

EECS 545: Machine Learning

Lecture 26. Decision Tree and Ensemble Methods

Honglak Lee and Michał Dereziński
4/13/2022



Logistics

- Reminder to email us at eeecs545.w22@gmail.com if you have conflicts with an in-person poster presentation on April 21st
 - Note grades will be assigned by group, so as long as someone from the group is able to present you will be fine
- 2-3 presentation blocks from 9am-4pm (exact times TBD)
- Posters can be printed off following instructions from ITS:
<https://its.umich.edu/computing/computers-software/campus-computing-sites/printing/poster-printing>
 - Note that printing takes up to 24 hours, so plan on having your posters done more than 24 hour before presentation

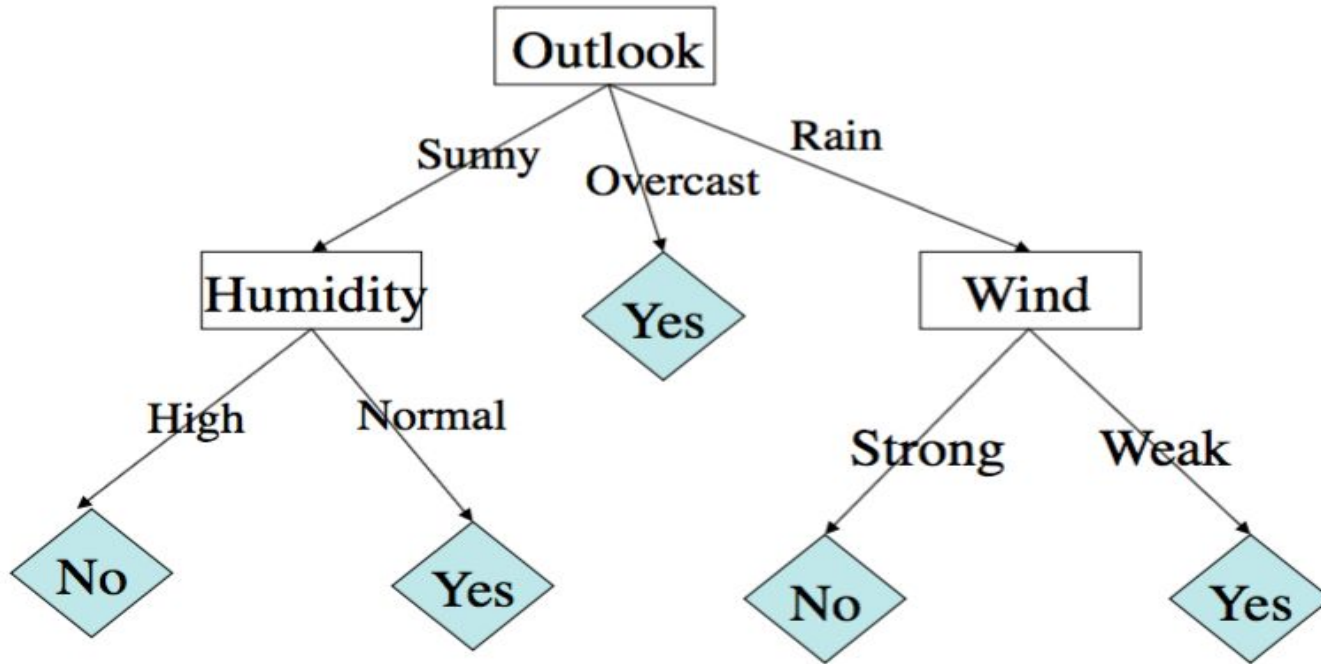
Outline

- **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

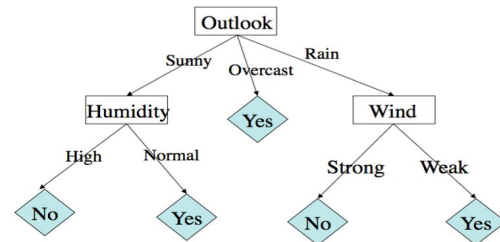
A Possible Decision Tree



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

$$H(X) = -\sum_{j=1}^M p(x_j) \log_2 p(x_j), \text{ where} \\ p(x_j) = P(X = x_j)$$

- 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

$$p_1 = p_2 = \cdots = p_M = \frac{1}{M}$$

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

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

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

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

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

$$H(X) = -[1 \log(1) + 0 \log(0) + 0 \log(0) + 0 \log(0)] = 0$$

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

Conditional Entropy

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

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

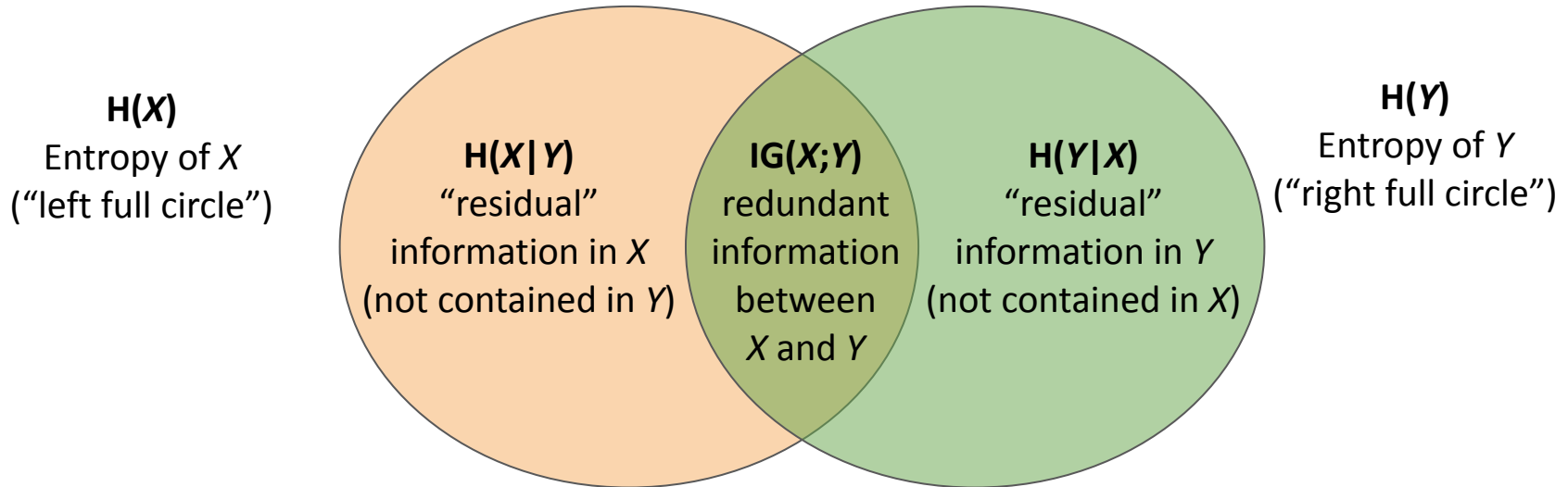
- Intuitively it is the expected entropy of X given different realizations of Y

Information Gain & Node Splitting

- The information gain is defined as:

$$IG(X, Y) = H(X) - H(X|Y)$$

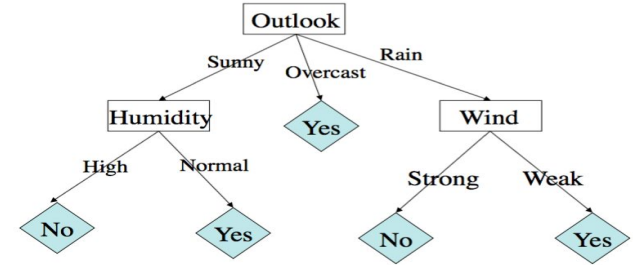
- Intuitively, the information gain measures how much more certain we are about X if we know Y



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

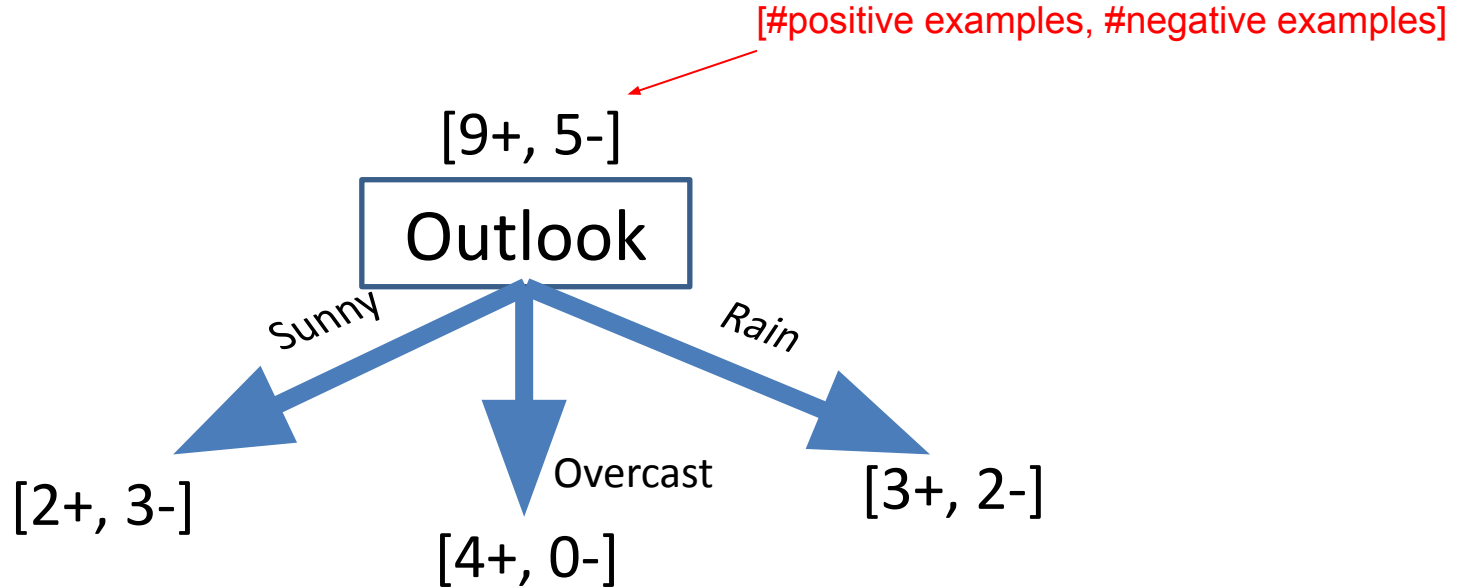
- Originally, in the whole dataset, the person played tennis on 9 out of the 14 days. Hence, the entropy of PlayTennis (P) is

