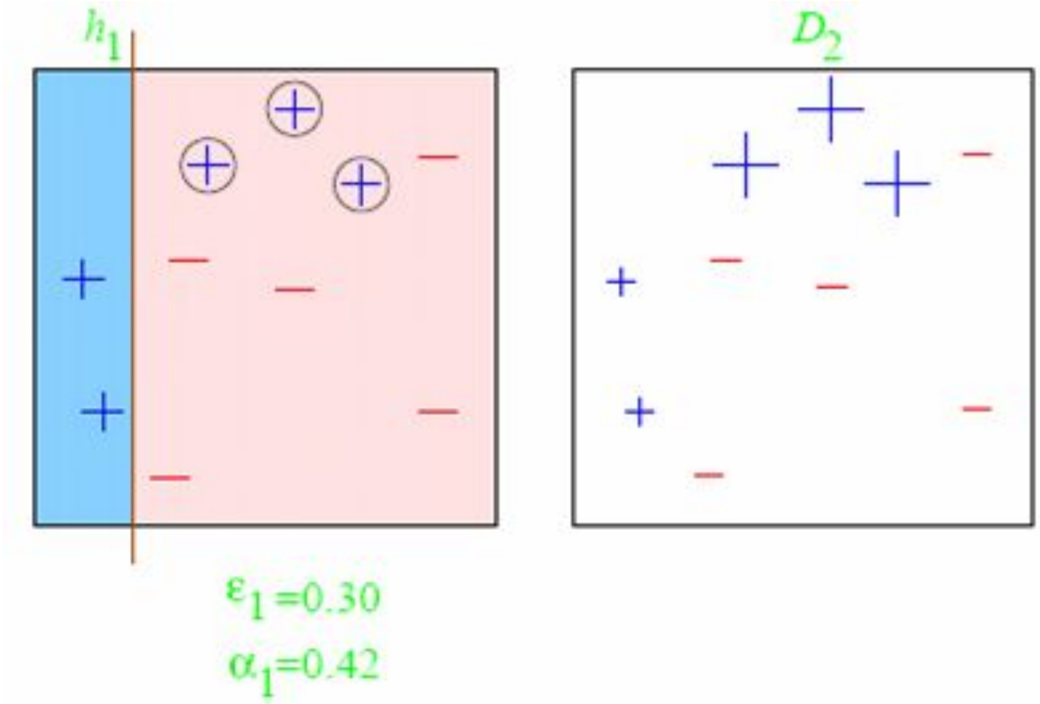
- The first round



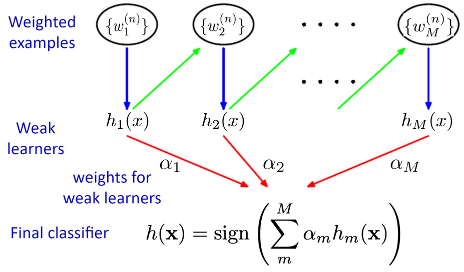
Example



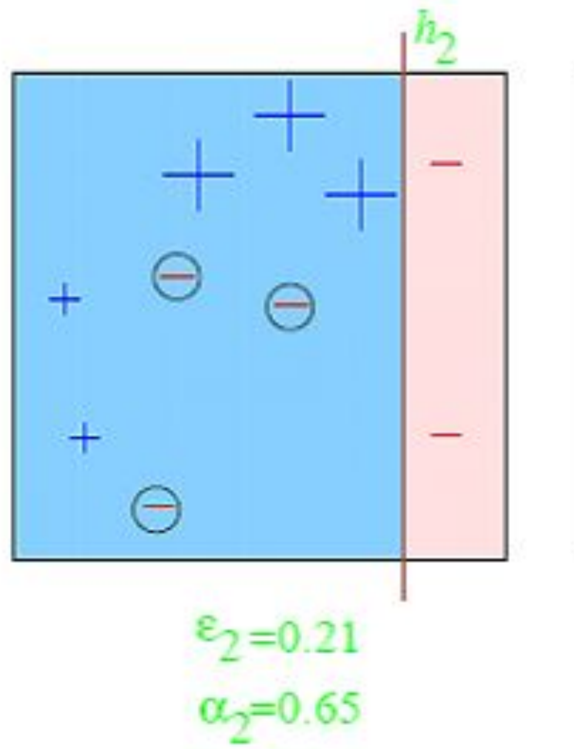
- The first round



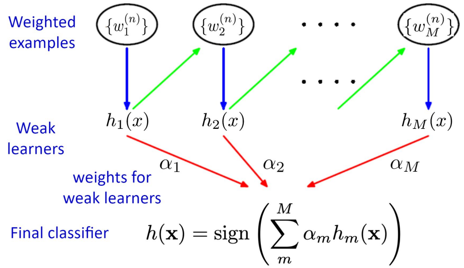
Example



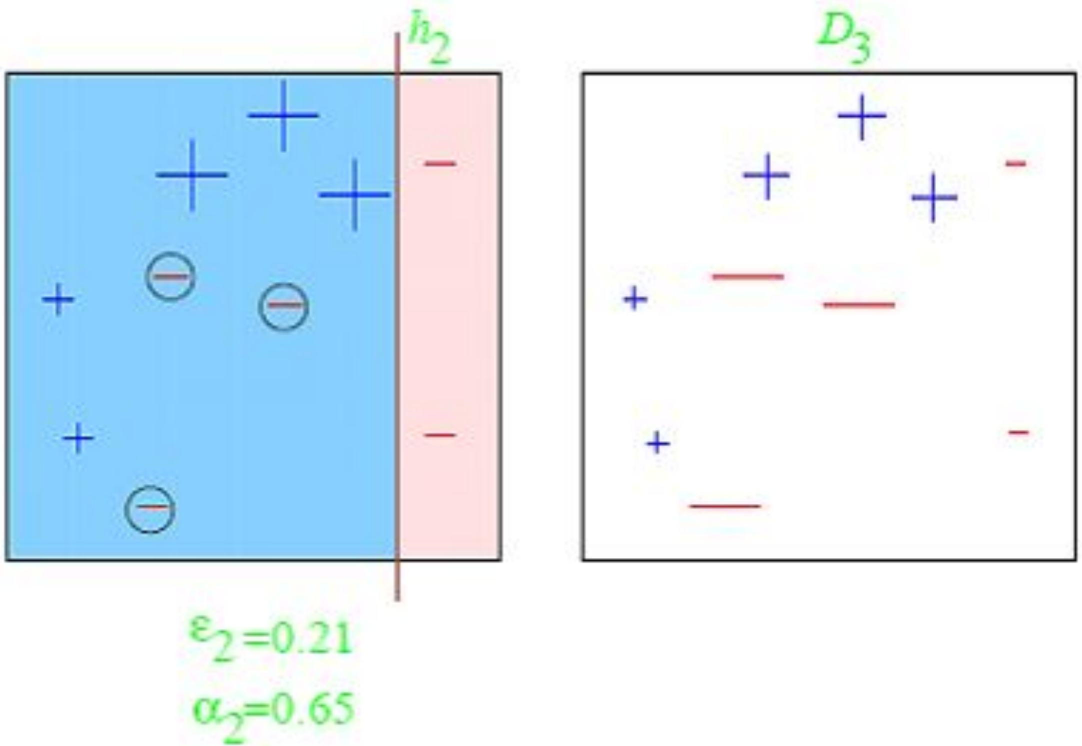
- The second round



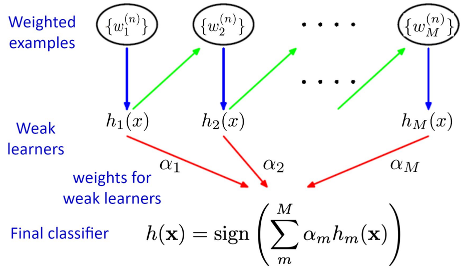
Example



