

CSE 417T

Introduction to Machine Learning

Lecture 14

Instructor: Chien-Ju (CJ) Ho

Logistics

- Homework 3: due yesterday (Wednesday)
 - If you still have late days, the final due is tomorrow
 - Track your own late-day usages
 - Assignments over-using late days won't be graded
- Exam 1: **October 27 (Thursday)**
 - Topics: LFD Chapters 1 to 5
 - Timed exam (75 min) during lecture time
 - Location TBD
 - Closed-book exam with 2 letter-size cheat sheets allowed (4 pages in total)
 - No format limitations (it can be typed, written, or a combination)
- October 25 (Tuesday) will be a review session
 - Practice questions are posted on Piazza (will be discussed during the review session)

Recap

Decision Tree Hypothesis



Credit Card Approval Example

- Pros
 - Easy to interpret (interpretability is getting attention and is important in some domains)
 - Can handle multi-type data (Numerical, categorical. ...)
 - Easy to implement (Bunch of if-else rules)
- Cons
 - Generally speaking, **bad generalization**
 - VC dimension is infinity
 - High variance (small change of data leads to very different hypothesis)
 - Easily overfit
- Why we care?
 - One of the classical model
 - Building block for other models (e.g., random forest)

ID3: Using Information Gain as Selection Criteria

- Information gain of choosing feature A to split
 - $Gain(D, A) = H(D) - \sum_i \frac{|D_i|}{|D|} H(D_i)$ [The amount of decrease in entropy]
- ID3: Choose the split that maximize $Gain(D, A)$

Notations:

$H(D)$: Entropy of D

$|D|$ is the number of points in D

DecisionTreeLearn(D)

Create a root node r

If **termination conditions** are met

return a single node tree with **leaf prediction** based on

Else: Greedily find a feature A to split according to **split criteria**

For each possible value v_i of A

Let D_i be the dataset containing data with value v_i for feature A

Create a subtree DecisionTreeLearn(D_i) that being the child of root r

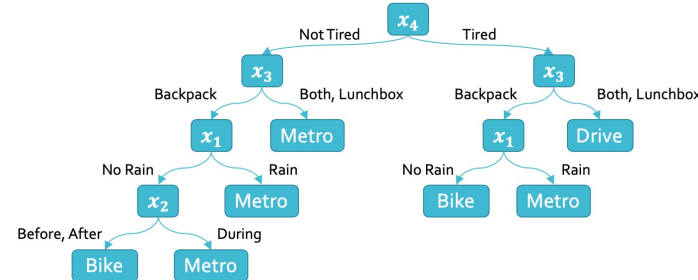
- ID3 termination conditions
 - If all labels are the same
 - If all features are the same
 - If dataset is empty
- ID3 leaf predictions
 - Most common labels (majority voting)
- ID3 split criteria
 - Information gain

Ensemble Learning

- Goal: Utilize a set of **weak learners** to obtain a **strong learner**.
- One common way to construct weak learners is via **decision trees**

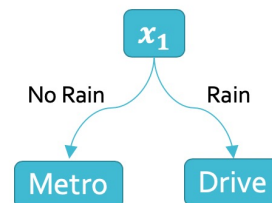
- Fully grown decision trees

- High variance
- Low bias



- Decision stump (One-depth decision trees, split on only one attribute)

- Low variance
- High bias



Ensemble Learning

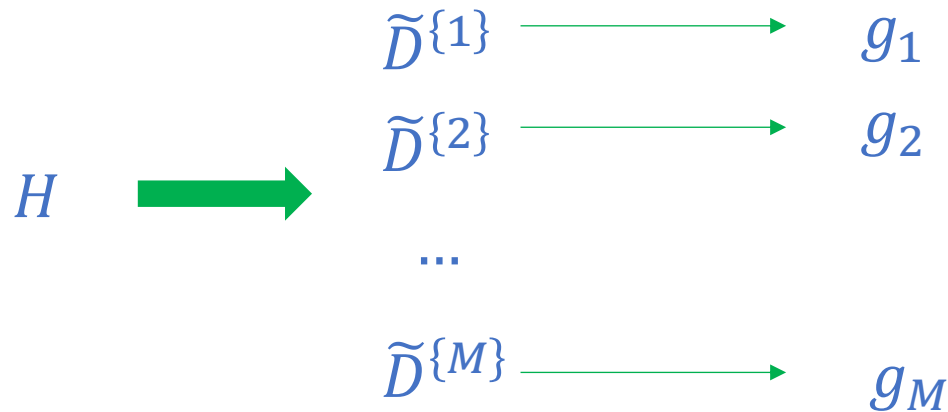
- Goal: Utilize a set of **weak learners** to obtain a **strong learner**.
- Format of ensemble learning
 - **Construct** many **diverse** weak learners
 - **Aggregate** the weak learners