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

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

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

- The information gain is therefore:

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

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

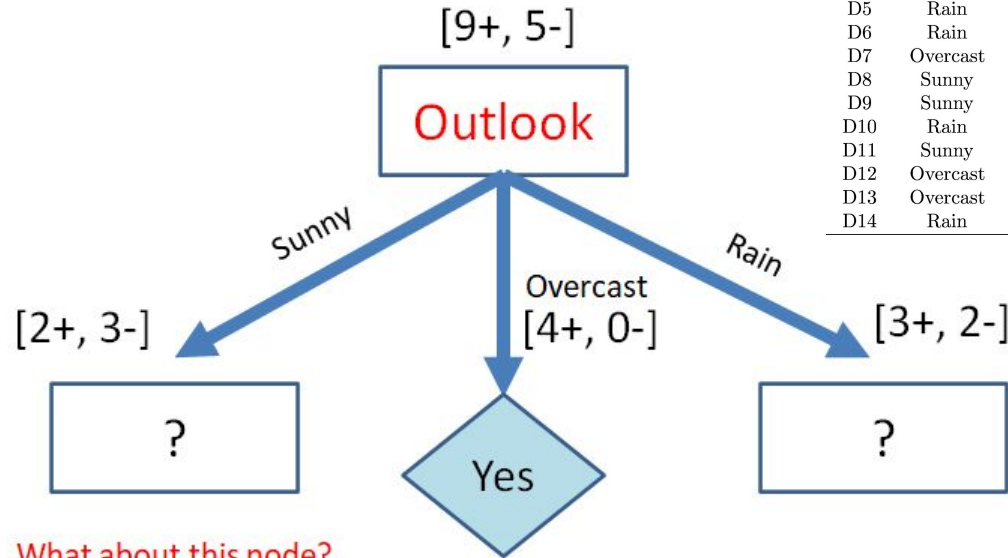
Decision Tree on Playing Tennis

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



What about this node?

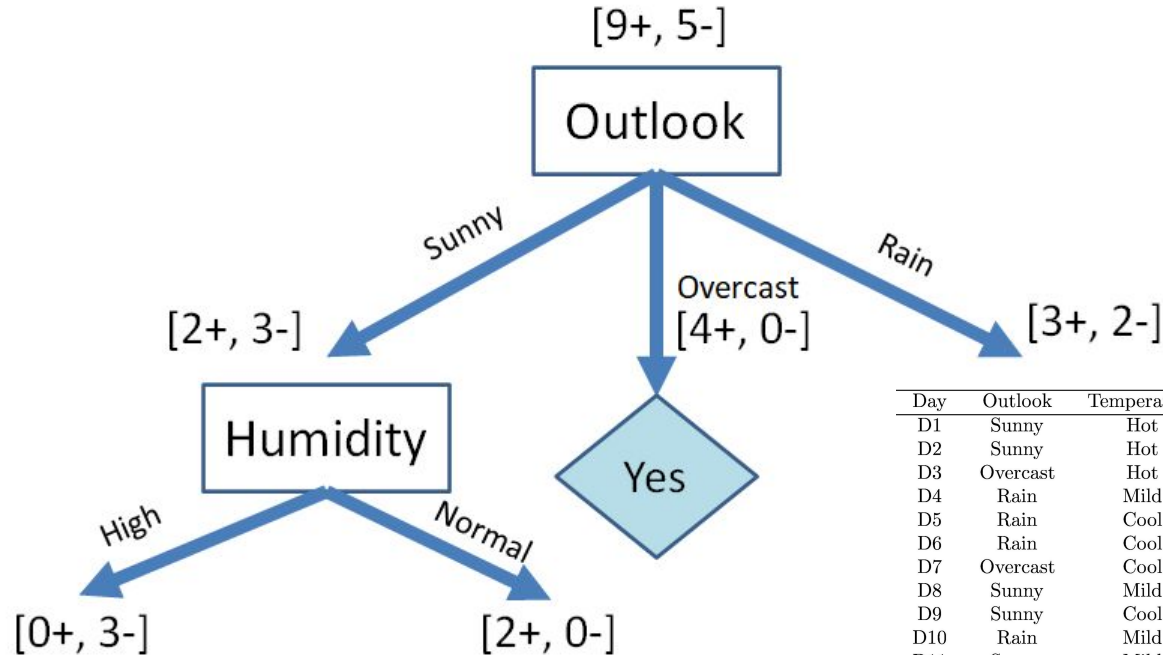
Decision Tree on Playing Tennis

- Given the outlook is sunny, the conditional entropy is

$$H(P|O = \textit{sunny}) = -\frac{2}{2+3}\log\left(\frac{2}{5}\right) - \frac{3}{2+3}\log\left(\frac{3}{5}\right) \\ = 0.97$$

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:

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

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

$$IG(P|Outlook = sunny, Hm) = 0.97 - 0 = 0.97$$

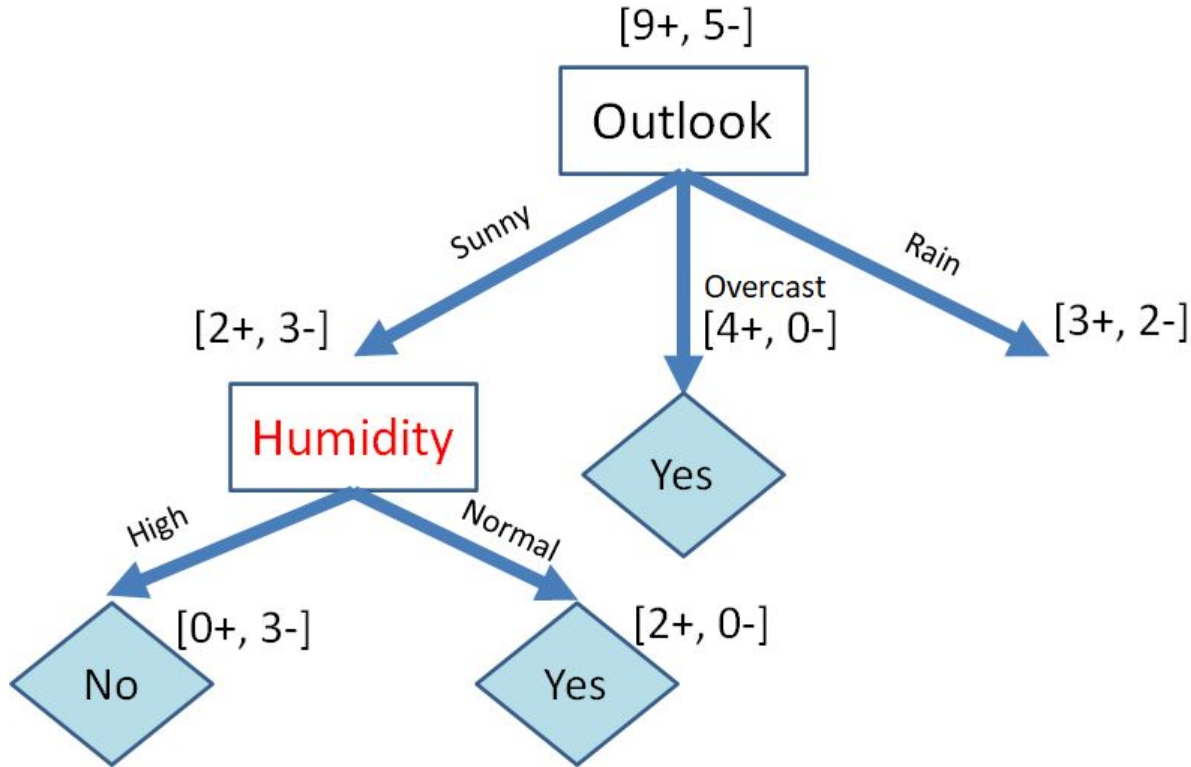
- Similarly

$$IG(P|Outlook = sunny, Temp) = 0.57$$

$$IG(P|Outlook = sunny, Wind) = 0.019$$

- 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)

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: $\{(x^{(n)}, y^{(n)}): n=1, \dots, N\}$, where $x^{(n)} \in \mathbf{R}^d$, $y \in \{+1, -1\}$
- Model
 - Hypothesis class: $H = \{h|h: \mathbf{R}^d \rightarrow \{+1, -1\}\}$
- Objective: $\hat{h} = \arg \min_{h \in \mathcal{H}} \frac{1}{N} \sum_{n=1}^N \mathbb{I} \left[y^{(n)} \neq h(x^{(n)}) \right]$
- 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 $h_1 \dots h_M$
- Bagging reduces variance (Increases classifier stability)
 - However, bagging does not reduce bias.