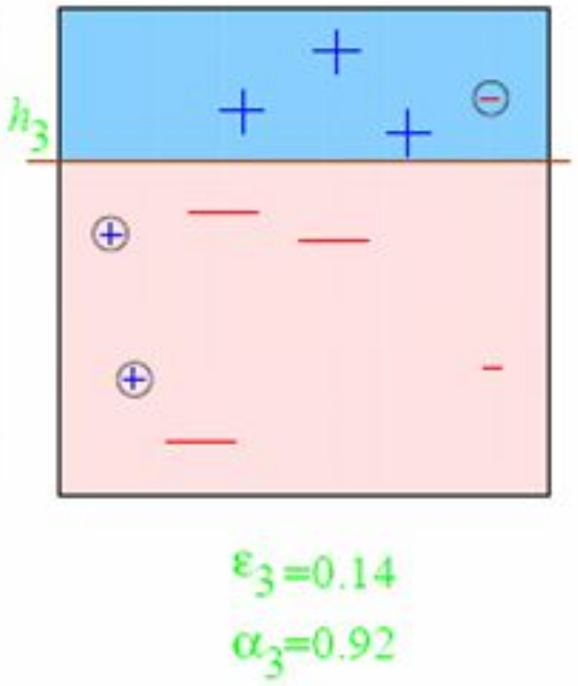
- The second round



Example

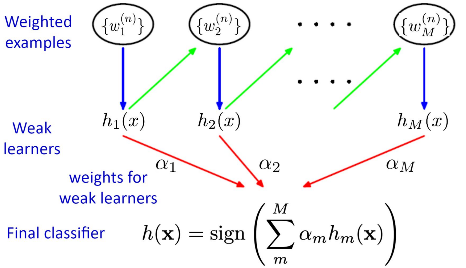


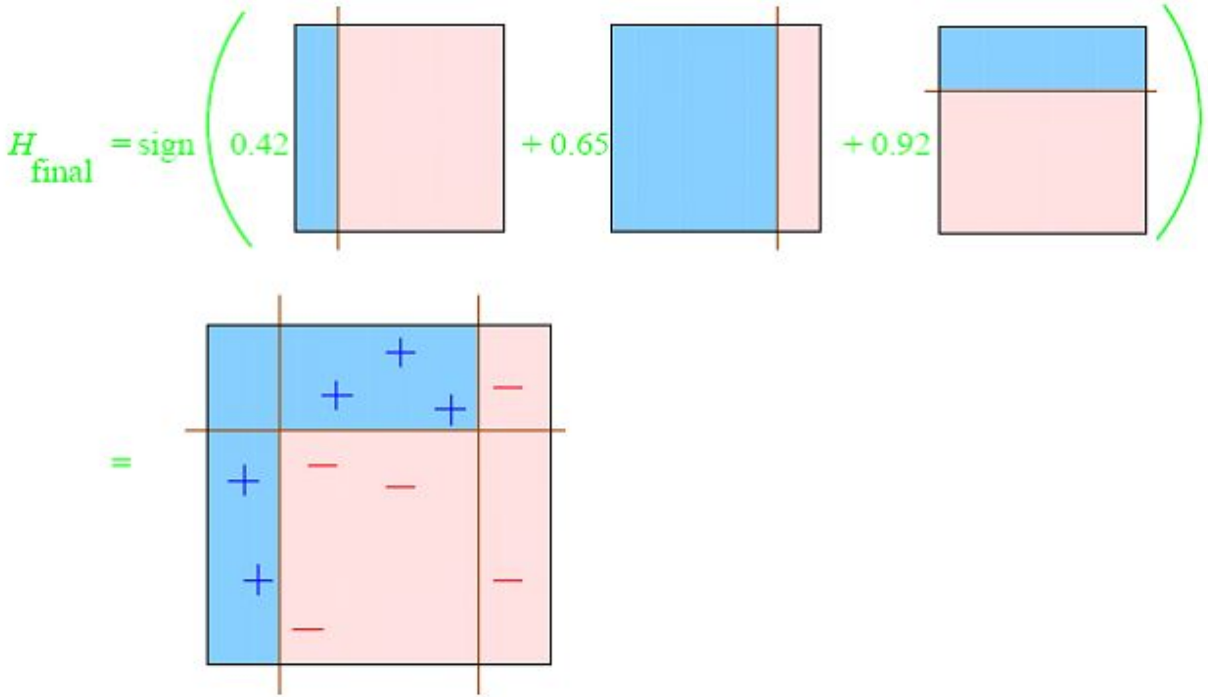
- The third round

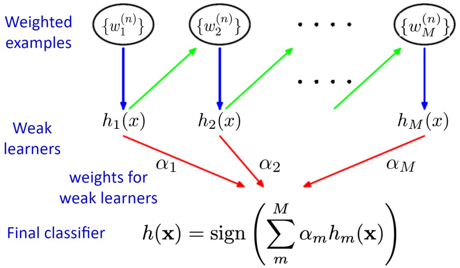


Example

- Final classifier (combining the weak learners)

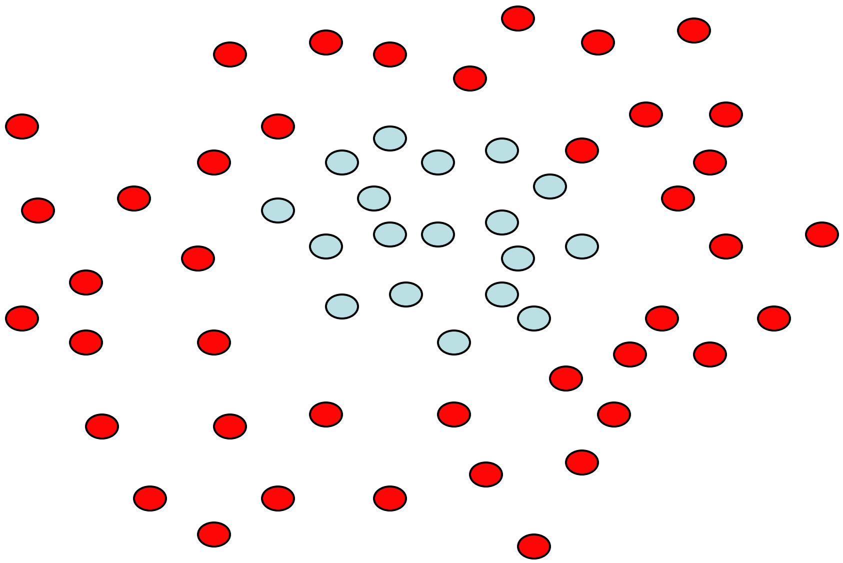






Another example

- Boosting is a sequential procedure:



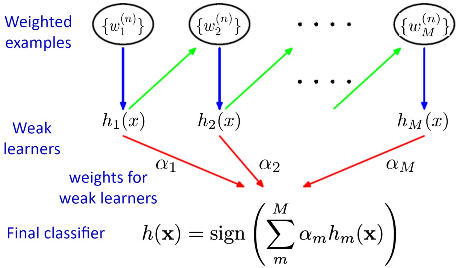
Each data point has a class label:

$$y^{(i)} = \begin{cases} +1 & (\ ) \\ -1 & (\ ) \end{cases}$$

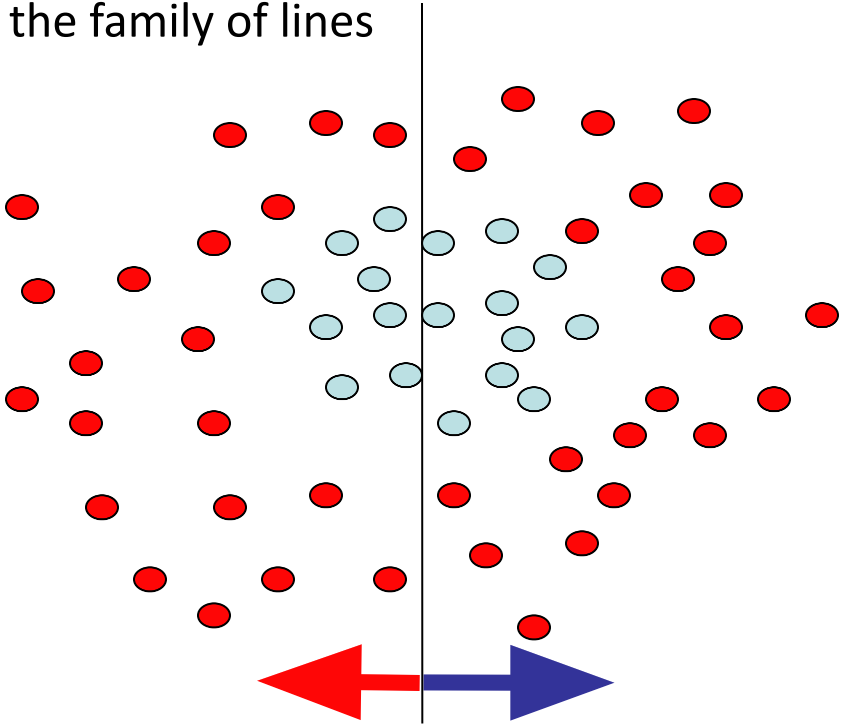
and a weight:

$$w^{(i)} = \frac{1}{N}$$

Another example



Weak learners from the family of lines



Each data point has

a class label:

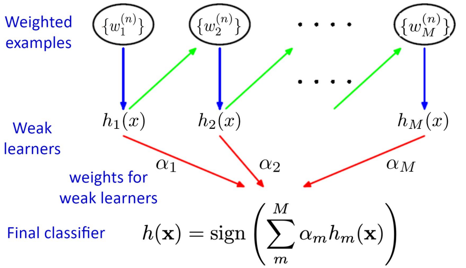
$$y^{(i)} = \begin{cases} +1 & (\bullet) \\ -1 & (\circ) \end{cases}$$

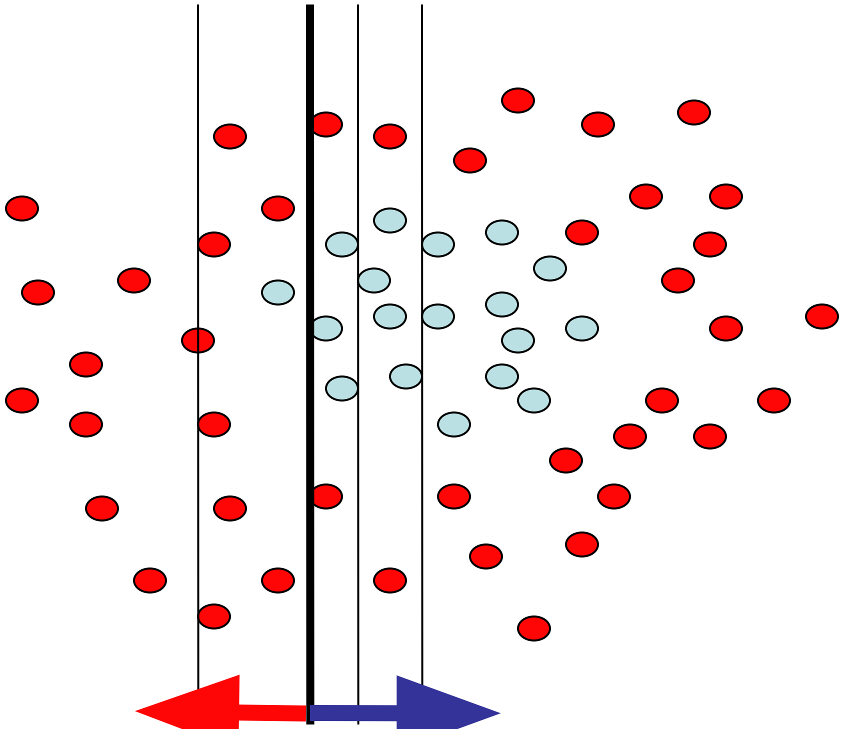
and a weight:

$$w^{(i)} = \frac{1}{N}$$

h => p(error) = 0.5 it is at chance

Another example





Each data point has a class label:

$$y^{(i)} = \begin{cases} +1 & (\;) \\ -1 & (\;) \end{cases}$$

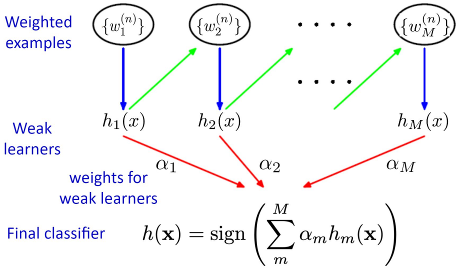
and a weight:

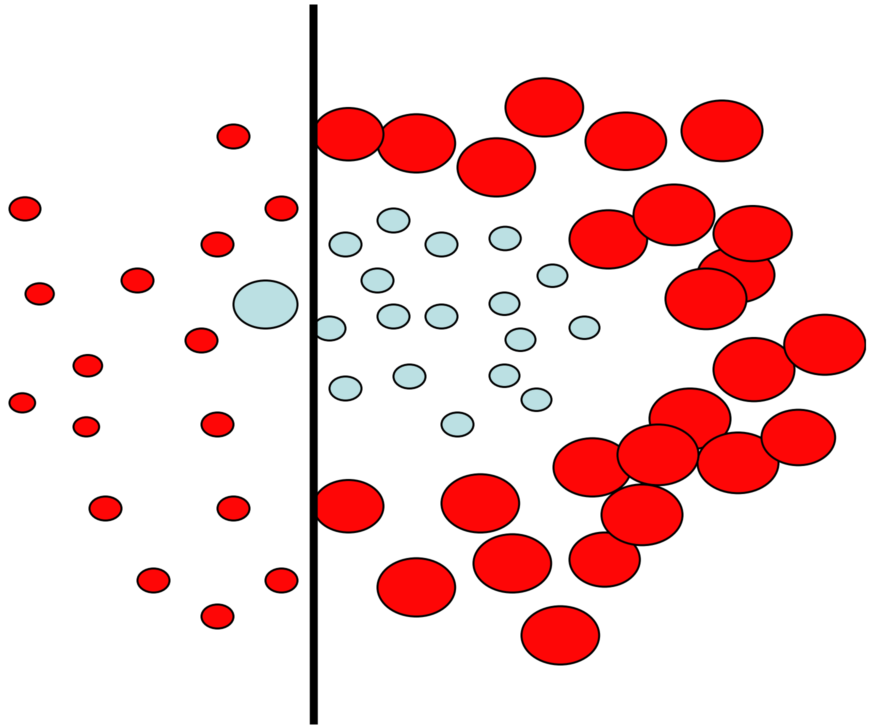
$$w^{(i)} = \frac{1}{N}$$

This one seems to be the best

This is a ‘**weak classifier**’: It performs slightly better than chance.