Bagging

- Construct diverse weak learners
 - (**Simultaneously**) bootstrap datasets
 - Train weak learners on them
- Aggregate the weak learners
 - **Uniform** aggregation

Bagging - Bootstrapped Aggregating

- Bootstrap M datasets $\{\tilde{D}^{\{m\}}\}$ (Sample with replacement from D)
- Learn a hypothesis from each of them



- Aggregate the learned hypothesis (assume we are doing classification)

$$G(\vec{x}) = \bar{g}(\vec{x}) = \text{sign}\left(\frac{1}{M} \sum_{m=1}^M g_m(\vec{x})\right)$$

Out-Of-Bag (OOB) Error

	$\tilde{D}^{(1)}$	$\tilde{D}^{(2)}$	$\tilde{D}^{(3)}$	$\tilde{D}^{(4)}$...
(\vec{x}_1, y_1)	Yes	Yes	No	No	...
(\vec{x}_2, y_2)	Yes	No	No	No	...
...
(\vec{x}_N, y_N)	No	Yes	Yes	Yes	...

Whether a point is in a bootstrapped dataset

- G_n^- : the aggregation of hypothesis that \vec{x}_n is OOB of

- $G_1^- = \text{aggregate}(g_3, g_4, \dots)$
- $G_2^- = \text{aggregate}(g_2, g_3, g_4, \dots)$
- $G_N^- = \text{aggregate}(g_1, \dots)$

Aggregate:
Majority voting for classification
Average for regression

- OOB Error

- $E_{OOB}(G) = \frac{1}{N} \sum_{n=1}^N \text{error}(G_n^-(\vec{x}_n), y_n)$

Error:
Binary error for classification
Squared error for regression

Out-Of-Bag (OOB) Error

$$E_{OOB}(G) = \frac{1}{N} \sum_{n=1}^N \text{error}(G_n^-(\vec{x}_n), y_n)$$

- Bagging provided an **intrinsic** mechanism for us to perform validation
 - We don't need to split the dataset into training and validation
- Practical issues (you might face this in HW4)
 - What if some \vec{x}_n appears in all bootstrapped datasets?
 - The probability of this happening is small when the number of bags M is large
 - Let S be the set of points that is out of bag for at least one bootstrapped dataset
 - $E_{OOB}(G) = \frac{1}{|S|} \sum_{(\vec{x}_n, y_n) \in S} \text{error}(G_n^-(\vec{x}_n), y_n)$

Random Forest

- Construct many random trees
 - Bootstrapping datasets and learn a **max-depth tree** for each of them
 - Other randomizations (not required in HW4)
 - When choosing split features, choose from a random subset (instead of all features)
 - Randomly project features (similar to non-linear transformation) for each tree
- Aggregate the random trees
 - Classification: Majority vote $\bar{g}(\vec{x}) = \text{sign} \left(\frac{1}{M} \sum_{m=1}^M g_m(\vec{x}) \right)$
 - Regression: Average $\bar{g}(\vec{x}) = \frac{1}{M} \sum_{m=1}^M g_m(\vec{x})$

Today's Lecture

The notes are not intended to be comprehensive. They should be accompanied by lectures and/or textbook.
Let me know if you spot errors.

Boosting

Ensemble Learning

- Goal: Utilize a set of **weak learners** to obtain a **strong learner**.
- Format of ensemble learning
 - **Construct** many **diverse** weak learners
 - **Aggregate** the weak learners

Bagging:

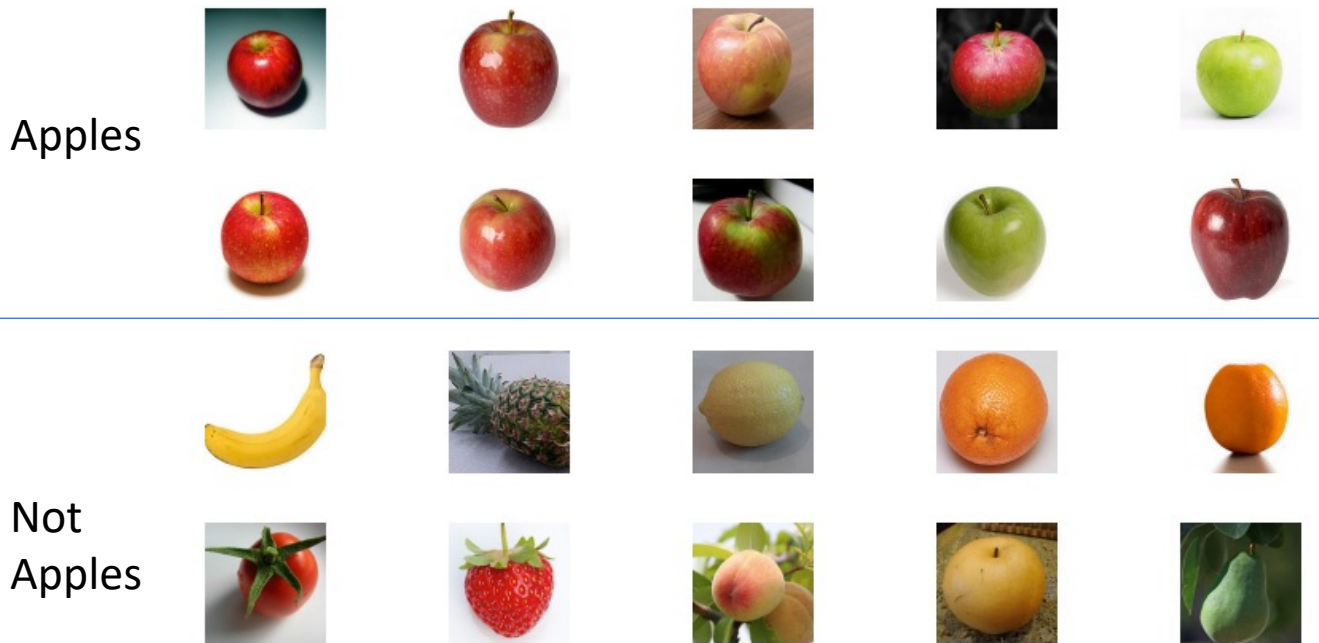
- Construct diverse weak learners
 - (**Simultaneously**) bootstrapping datasets
 - Train weak learners on them
- Aggregate the weak learners
 - **Uniform** aggregation

Boosting

- Construct diverse weak learners
 - **Adaptively** generating datasets
 - Train weak learners on them
- Aggregate the weak learners
 - **Weighted** aggregation

Informal Intuitions about Boosting

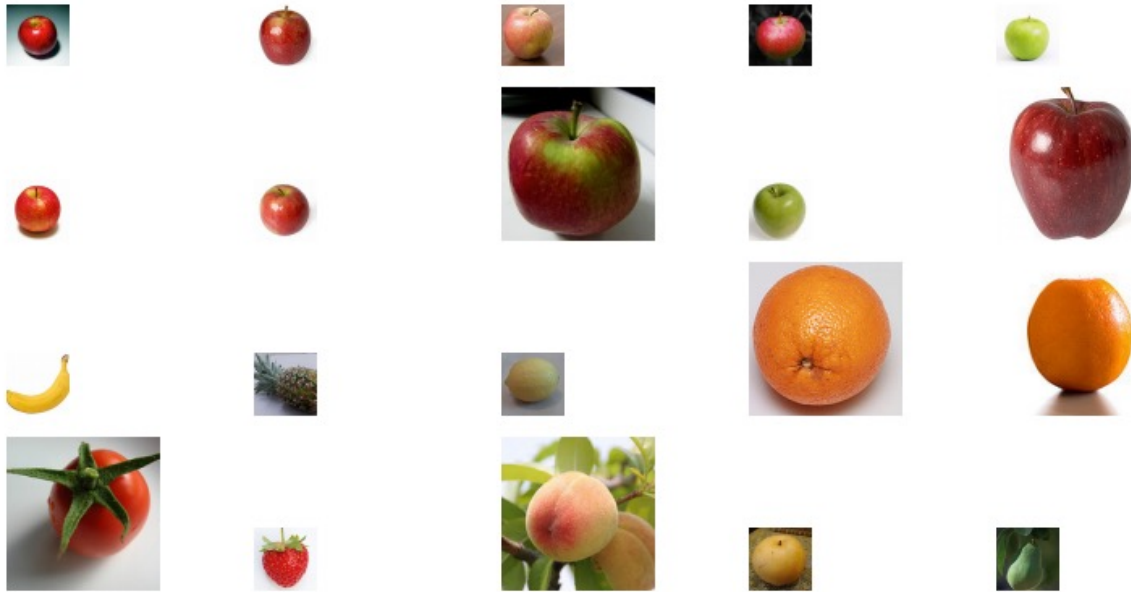
- Example : Teach a class of kids to identify apples from data



- Alice: Apples are **circular**
- Teacher:
Circular is a good feature, but using this feature might make some mistakes
Let me **highlight** the mistakes.
 - Make correct images smaller
 - Make incorrect images larger