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
- Key in combining the weak rules

Boosting (Schapire 1989)

- Randomly select $n_1 < n$ samples from D without replacement 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, re-weight
 - Previous weak learner has only $>50\%$ accuracy over new distribution
- Can be used to learn weak classifiers
- Final classification based on weighted vote of weak classifiers

What's Good about Adaboost

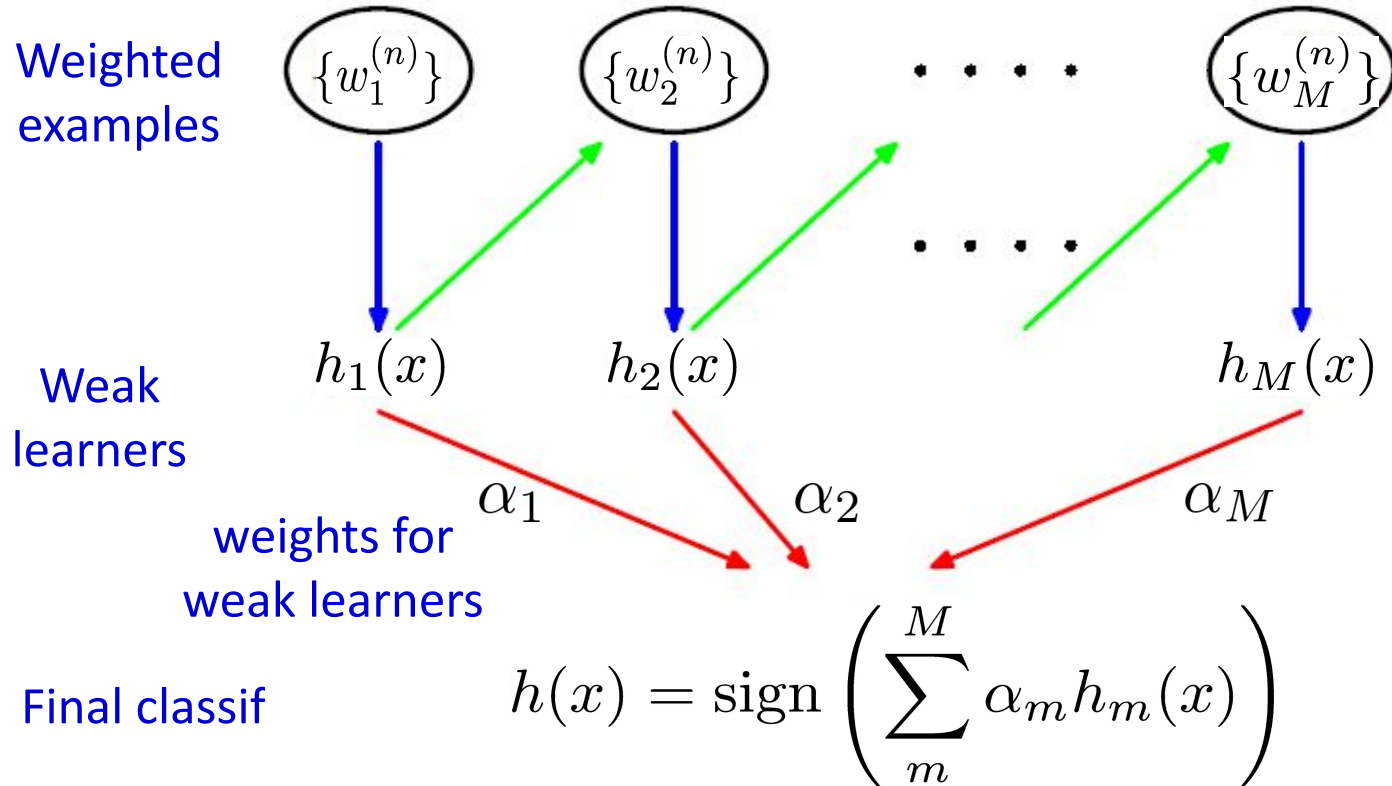
- Improves classification accuracy
- Can be used with many different classifiers
- Commonly used in many areas
- Simple to implement
- Robust to overfitting

Learning framework

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

$$h(x) = \text{sign} \left(\sum_m^M \alpha_m h_m(x) \right)$$

AdaBoost: Illustration



AdaBoost Algorithm

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

AdaBoost Algorithm

- Initialize data weights: $w_1^{(n)} = 1/N$ $n = 1, \dots, N$.
- For $m=1, \dots, M$ (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: $w_1^{(n)} = 1/N$ $n = 1, \dots, N$.
- For $m=1, \dots, M$ (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
$$\epsilon_m = \frac{\sum_{n=1}^N w_m^{(n)} I \left(h_m \left(\mathbf{x}^{(n)} \right) \neq y^{(n)} \right)}{\sum_{n=1}^N w_m^{(n)}} \quad \alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\} .$$
 - Evaluate weighted error and set the classifier weight

AdaBoost Algorithm

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

- For $m=1, \dots, M$ (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

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

- Evaluate weighted error and set the classifier weight

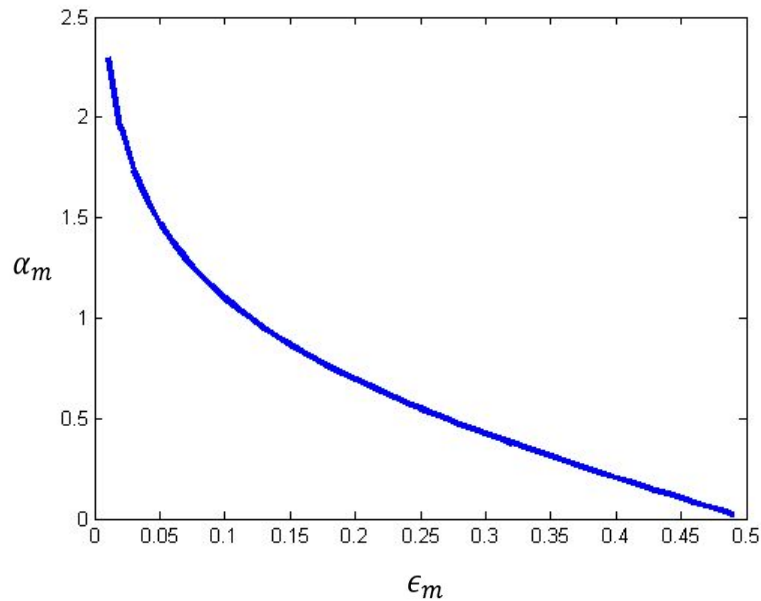
- update data weights $w_{m+1}^{(n)} = w_m^{(n)} \exp \left\{ \alpha_m I \left(h_m \left(\mathbf{x}^{(n)} \right) \neq y^{(n)} \right) \right\}$

- Make final prediction

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

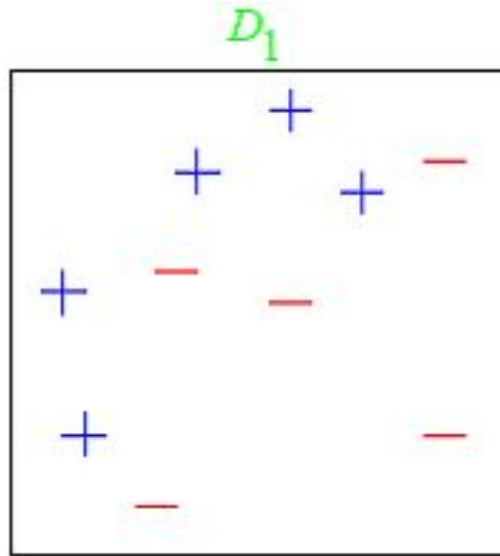
Relationship between weighted error and classifier weights