Another example





Each data point has

a class label:

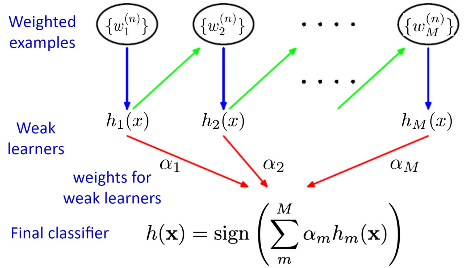
$$y^{(i)} = \begin{cases} +1 & (\quad) \\ -1 & (\quad) \end{cases}$$

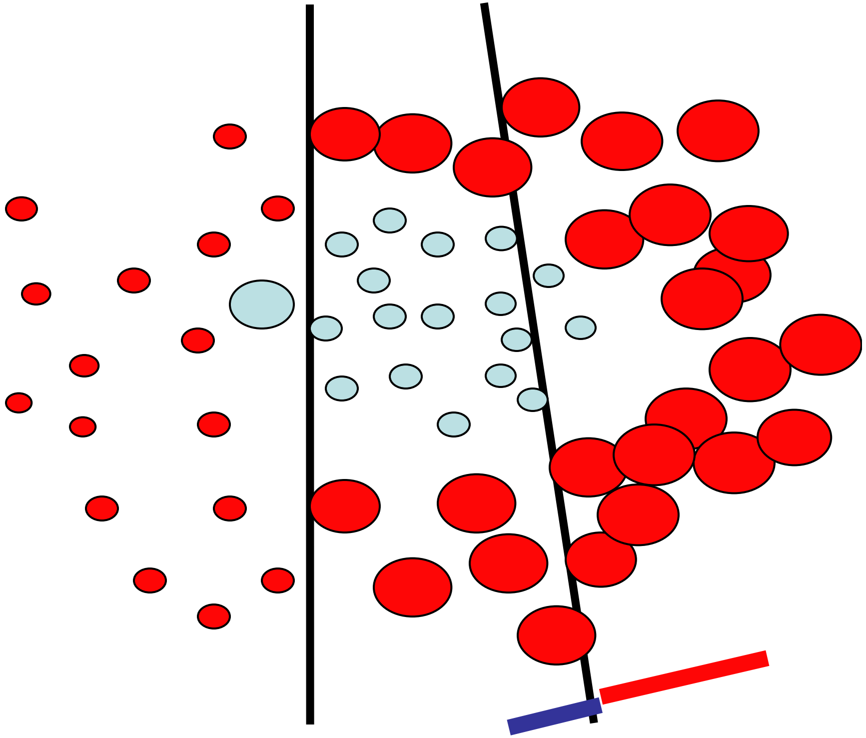
We update the weights:

$$w^{(i)} \leftarrow w^{(i)} \exp\{-y^{(n)} h_m\}$$

We set a new problem for which the previous weak classifier performs at chance again

Another example





Each data point has

a class label:

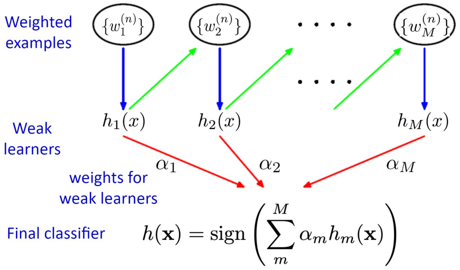
$$y^{(i)} = \begin{cases} +1 & (\bullet) \\ -1 & (\circ) \end{cases}$$

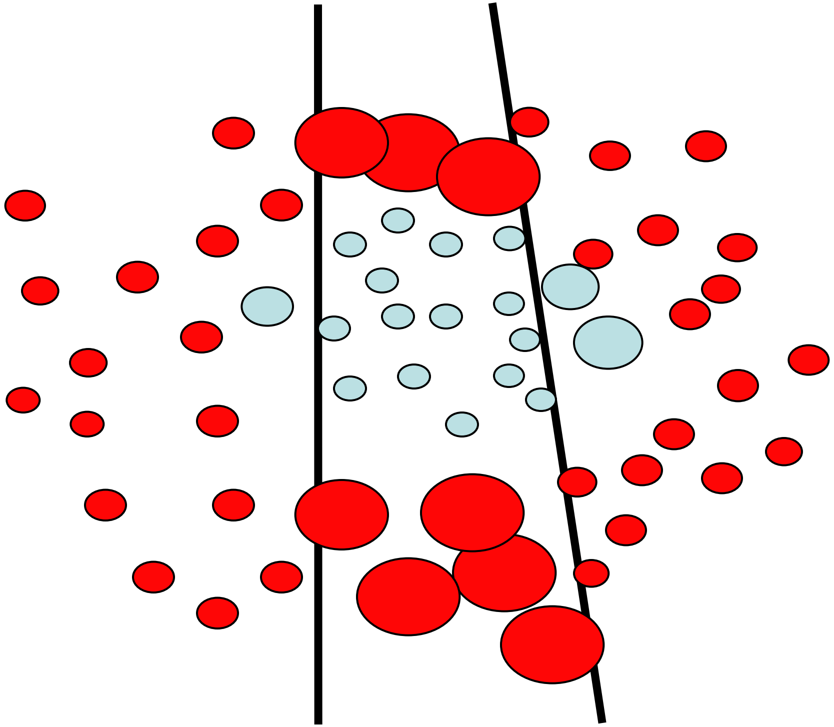
We update the weights:

$$w^{(i)} \leftarrow w^{(i)} \exp\{-y^{(n)} h_m\}$$

We set a new problem for which the previous weak classifier performs at chance again

Another example





Each data point has

a class label:

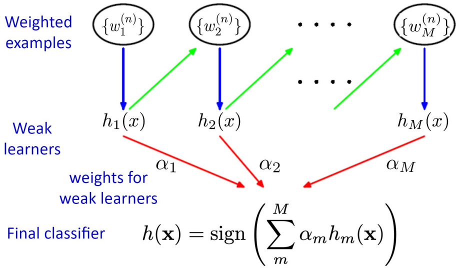
$$y^{(i)} = \begin{cases} +1 & (\bullet) \\ -1 & (\circ) \end{cases}$$