Informal Intuitions about Boosting

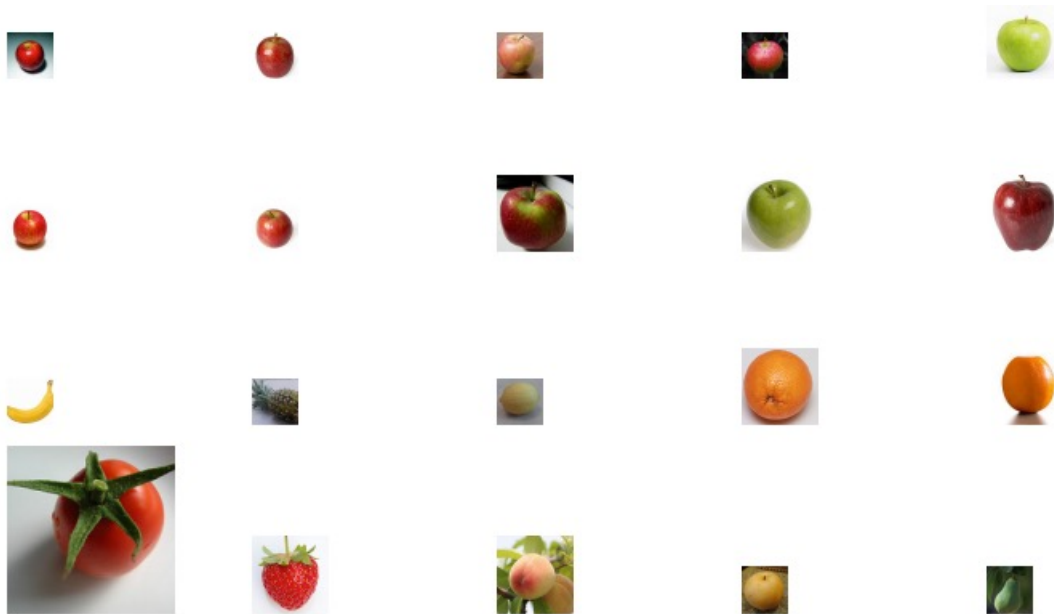
- Example : Teach a class of kids to identify apples from data



- Alice: Apples are **circular**
- Bob: Apples are **red**

Informal Intuitions about Boosting

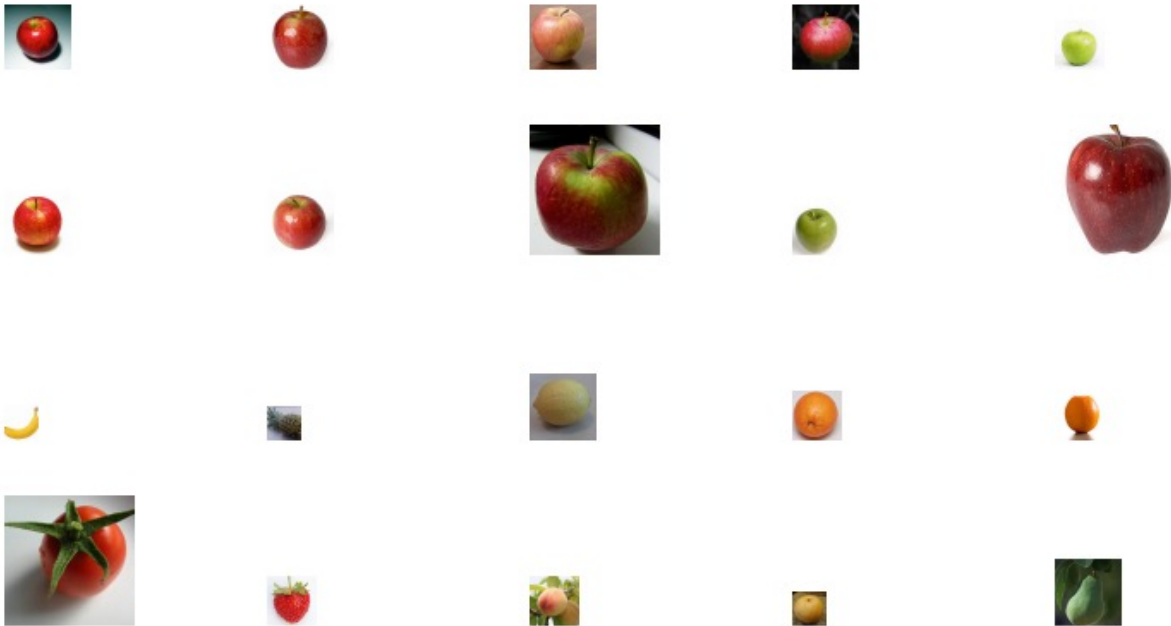
- Example : Teach a class of kids to identify apples from data



- Alice: Apples are **circular**
- Bob: Apples are **red**
- Charlie: Apples could be **green**