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



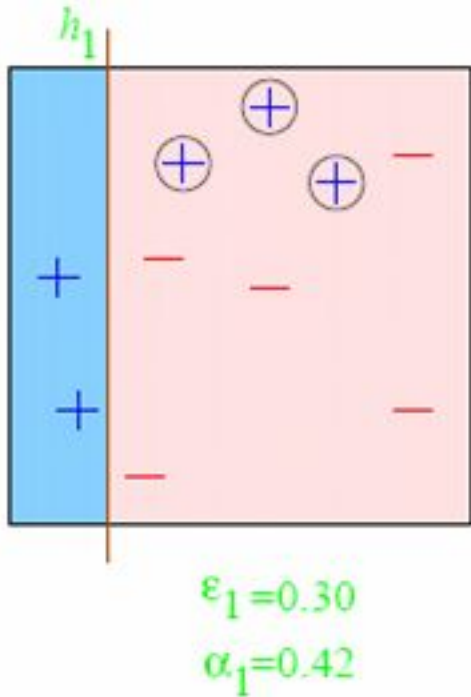
Example

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



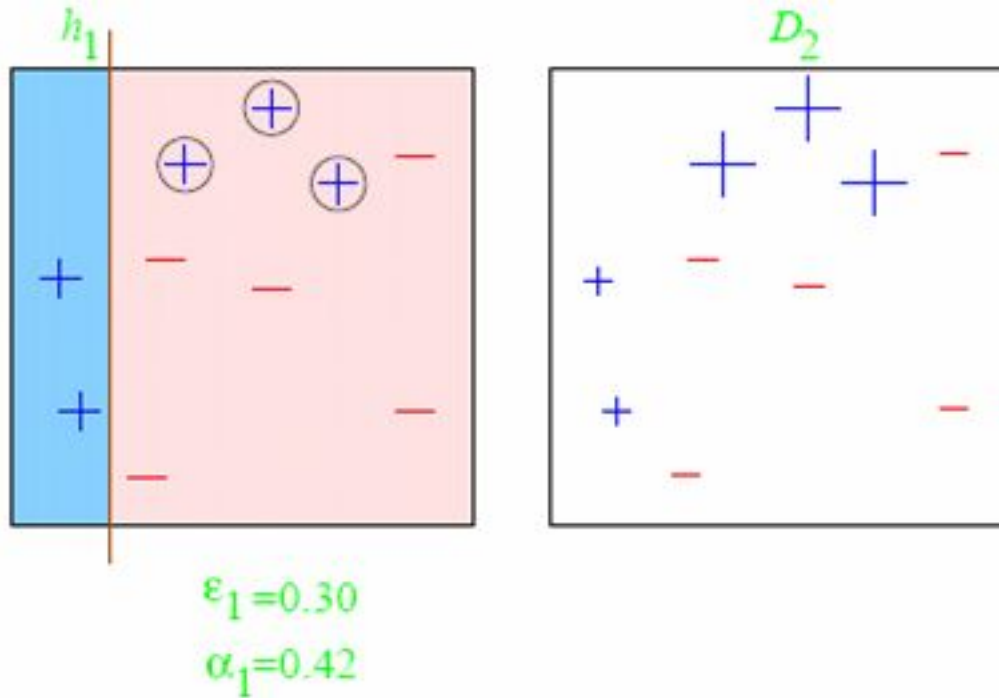
Example

- The first round



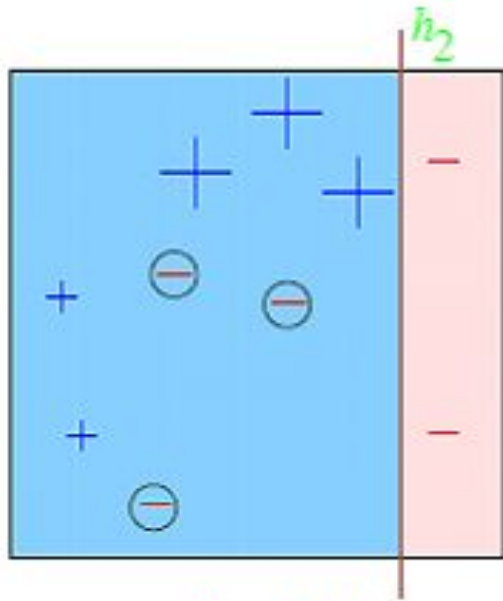
Example

- The first round



Example

- The second round

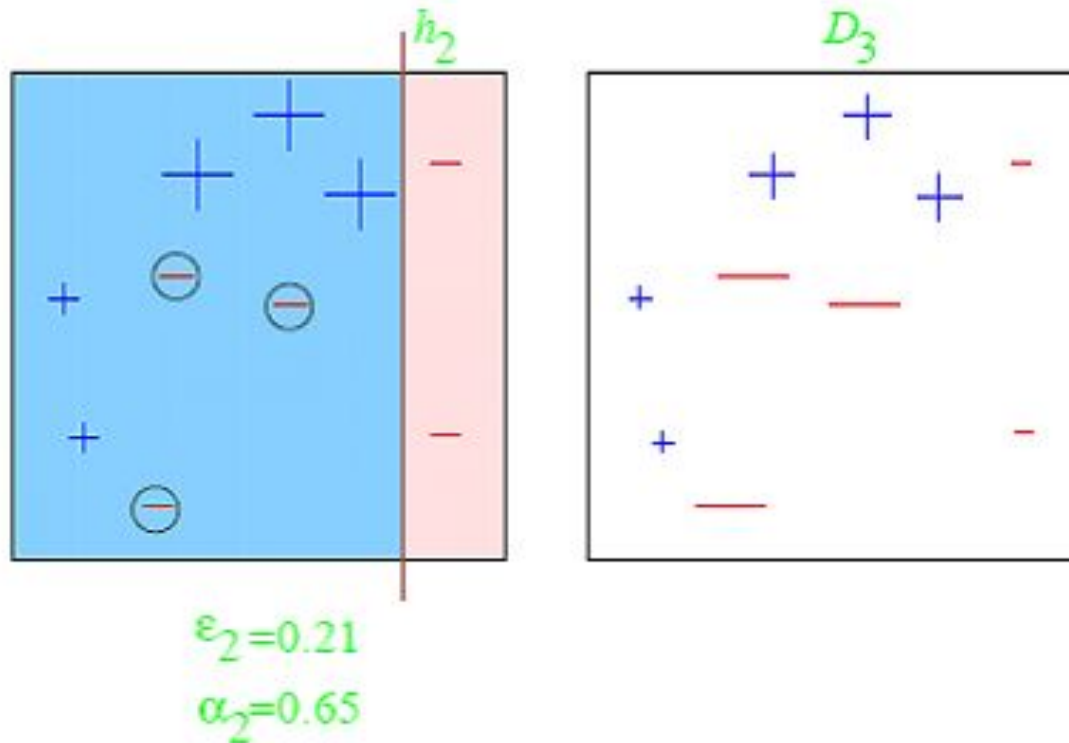


$$\epsilon_2 = 0.21$$

$$\alpha_2 = 0.65$$

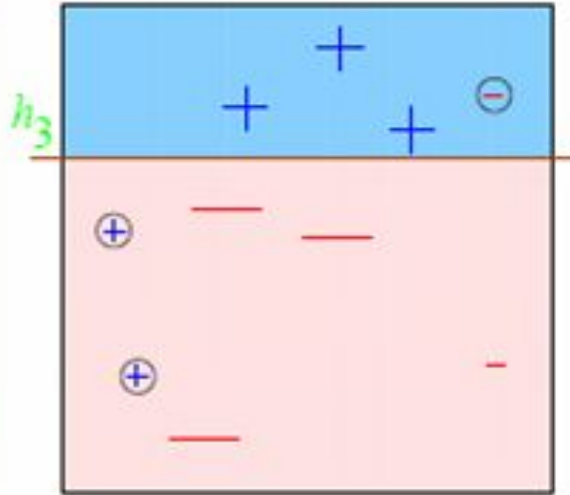
Example

- The second round



Example

- The third round

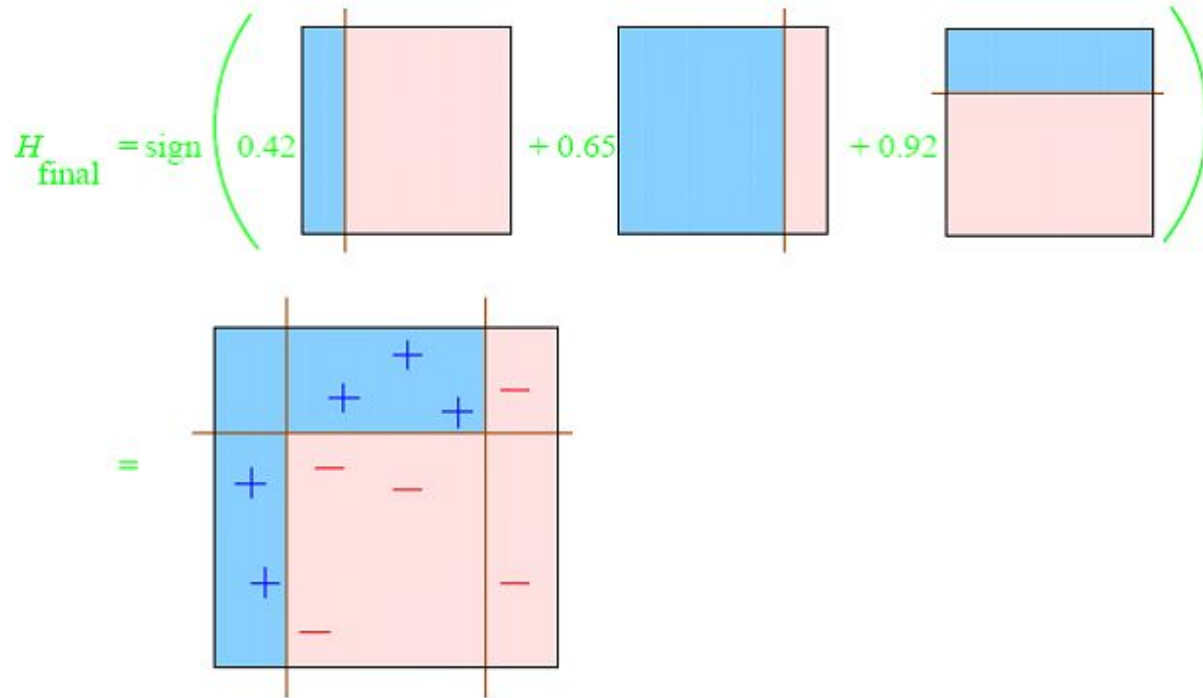


$$\epsilon_3=0.14$$

$$\alpha_3=0.92$$

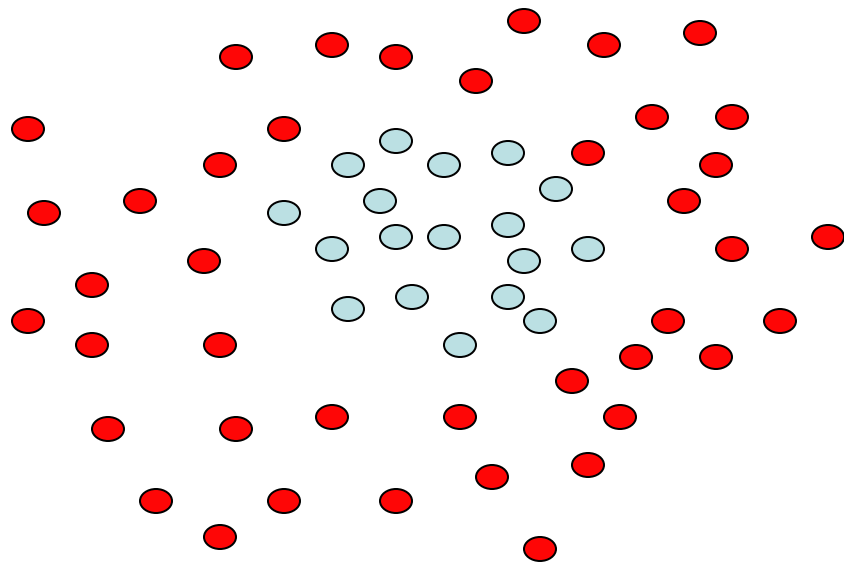
Example

- Final classifier (combining the weak learners)