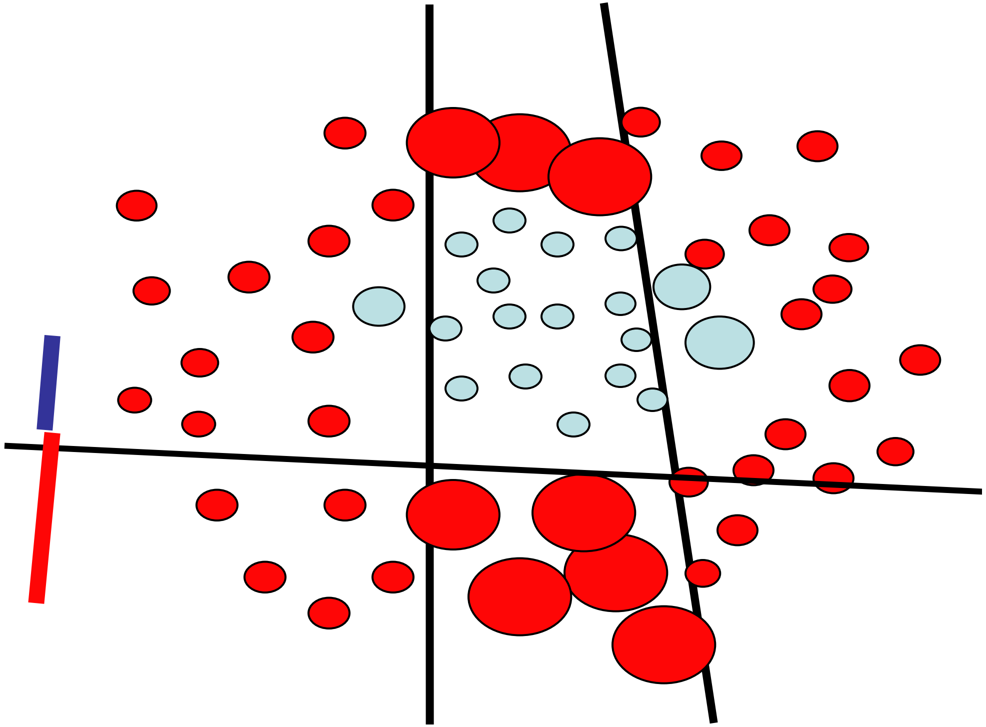
We update the weights:

$$w^{(i)} \leftarrow w^{(i)} \exp\{-y^{(n)} h_m\}$$

We set a new problem for which the previous weak classifier performs at chance again

Another example





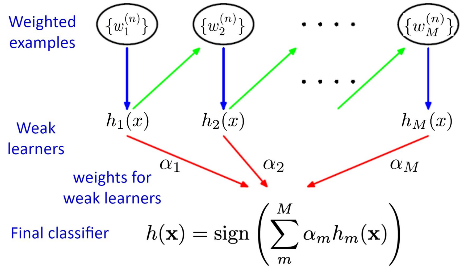
Each data point has a class label:

$$y^{(i)} = \begin{cases} +1 & (\bullet) \\ -1 & (\circ) \end{cases}$$

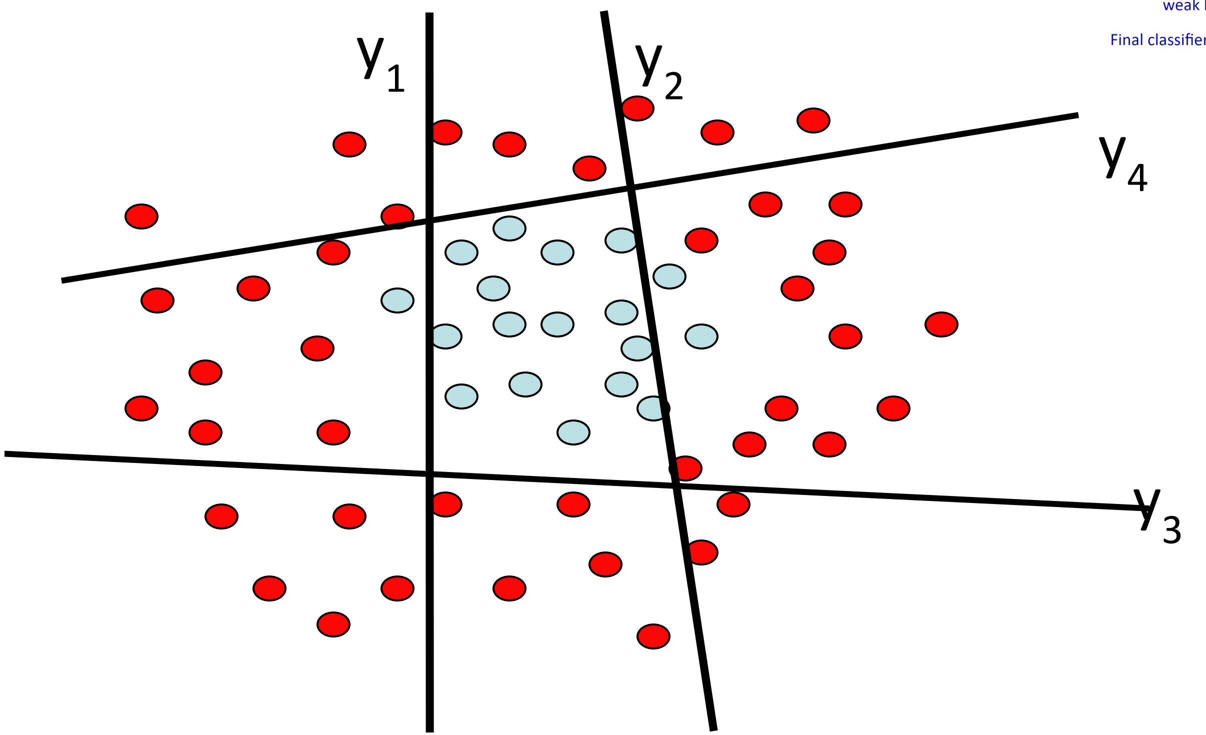
We update the weights:

$$w^{(i)} \leftarrow w^{(i)} \exp\{-y^{(n)} h_m\}$$

We set a new problem for which the previous weak classifier performs at chance again



Another example



The strong (non- linear) classifier is built as the combination of all the weak (linear) classifiers.

Minimizing exponential error

- Interpretation by Friedman et al. (2000): boosting is a sequential minimization of exponential error function

$$\mathcal{J} = \sum_{n=1}^{N} \exp\left\{-y^{(n)} f_m\left(\mathbf{x}^{(n)}\right)\right\}$$

  - Where

$$f_m(\mathbf{x}) = \frac{1}{2}\sum_{l=1}^{m} \alpha_l h_l(\mathbf{x})$$

Minimizing exponential error

- Consider a sequential optimization: given base classifiers $h_1(\mathbf{x}), \ldots, h_{m-1}(\mathbf{x})$ and their weights $\alpha_1, \ldots, \alpha_{m-1}$, we can write *J* as:

$$\begin{aligned}
\mathcal{J} &= \sum_{n=1}^{N} \exp\left\{-y^{(n)} f_{m-1}\left(\mathbf{x}^{(n)}\right) - \frac{1}{2} y^{(n)} \alpha_m h_m\left(\mathbf{x}^{(n)}\right)\right\} \\
&= \sum_{n=1}^{N} w_m^{(n)} \exp\left\{-\frac{1}{2} y^{(n)} \alpha_m h_m\left(\mathbf{x}^{(n)}\right)\right\}
\end{aligned}$$