Informal Intuitions about Boosting

- Example : Teach a class of kids to identify apples from data



- Alice: Apples are **circular**
- Bob: Apples are **red**
- Charlie: Apples could be **green**
- David: Apples have **stems** at the top
- Class: Apples are **somewhat circular, somewhat red, possibly green, and may have stems at the top**

Informal Intuitions about Boosting

- Example : Teach a class of kids to identify apples from data



Key steps of this process:

- Learn a **simple** hypothesis for each dataset
- Iteratively update the dataset to focus on what we got wrong (i.e., create **diversity**)
- **Aggregate** the learned simple hypothesis



- Alice: Apples are **circular**
- Bob: Apples are **red**
- Charlie: Apples could be **green**
- David: Apples have **stems** at the top
- Class: Apples are **somewhat circular, somewhat red, possibly green, and may have stems at the top**

Outline of a Boosting Algorithm

- Initialize D_1 (usually the same as the initial dataset D)
- For $t = 1$ to T
 - Learn g_t from D_t
 - Reweight the distribution and obtain D_{t+1} based on g_t and D_t
- Output **weighted**-aggregate(g_1, \dots, g_T)
 - Classification: $G(\vec{x}) = \bar{g}(\vec{x}) = \text{sign}\left(\frac{1}{T} \sum_{t=1}^T \alpha_t g_t(\vec{x})\right)$

Questions

How to learn g_t from D_t

How to reweight the distribution and obtain D_{t+1}

How to perform weighted aggregation

Discussion on Re-weighted D_t (What does re-weighting mean?)

- Original Dataset $D = \{(\vec{x}_1, y_1), \dots, (\vec{x}_N, y_N)\}$
- Notation of D_t
 - $D_t(n)$ is the weight/probability of data point (\vec{x}_n, y_n) in D_t
 - $\sum_{n=1}^N D_t(n) = 1$
- What is $E_{in}(h)$ on D_t ? (Expressed as $E_{in}^{(D_t)}(h)$)
 - Re-sample dataset (noisier)
 - Re-sample the dataset from D according to distribution D_t
 - Calculate E_{in} on the re-sampled dataset as usual
 - Calculate weighted error
 - $E_{in}^{(D_t)}(h) = \sum_{n=1}^N D_t(n) \text{error}(h(\vec{x}_n), y_n)$

When $D_t(n) = 1/N$. This reduces to standard definition of E_{in} .

AdaBoost – Adaptive Boosting

How to learn g_t from D_t

How to reweight the distribution and obtain D_{t+1}

How to perform weighted aggregation

[AdaBoost focuses on **classification** problem]

Boosting Background

- A theoretical question asked by Kearns and Valiant
 - Whether a “weak” learning algorithm which performs just slightly better than random guessing in the PAC model can be “boosted” into an arbitrarily accurate “strong” learning algorithm
- AdaBoost
 - The first adaptive boosting algorithm that
 - has nice theoretical guarantees
 - successfully deployed in real-world applications

What Does AdaBoost Do?

Outline of a Boosting Algorithm

Initialize D_1 (usually the same as the initial dataset D)

For $t = 1$ to T

 Learn g_t from D_t

 Reweight the distribution and obtain D_{t+1} based on g_t and D_t

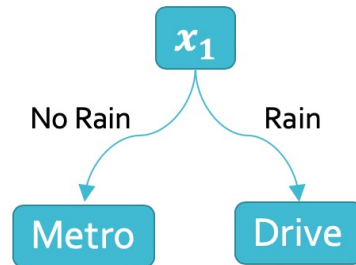
Output **weighted-aggregate**(g_1, \dots, g_T)

Classification: $G(\vec{x}) = \bar{g}(\vec{x}) = \text{sign}\left(\frac{1}{T} \sum_{t=1}^T \alpha_t g_t(\vec{x})\right)$

- Will discuss the following for AdaBoost
 1. How to learn g_t from D_t
 2. How to reweight the distribution and obtain D_{t+1}
 3. How to perform weighted aggregation

1. Learn a Weak Learner g_t from D_t

- AdaBoost uses *simple* weak learners
 - low variance, high bias
 - Decision stump (one-level decision tree) is one good option



- How to learn g_t from D_t
 - Find the decision stump that
 - Minimizes $E_{in}^{(D_t)}$
 - Maximize (weighted) information gain (you can call decision tree library directly)