Another example

- Boosting is a sequential procedure:



Each data point has
a class label:

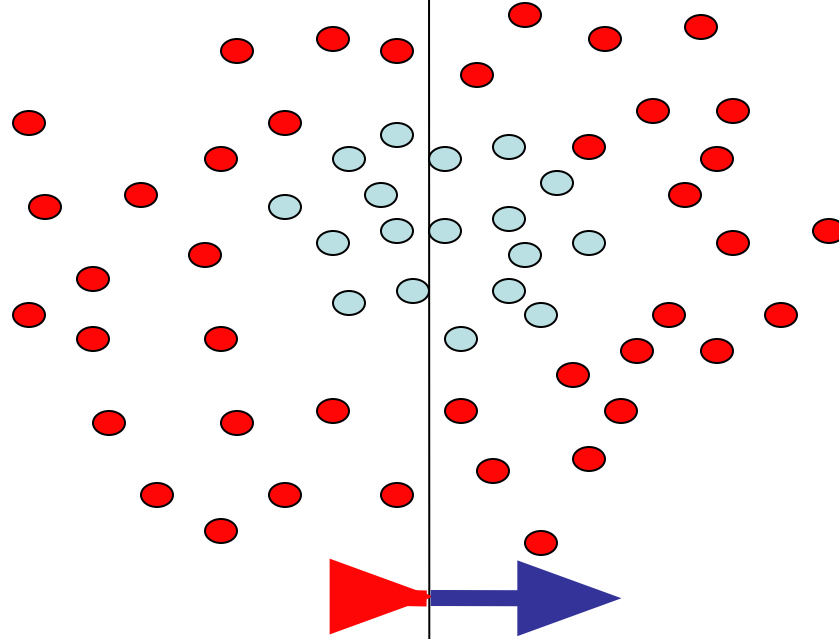
$$y^{(i)} = \begin{cases} +1 & (\bullet) \\ -1 & (\circ) \end{cases}$$

and a weight:

$$W^{(i)} = 1/N$$

Another example

Weak learners from the family of lines



Each data point has
a class label:

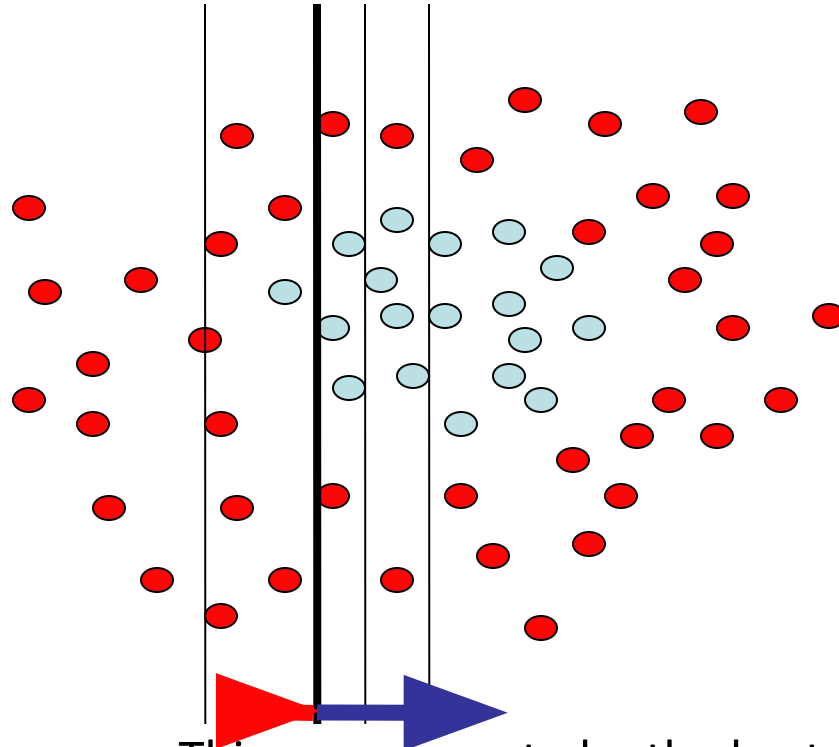
$$y^{(i)} = \begin{cases} +1 & (\text{red circle}) \\ -1 & (\text{blue circle}) \end{cases}$$

and a weight:

$$W^{(i)} = 1/N$$

$h \Rightarrow p(\text{error}) = 0.5$ it is at chance

Another example



Each data point has
a class label:

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

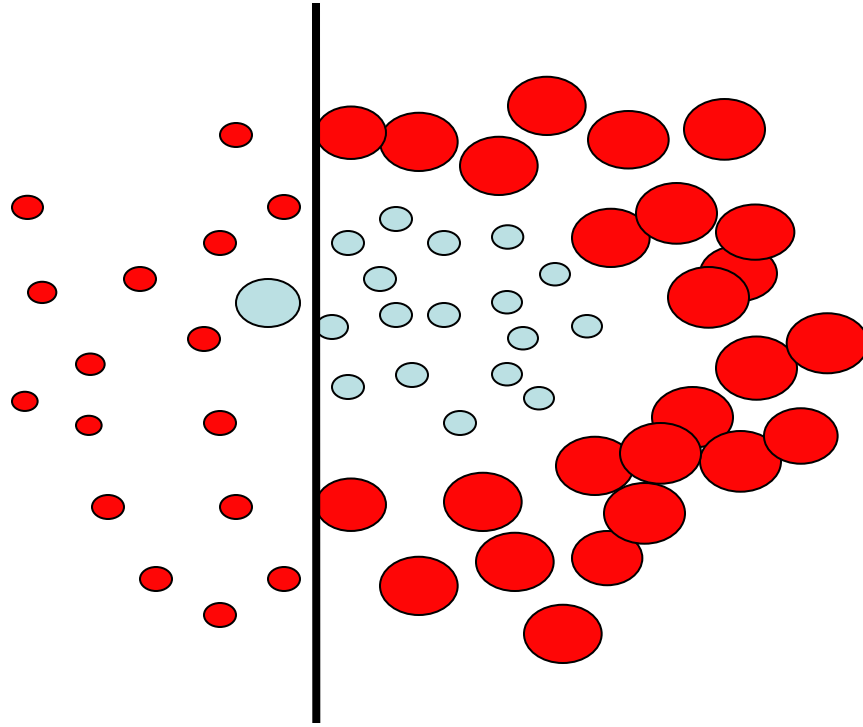
and a weight:

$$W^{(i)} = 1/N$$

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:

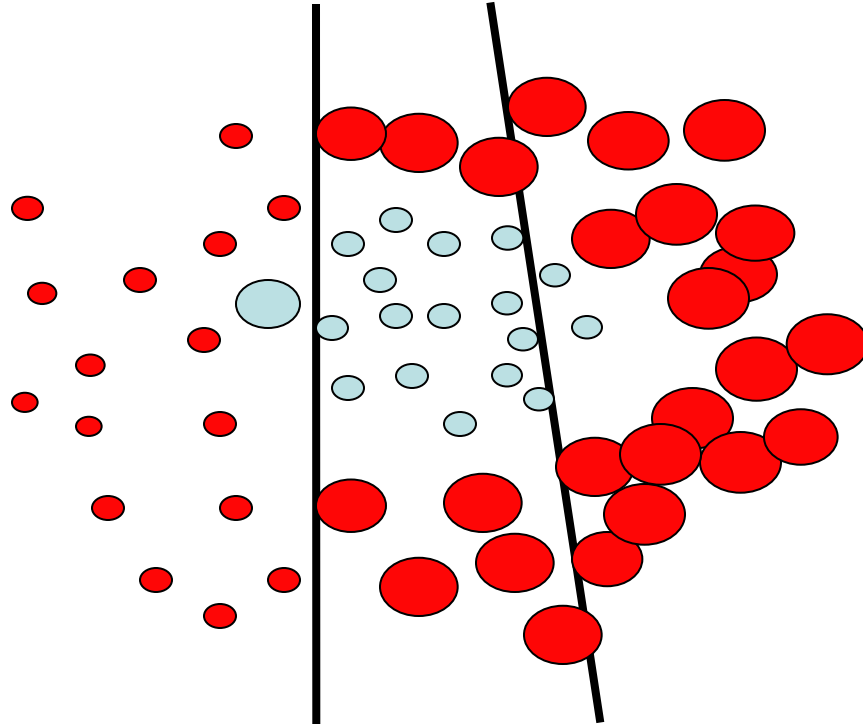
$$y^{(n)} = \begin{cases} +1 & (\bullet) \\ -1 & (\circ) \end{cases}$$

**We update the
weights:**

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

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

Another example



Each data point has
a class label:

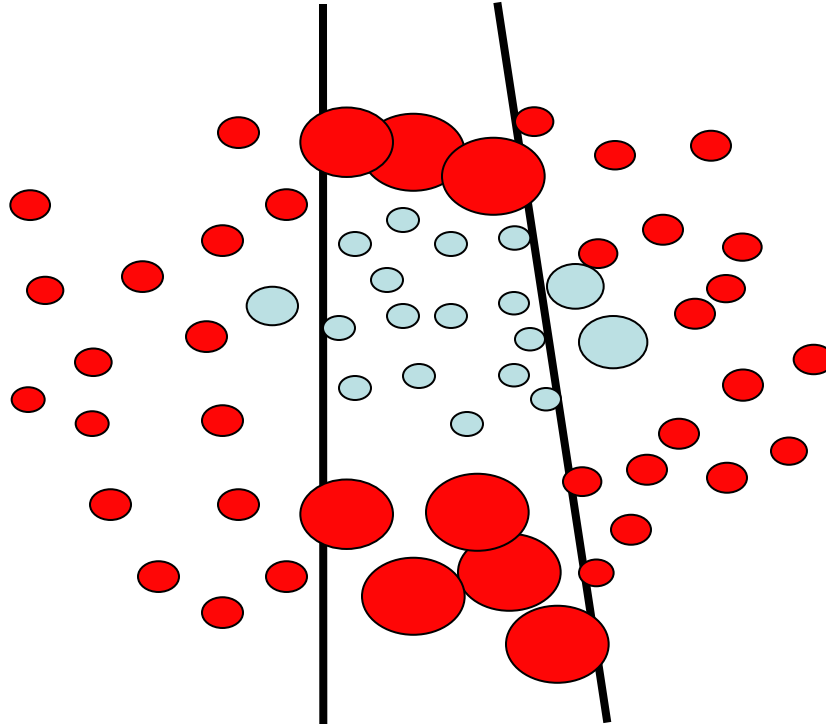
$$y^{(n)} = \begin{cases} +1 & (\bullet) \\ -1 & (\circ) \end{cases}$$

**We update the
weights:**

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

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

Another example



Each data point has
a class label:

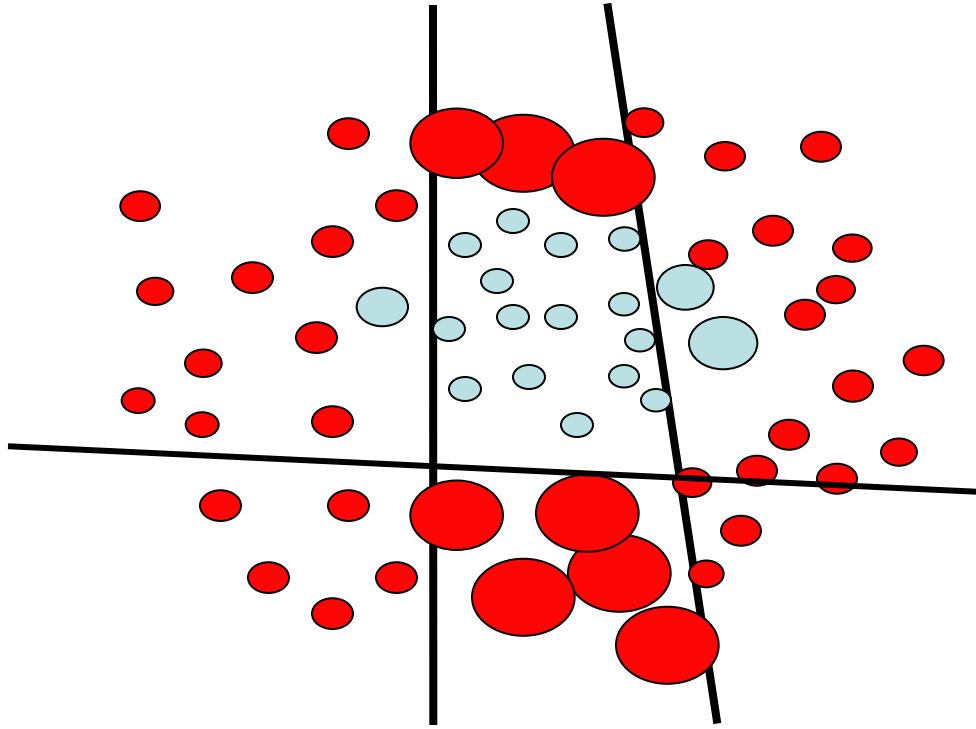
$$y^{(n)} = \begin{cases} +1 & (\bullet) \\ -1 & (\circ) \end{cases}$$

**We update the
weights:**

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

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

Another example



Each data point has
a class label:

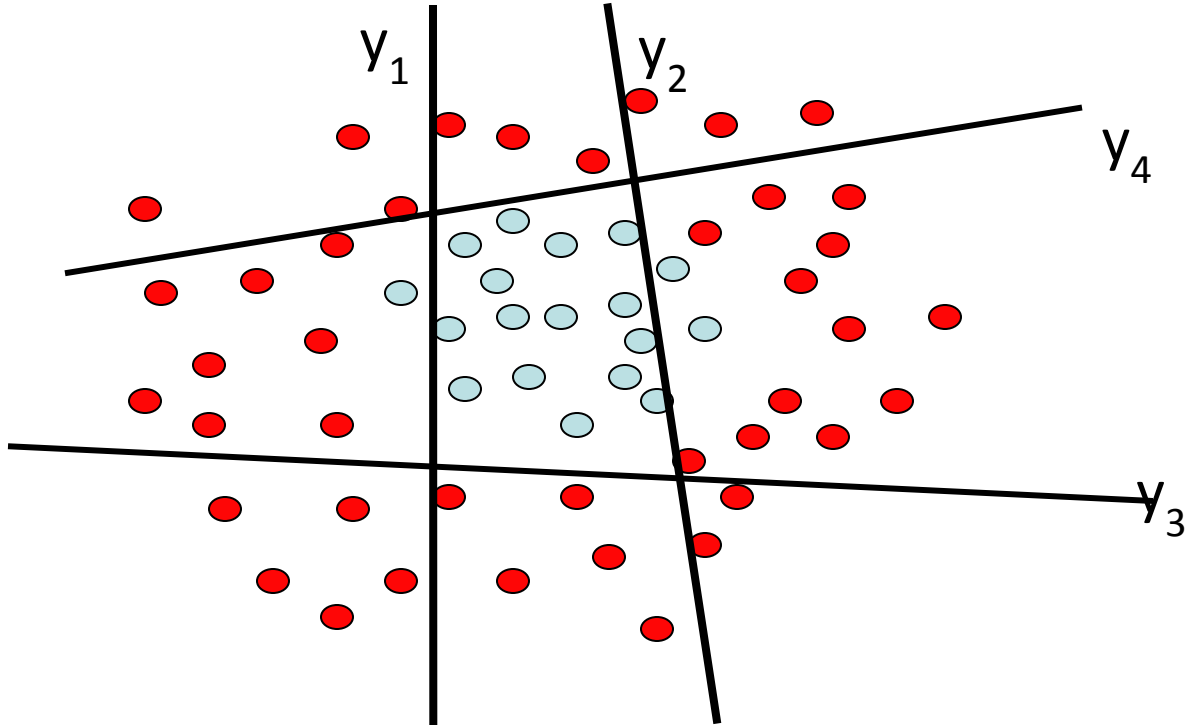
$$y^{(n)} = \begin{cases} +1 & (\bullet) \\ -1 & (\circ) \end{cases}$$

**We update the
weights:**

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

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

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

- 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 $h_1(\mathbf{x}), \dots, h_{m-1}(\mathbf{x})$ and their weights $\alpha_1, \dots, \alpha_{m-1}$, we can write J as:

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

Minimizing exponential error

$$J = \sum_{n=1}^N w_m^{(n)} \exp \left\{ -\frac{1}{2} y^{(n)} \alpha_m h_m \left(\mathbf{x}^{(n)} \right) \right\}$$

$$w_m^{(n)} = \exp \left\{ -y^{(n)} f_{m-1} \left(\mathbf{x}^{(n)} \right) \right\} \quad f_m(\mathbf{x}) = \frac{1}{2} \sum_{l=1}^m \alpha_l h_l(\mathbf{x})$$

- Define T_m / M_m as correctly/incorrectly classified examples by classifier h_m , respectively.

$$\begin{aligned} J &= e^{-\alpha_m/2} \sum_{n \in T_m} w_m^{(n)} + e^{\alpha_m/2} \sum_{n \in M_m} w_m^{(n)} \\ &= \left(e^{\alpha_m/2} - e^{-\alpha_m/2} \right) \sum_{n=1}^N w_m^{(n)} I \left(h_m \left(\mathbf{x}^{(n)} \right) \neq y^{(n)} \right) + e^{-\alpha_m/2} \sum_{n=1}^N w_m^{(n)} \end{aligned}$$

- Therefore, for each round, AdaBoost trains a weak learner by minimizing the weighted misclassification error.

Minimizing exponential error

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

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

$$w_{m+1}^{(n)} = w_m^{(n)} \exp \left\{ -\frac{1}{2} y^{(n)} \alpha_m h_m(\mathbf{x}^{(n)}) \right\}$$

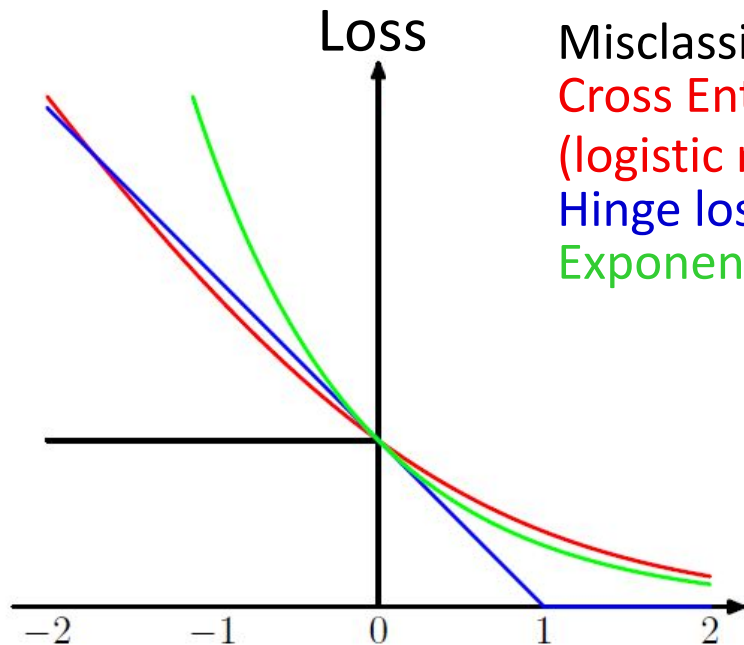
$$y^{(n)} h_m(\mathbf{x}^{(n)}) = 1 - 2I(h_m(\mathbf{x}^{(n)}) \neq y^{(n)})$$

$$w_{m+1}^{(n)} = w_m^{(n)} \exp(-\alpha_m/2) \exp \left\{ \alpha_m I(h_m(\mathbf{x}^{(n)}) \neq y^{(n)}) \right\}$$