$$w_m^{(n)} = \exp\left\{-y^{(n)} f_{m-1}\left(\mathbf{x}^{(n)}\right)\right\}$$

Minimizing exponential error

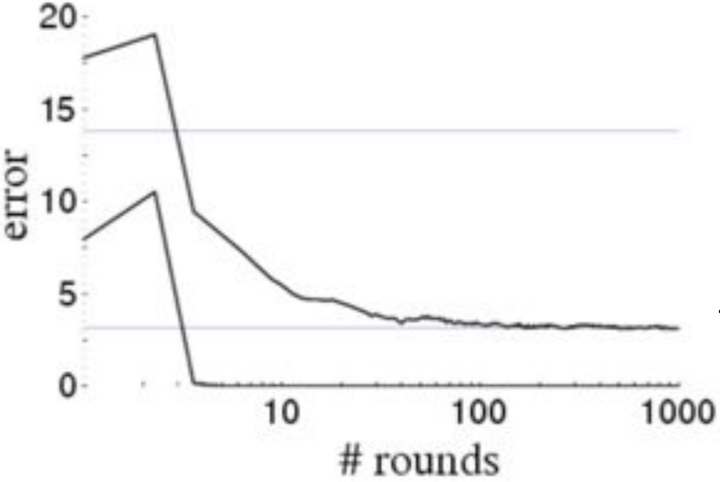
$$
\begin{aligned}
f_m(\mathbf{x}) &= \frac{1}{2}\sum_{l=1}^{m} \alpha_l h_l(\mathbf{x}) \\
w_m^{(n)} &= \exp\left\{-y^{(n)} f_{m-1}(\mathbf{x}^{(n)})\right\} \\
w_{m+1}^{(n)} &= w_m^{(n)} \exp\left\{-\frac{1}{2} y^{(n)} \alpha_m h_m(\mathbf{x}^{(n)})\right\} \\
y^{(n)} h_m(\mathbf{x}^{(n)}) &= 1 - 2\mathbb{I}\left[h_m(\mathbf{x}^{(n)}) \neq y^{(n)}\right] \\
w_{m+1}^{(n)} &= w_m^{(n)} \exp\left\{-\frac{\alpha_m}{2}\right\} \exp\left\{\alpha_m \mathbb{I}\left[h_m(\mathbf{x}^{(n)}) \neq y^{(n)}\right]\right\} \\
&\propto w_m^{(n)} \exp\left\{\alpha_m \mathbb{I}\left[h_m(\mathbf{x}^{(n)}) \neq y^{(n)}\right]\right\}
\end{aligned}
$$

Key idea:

- per-sample weight is increased (amplified) if $h_m()$ **incorrectly** classifies example $x^{(n)}$
- per-sample weight is decreased (de-amplified) if $h_m()$ **correctly** classifies example $x^{(n)}$

Example runs of Adaboost

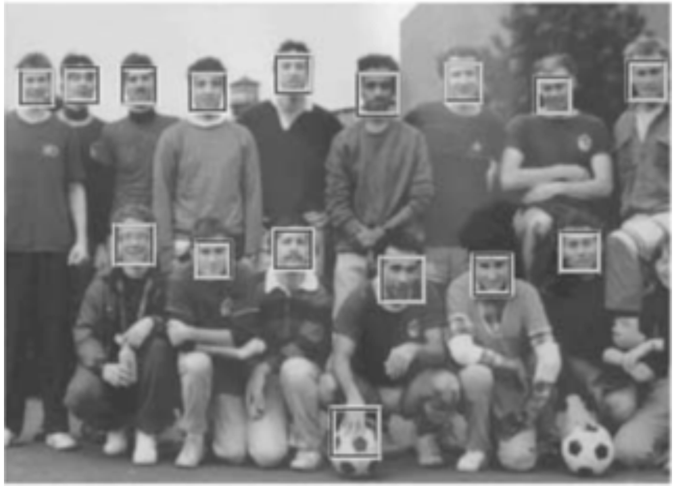
- Training error rapidly decreases as the number of boosting rounds increases.
- # boosting rounds are chosen by cross validation.



Test error
Training error

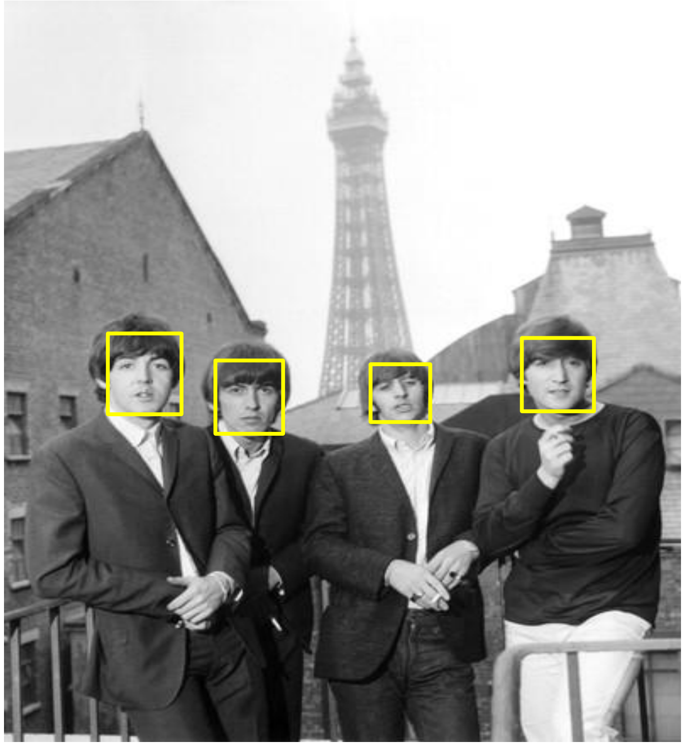
Face detection with boosting

- Viola-Jones face detector (2001)

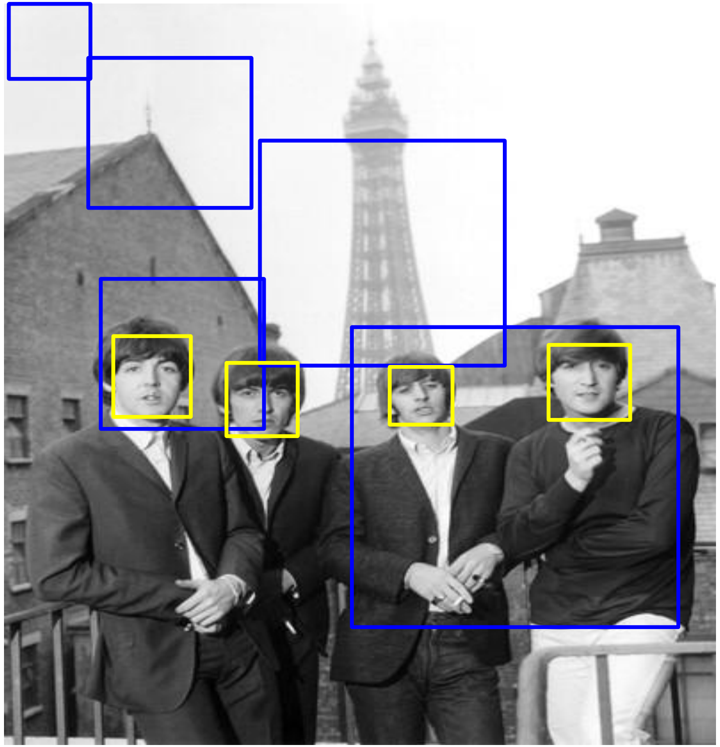


Face detection with boosting

- Viola-Jones face detector



Sliding Windows



1. hypothesize:

try all possible rectangle locations, sizes

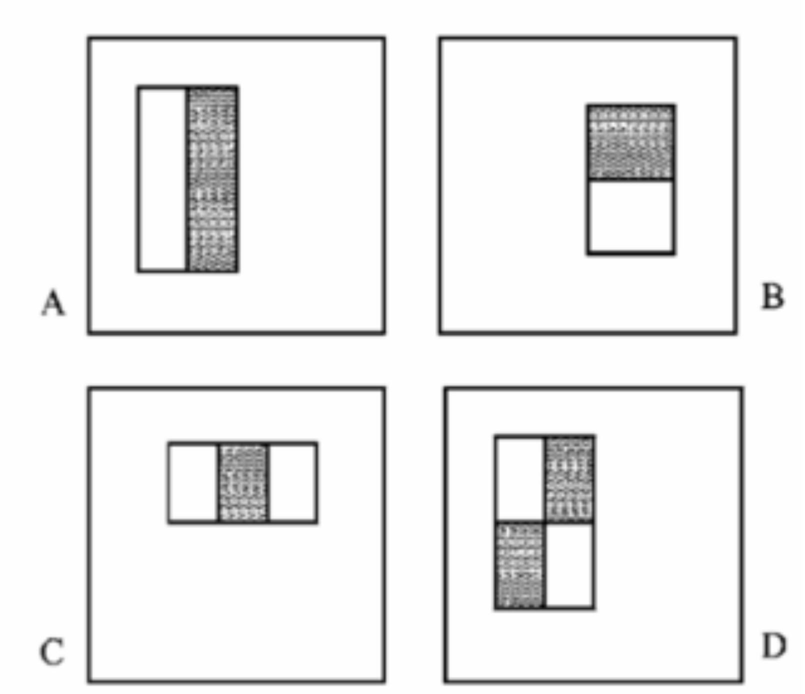
2. test:

classify if rectangle contains a face (and only the face)

Note: 1000's more false windows then true ones.