2. How to Reweight D_{t+1}

- We want to make g_{t+1} (learned from D_{t+1}) to be **diverse** from g_t
 - **Increase** the weights of points that g_t makes **wrong** predictions
 - **Decrease** the weights of points that g_t makes **correct** predictions
- Define a parameter $\gamma > 1$
 - If g_t makes **wrong** predictions on \vec{x}_n
 - $D_{t+1}(n) = \frac{1}{Z_t} D_t(n) \cdot \gamma$ (increase the weight)
 - If g_t makes **correct** predictions on \vec{x}_n
 - $D_{t+1}(n) = \frac{1}{Z_t} D_t(n) / \gamma$ (decrease the weight)
- Goal:
 - Choose γ such that $E_{in}^{(D_{t+1})}(g_t) = 0.5$
 - Since g_{t+1} minimizes $E_{in}^{(D_{t+1})} \Rightarrow g_t$ and g_{t+1} are **diverse**

Z_t : normalization constant
We need to ensure
 $\sum_{n=1}^N D_{t+1}(n) = 1$

Choose γ such that $E_{in}^{(D_{t+1})}(g_t) = 0.5$

Math derivations in the next few slides

- Define $\epsilon_t = E_{in}^{(D_t)}(g_t) = \sum_{n=1}^N D_t(n) \mathbb{I}[g_t(\vec{x}_n) \neq y_n]$
 - Weighted in-sample error of g_t on D_t
 - $\epsilon_t < 0.5$ (requirement of weak learners)

We consider the case weak learners are better than random guessing:
 $\epsilon_t < 0.5$

- Goal: Want to make $E_{in}^{(D_{t+1})}(g_t) = 0.5$

$$E_{in}^{(D_t)}(h) = \sum_{n=1}^N D_t(n) \text{error}(h(\vec{x}_n), y_n)$$

- If g_t makes **wrong** predictions on \vec{x}_n
 - $D_{t+1}(n) = \frac{1}{Z_t} D_t(n) \cdot \gamma$ (increase the weight)
- If g_t makes **correct** predictions on \vec{x}_n
 - $D_{t+1}(n) = \frac{1}{Z_t} D_t(n) / \gamma$ (decrease the weight)

- Define $\epsilon_t = E_{in}^{(D_t)}(g_t) = \sum_{n=1}^N D_t(n) \mathbb{I}[g_t(\vec{x}_n) \neq y_n]$
 - Weighted in-sample error of g_t on D_t
 - $\epsilon_t < 0.5$ (requirement of weak learners)

We consider the case weak learners are better than random guessing:
 $\epsilon_t < 0.5$

- Goal: Want to make $E_{in}^{(D_{t+1})}(g_t) = 0.5$

$$\begin{aligned}
 E_{in}^{(D_{t+1})}(g_t) &= \sum_{n=1}^N D_{t+1}(n) \mathbb{I}[g_t(\vec{x}_n) \neq y_n] \\
 &= \sum_{n=1}^N \frac{1}{Z_t} D_t(n) \gamma \mathbb{I}[g_t(\vec{x}_n) \neq y_n] \\
 &= \frac{\gamma}{Z_t} \sum_{n=1}^N D_t(n) \mathbb{I}[g_t(\vec{x}_n) \neq y_n] = \frac{\gamma}{Z_t} \epsilon_t
 \end{aligned}$$

$$E_{in}^{(D_t)}(h) = \sum_{n=1}^N D_t(n) \text{error}(h(\vec{x}_n), y_n)$$

- If g_t makes **wrong** predictions on \vec{x}_n
 - $D_{t+1}(n) = \frac{1}{Z_t} D_t(n) \cdot \gamma$ (increase the weight)
- If g_t makes **correct** predictions on \vec{x}_n
 - $D_{t+1}(n) = \frac{1}{Z_t} D_t(n) / \gamma$ (decrease the weight)

- Remember Z_t is the normalization constant

$$\begin{aligned}
 Z_t &= \sum_{n=1}^N D_t(n) \gamma \mathbb{I}[g_t(\vec{x}_n) \neq y_n] + \sum_{n=1}^N D_t(n) \frac{1}{\gamma} \mathbb{I}[g_t(\vec{x}_n) = y_n] \\
 &= \gamma \epsilon_t + \frac{1}{\gamma} (1 - \epsilon_t)
 \end{aligned}$$

- Want to make $E_{in}^{(D_{t+1})}(g_t) = 0.5$
 - $E_{in}^{(D_{t+1})}(g_t) = \frac{\gamma}{Z_t} \epsilon_t$
 - $Z_t = \gamma \epsilon_t + \frac{1}{\gamma} (1 - \epsilon_t)$

- Want to make $E_{in}^{(D_{t+1})}(g_t) = 0.5$

- $E_{in}^{(D_{t+1})}(g_t) = \frac{\gamma}{Z_t} \epsilon_t$

- $Z_t = \gamma \epsilon_t + \frac{1}{\gamma} (1 - \epsilon_t)$

- $\frac{\gamma \epsilon_t}{\gamma \epsilon_t + (1 - \epsilon_t)/\gamma} = 0.5 \Rightarrow \frac{1 - \epsilon_t}{\gamma} = \gamma \epsilon_t \Rightarrow \gamma = \sqrt{\frac{1 - \epsilon_t}{\epsilon_t}}$