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 $x^{(n)}$

Exponential loss



Misclassification error

Cross Entropy
(logistic regression)

Hinge loss (SVM)

Exponential loss

Cross Entropy

$$\ln(1 + \exp(-y f(x)))$$

Hinge loss

$$\max(1 - y f(x), 0)$$

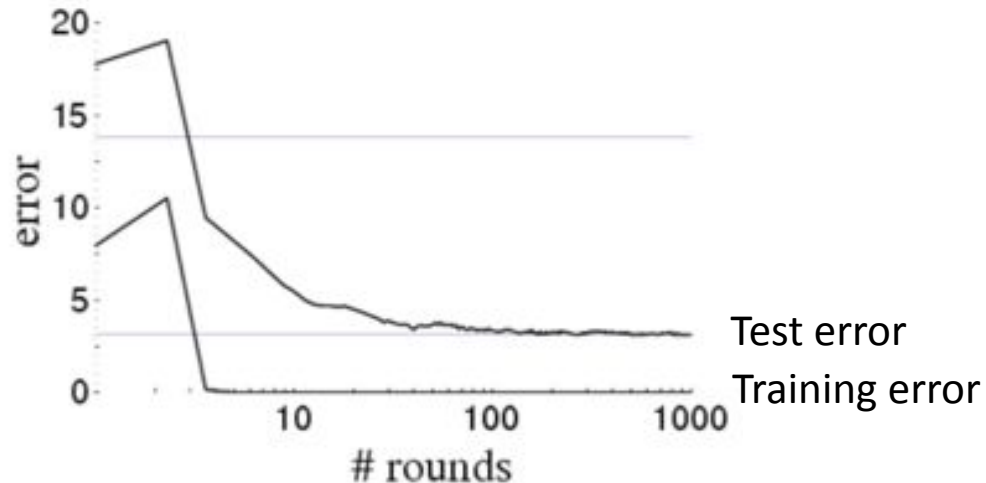
Exponential
loss

$$\exp(-y f(x))$$

$y f(x)$ = “margin”

Example runs of Adaboost

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



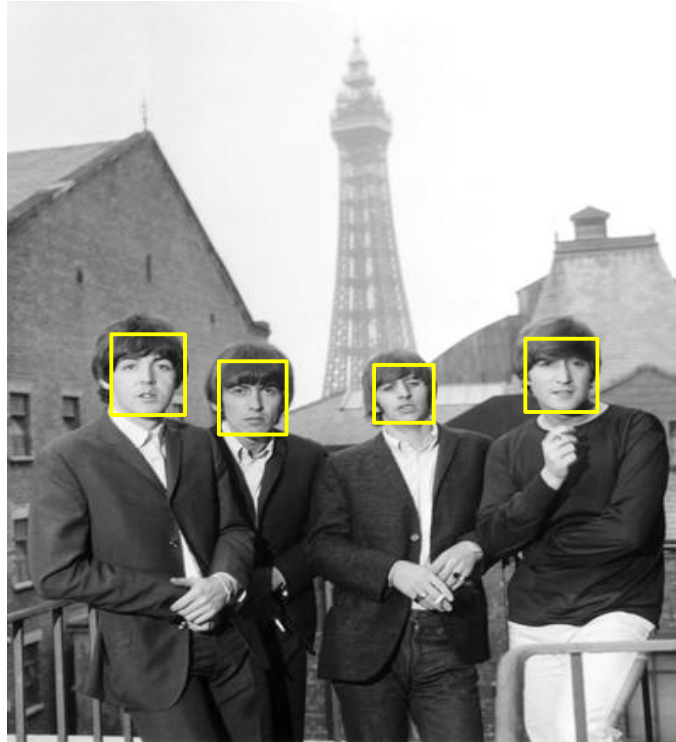
Face detection with boosting

- Viola-Jones face detector

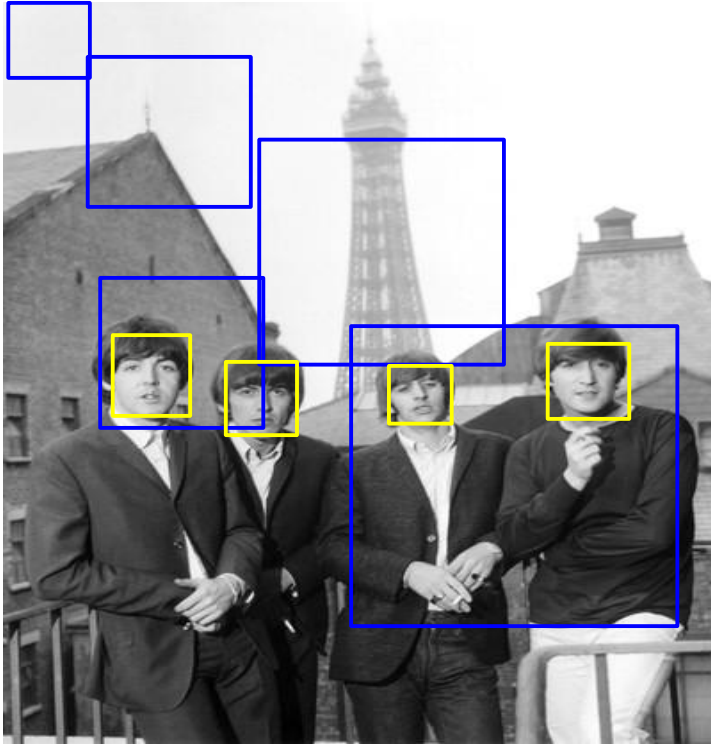


Face detection with boosting

- Viola-Jones face detector



Sliding Windows



1. hypothesize:

try all possible rectangle locations, sizes

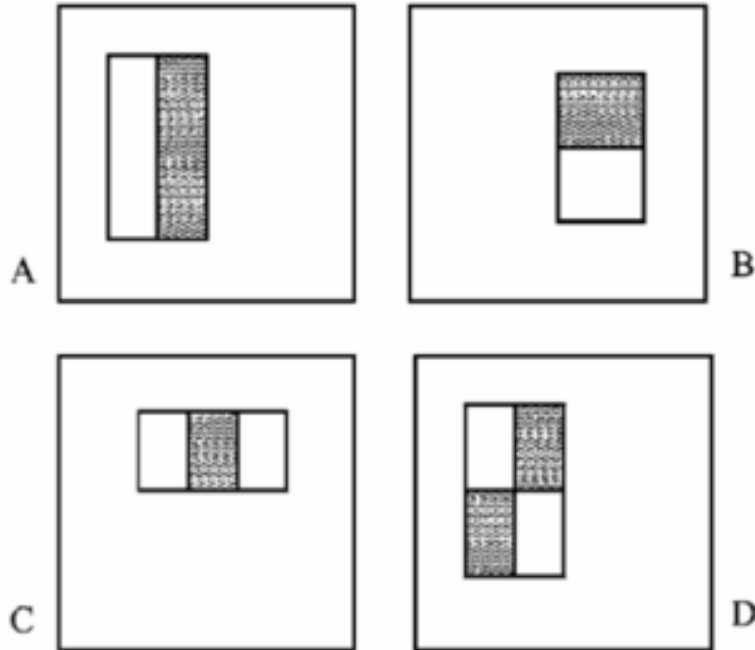
2. test:

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

Note: 1000's more false windows
than 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

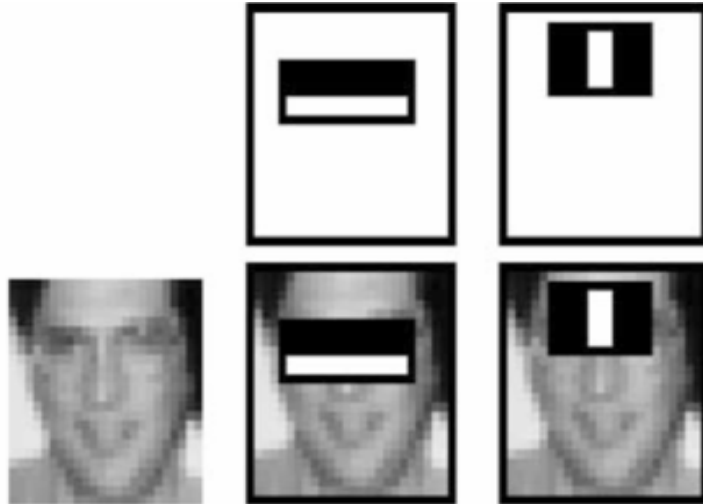


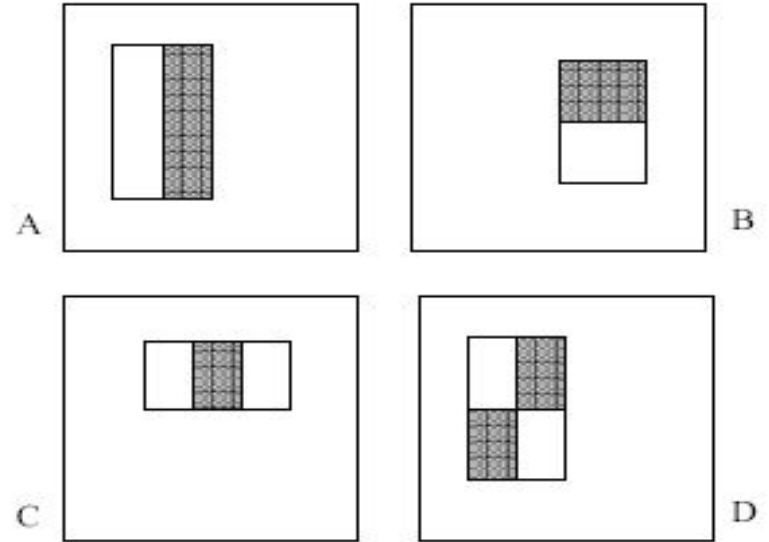
Image Features

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



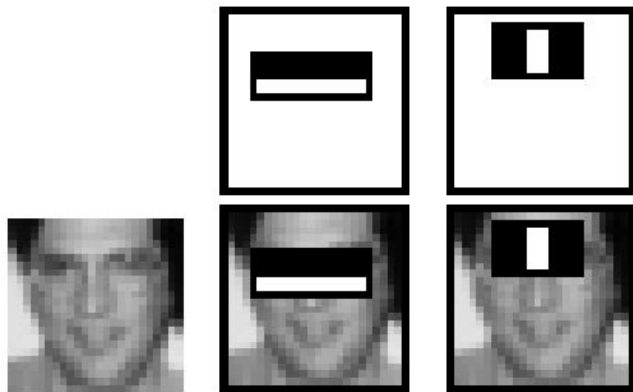
4 Types of “Rectangle filters”