Face detection with boosting

- The classifier is based on boosting, and it uses Haar wavelet features



Face detection with boosting

- The algorithm proceeds with learning a simple detector with specific Haar wavelet

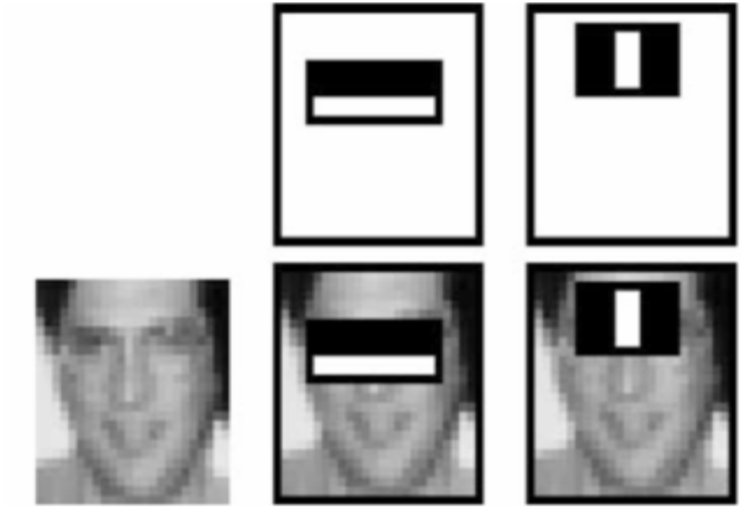
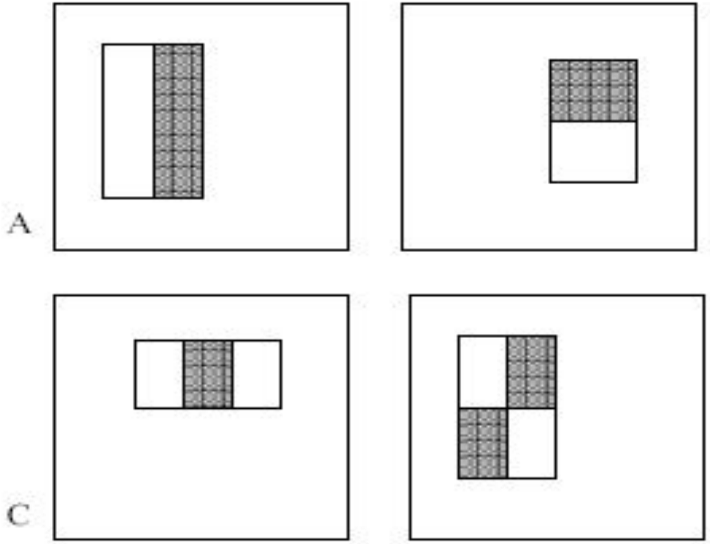


Image Features

- The algorithm proceeds with learning a simple detector with specific Haar wavelets

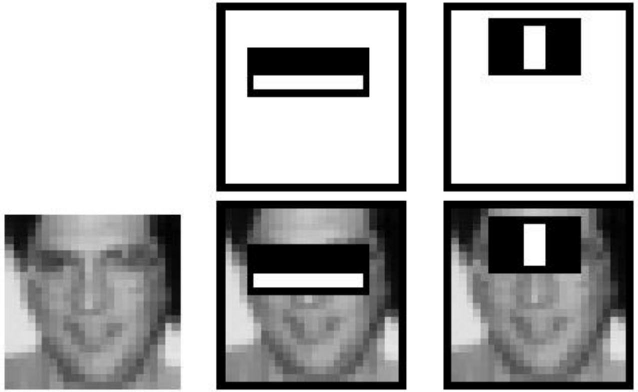


4 Types of “Rectangle filters”

Based on 24x24 grid:
160,000 features to choose from

$$g(\mathbf{x}) = \mathrm{sum(WhiteArea)}\ \text{-}\ \mathrm{sum(BlackArea)}$$

Image Features



$$F(\mathbf{x}) = \alpha_1 h_1(\mathbf{x}) + \alpha_2 h_2(\mathbf{x}) + \dots$$

$$h_i(\mathbf{x}) = \begin{cases} 1 & \text{if } g_i(\mathbf{x}) > \theta_i \\ -1 & \text{otherwise} \end{cases}$$

Need to: (1) Select Features i = 1, …, n

(2) Learn thresholds θ_i

(3) Learn weights α_i

Face detection with boosting

- Defines a classifier using an additive model:

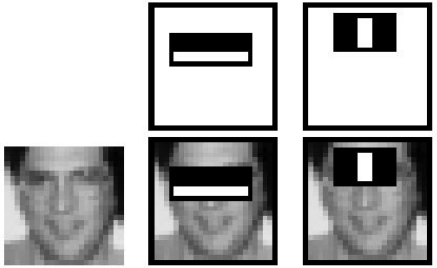
$$f(x) = \alpha_1 h_1(x) + \alpha_2 h_2(x) + ...$$