Both $g_t(\vec{x}_n)$ and y_n are either +1 or -1
 If $g_t(\vec{x}_n) \neq y_n$, $g_t(\vec{x}_n)y_n = -1$
 If $g_t(\vec{x}_n) = y_n$, $g_t(\vec{x}_n)y_n = 1$

- The rule for reweighting

- If $g_t(\vec{x}_n) \neq y_n$, then $D_{t+1}(n) = \frac{1}{Z_t} D_t(n) \left(\sqrt{\frac{1 - \epsilon_t}{\epsilon_t}} \right)$

- If $g_t(\vec{x}_n) = y_n$, then $D_{t+1}(n) = \frac{1}{Z_t} D_t(n) \left(\sqrt{\frac{1 - \epsilon_t}{\epsilon_t}} \right)^{-1}$

- Want to make $E_{in}^{(D_{t+1})}(g_t) = 0.5$

- $E_{in}^{(D_{t+1})}(g_t) = \frac{\gamma}{Z_t} \epsilon_t$

- $Z_t = \gamma \epsilon_t + \frac{1}{\gamma} (1 - \epsilon_t)$

- $\frac{\gamma \epsilon_t}{\gamma \epsilon_t + (1 - \epsilon_t)/\gamma} = 0.5 \Rightarrow \frac{1 - \epsilon_t}{\gamma} = \gamma \epsilon_t \Rightarrow \gamma = \sqrt{\frac{1 - \epsilon_t}{\epsilon_t}}$

- The rule for reweighting

- If $g_t(\vec{x}_n) \neq y_n$, then $D_{t+1}(n) = \frac{1}{Z_t} D_t(n) \left(\sqrt{\frac{1 - \epsilon_t}{\epsilon_t}} \right) = \frac{1}{Z_t} D_t(n) \left(\sqrt{\frac{1 - \epsilon_t}{\epsilon_t}} \right)^{-g_t(\vec{x}_n) y_n}$

- If $g_t(\vec{x}_n) = y_n$, then $D_{t+1}(n) = \frac{1}{Z_t} D_t(n) \left(\sqrt{\frac{1 - \epsilon_t}{\epsilon_t}} \right)^{-1} = \frac{1}{Z_t} D_t(n) \left(\sqrt{\frac{1 - \epsilon_t}{\epsilon_t}} \right)^{-g_t(\vec{x}_n) y_n}$

- Reweight rule: $D_{t+1}(n) = \frac{1}{Z_t} D_t(n) \left(\sqrt{\frac{1 - \epsilon_t}{\epsilon_t}} \right)^{-g_t(\vec{x}_n) y_n}$

2. How to Reweight D_{t+1} : Summary

- Reweight rule:

- $D_{t+1}(n) = \frac{1}{Z_t} D_t(n) \left(\sqrt{\frac{1-\epsilon_t}{\epsilon_t}} \right)^{-g_t(\vec{x}_n)y_n}$

- A bit more manipulations (the reason will be clear later)

- Define $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$

- $e^{\alpha_t} = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$

- Final reweight rule: $D_{t+1}(n) = \frac{1}{Z_t} D_t(n) e^{-\alpha_t g_t(\vec{x}_n)y_n}$

3. How to Aggregate Weak Learners

- Intuition:
 - We want to put more weights on better weak learners
 - $\epsilon_t = E_{in}^{(D_t)}(g_t)$ is a proxy on how well g_t performs (smaller $\epsilon_t \Rightarrow$ better g_t)
 - Recall that $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
 - Better g_t , smaller ϵ_t , higher α_t
 - When $\epsilon_t = 0.5$, $\alpha_t = 0$ (random guessing leads to 0 weights)
 - When $\epsilon_t = 0$, $\alpha_t = \infty$ (if a feature perfectly classifies the data, use it as our final hypothesis)
- Aggregation rule
 - $G(\vec{x}) = \text{sign}(\sum_{t=1}^T \alpha_t g_t(\vec{x}))$

AdaBoost Algorithm

- Given $D = \{(\vec{x}_1, y_1), \dots, (\vec{x}_N, y_N)\}$
- Initialize $D_1(n) = 1/N$ for all $n = 1, \dots, N$
- For $t = 1, \dots, T$
 - Learn g_t from D_t (using decision stumps)
 - Calculate $\epsilon_t = E_{in}^{(D_t)}(g_t)$
 - Set $\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$
 - Update $D_{t+1}(n) = \frac{1}{Z_t} D_t(n) e^{-\alpha_t y_n g_t(\vec{x}_n)}$
- Output $G(\vec{x}) = \text{sign}(\sum_{t=1}^T \alpha_t g_t(\vec{x}))$