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



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

Image Features



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

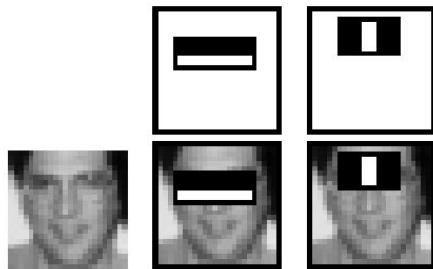
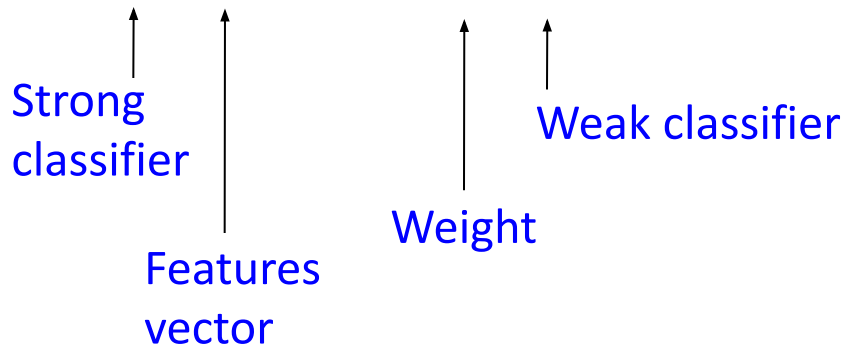
$$h_i(x) = \begin{cases} 1 & \text{if } g_i(x) > \theta_i \\ -1 & \text{otherwise} \end{cases}$$

- 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:

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



$$F(x) = \alpha_1 y_1(x) + \alpha_2 y_2(x) + \dots$$

A peek ahead: the learned features

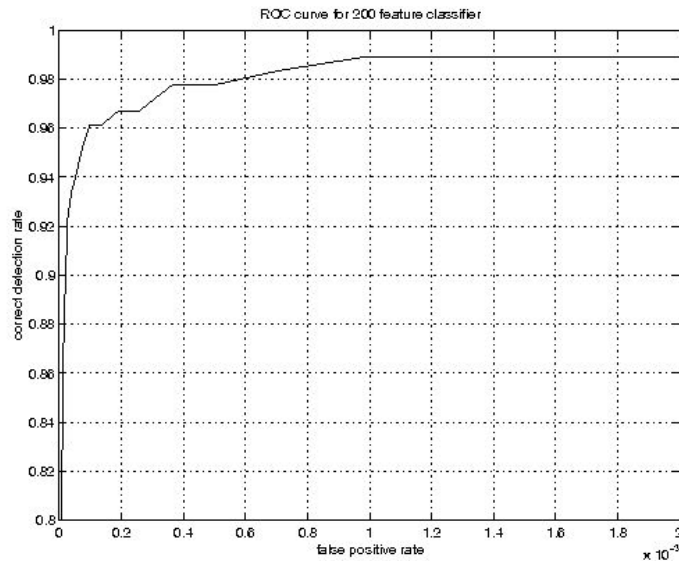
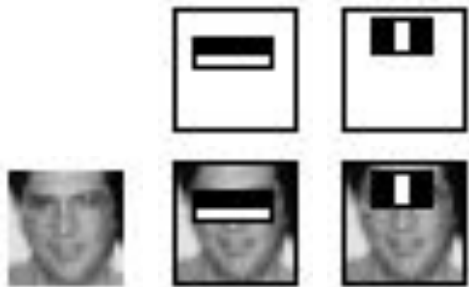


Example Classifier for Face Detection

A classifier with 200 rectangle features was learned using AdaBoost

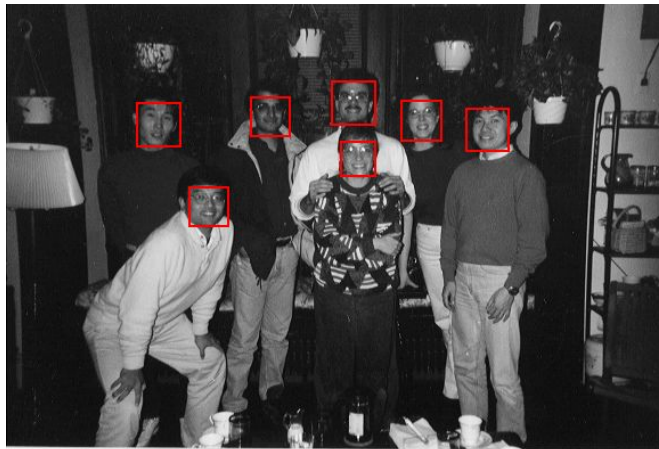
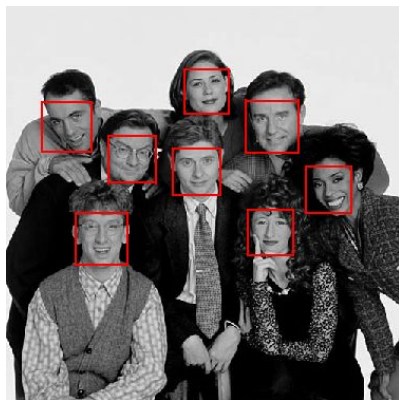
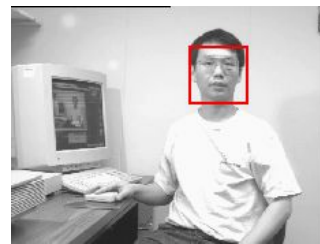
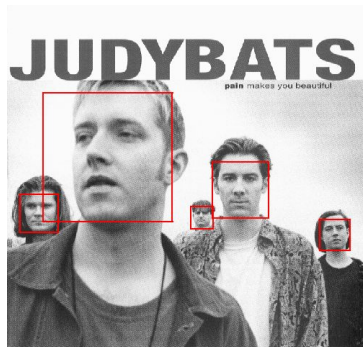
95% correct detection on test set with 1 in 14084 false positives.

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



ROC curve for 200 feature classifier

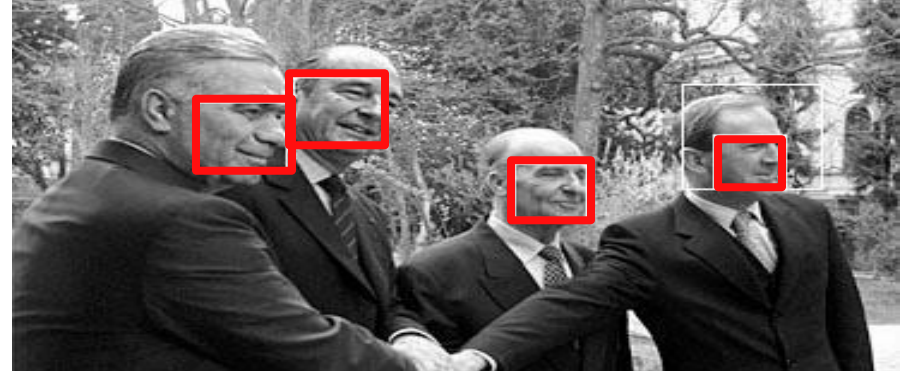
Output of Face Detector on Test Images



Solving other “Face” Tasks

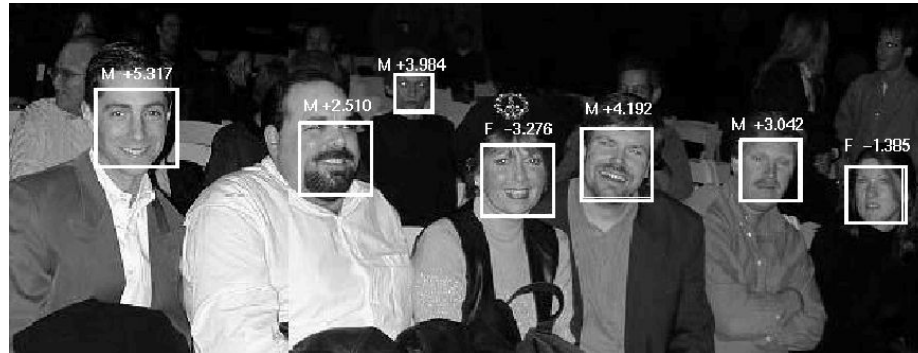


Facial Feature Localization



Profile Detection

Demographic Analysis



Quiz

[Link here](#) or QR code below