Strong classifier

Features vector

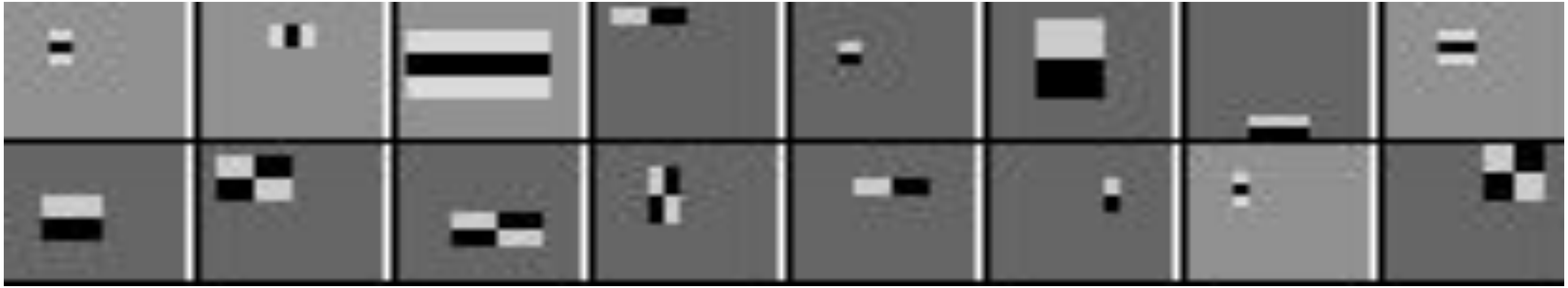
Weight

Weak classifier



$$F(\mathbf{x}) = \alpha_1 y_1(\mathbf{x}) + \alpha_2 y_2(\mathbf{x}) + \ldots$$

A peek ahead: the learned features

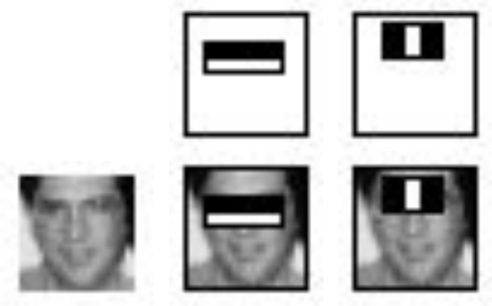


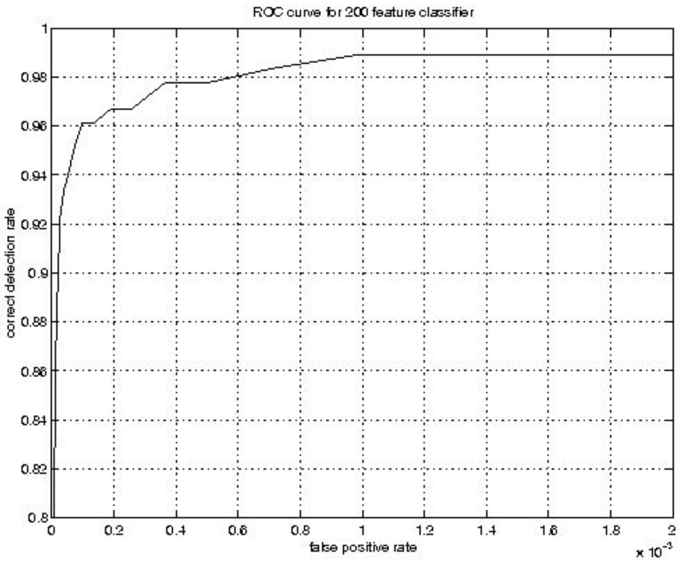
Example Classifier for Face Detection

A classifier with 200 rectangle features was learned using AdaBoost

95% correct detection on test set with 1 in 14084 false positives.

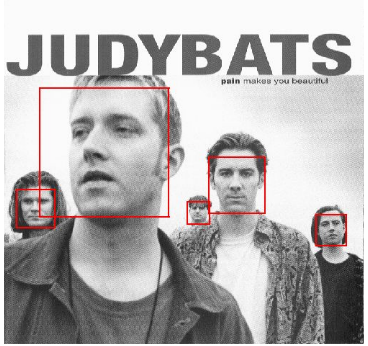
Remark: In Viola-Jones paper, a variant boosting algorithm called "cascaded classifier" is being used.

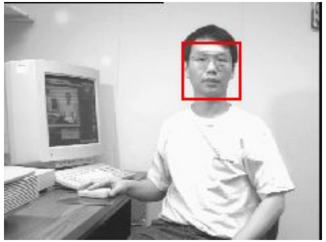


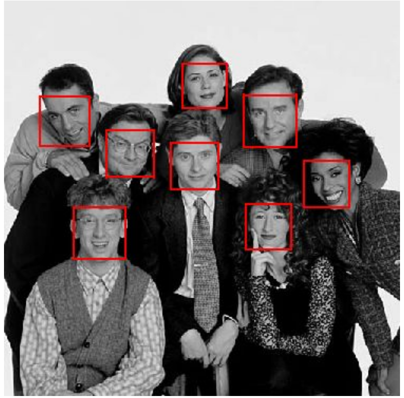


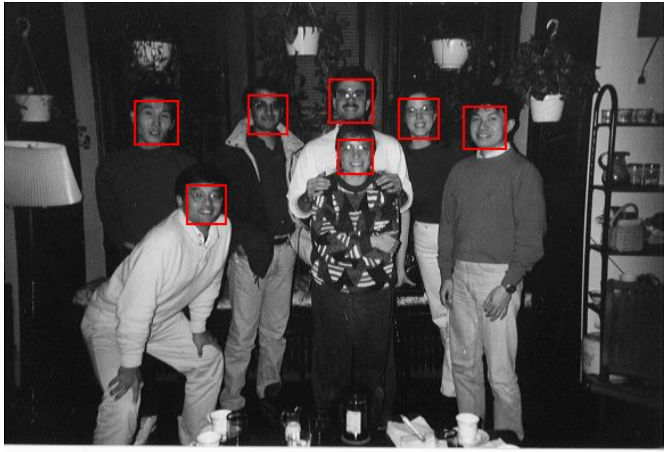
ROC curve for 200 feature classifier

Output of Face Detector on Test Images



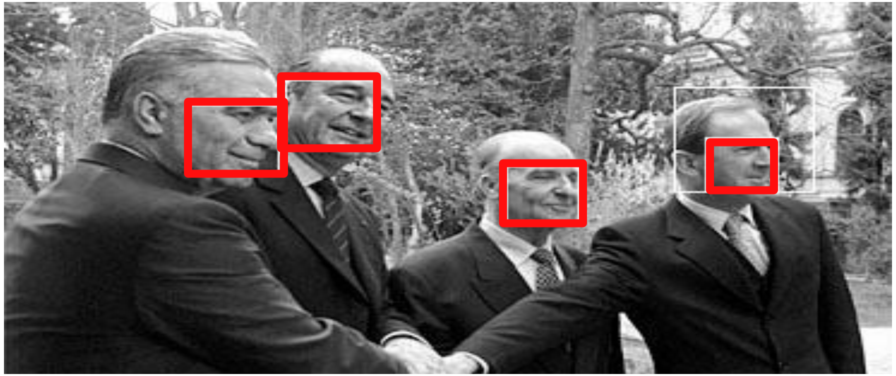






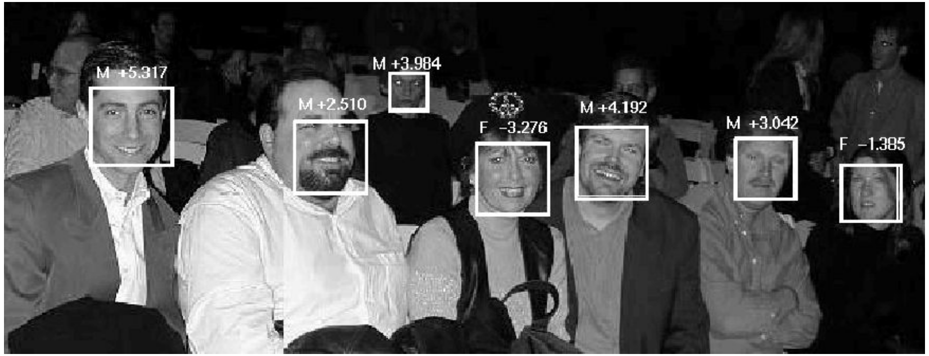
Solving other “Face” Tasks

Facial Feature Localization



Profile Detection

Demographic Analysis



Any feedback (about lecture, slide, homework, project, etc.)?

(via **anonymous** google form: https://forms.gle/fpYmiBtG9Me5qbP37)

Change Log of lecture slides:
https://docs.google.com/document/d/e/2PACX-1vSSIHJjkIypK7rKFSR1-5GYXyBCEW8UPtpSfCR9AR6M1l7K9ZQEmxfFwaWaW7kLDxusthsF8WlCyZJ-/pub