Theoretical Properties of AdaBoost

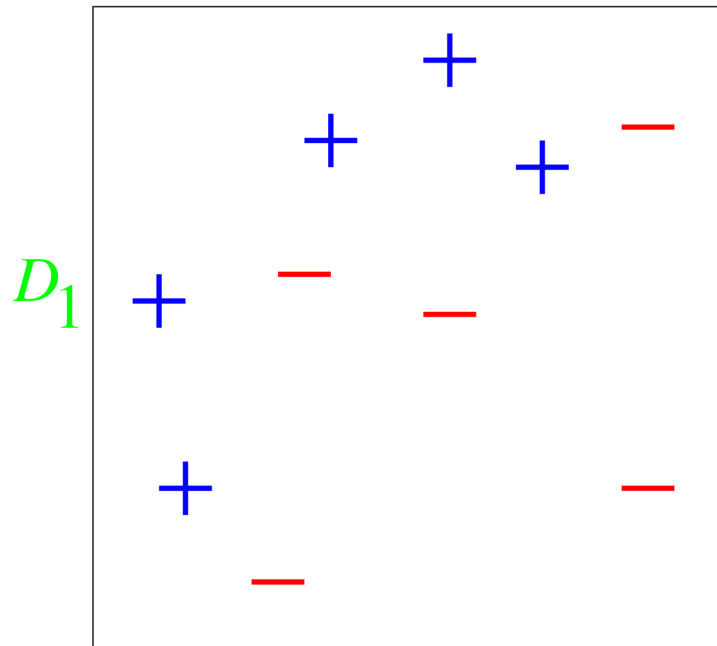
- See [Freund & Schapire's Tutorial](#) for more discussion
- The training error of AdaBoost converges fast
 - Let $\gamma_t = \frac{1}{2} - \epsilon_t$ (how good each weak learner is better than random guessing)
 - $E_{in} \leq e^{-2 \sum_{t=1}^T \gamma_t^2}$
- Generalization error
 - VC analysis gives us $E_{out} \leq E_{in} + \tilde{O}\left(\sqrt{\frac{T d_{vc}}{m}}\right)$
 - It seems as T goes large, overfitting could happen
 - Empirically, AdaBoost is relatively robust to overfitting
 - There are some more delicate analysis using the idea of **margins** to explain why

d_{vc} is the VC dimension
of the weak learner

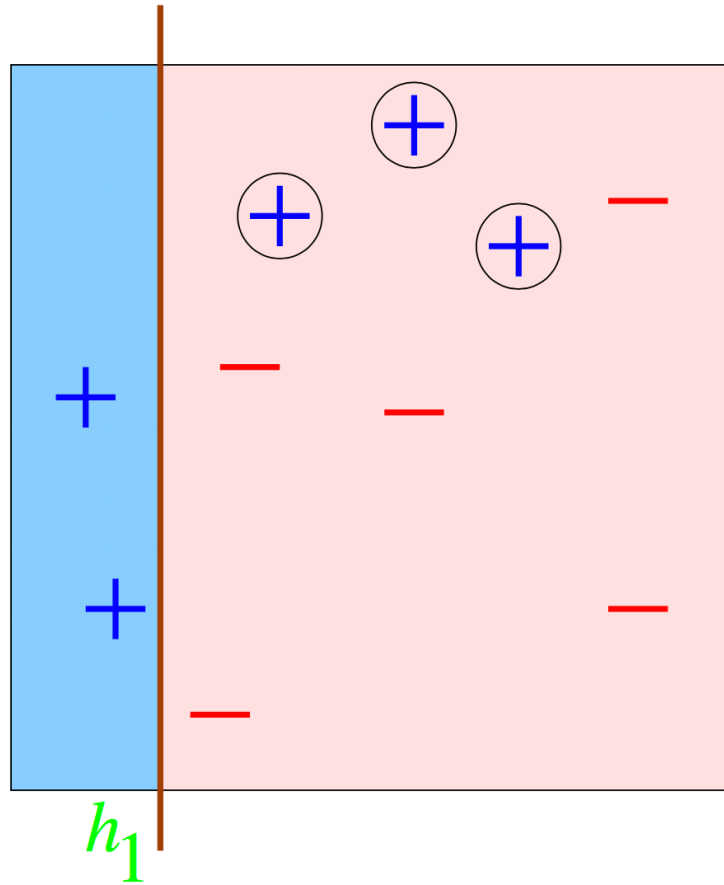
AdaBoost in Action

AdaBoost in Action

- A toy example (by Yoav Freund Rob Schapire)
- Weak learner: decision stump (one-level decision tree)

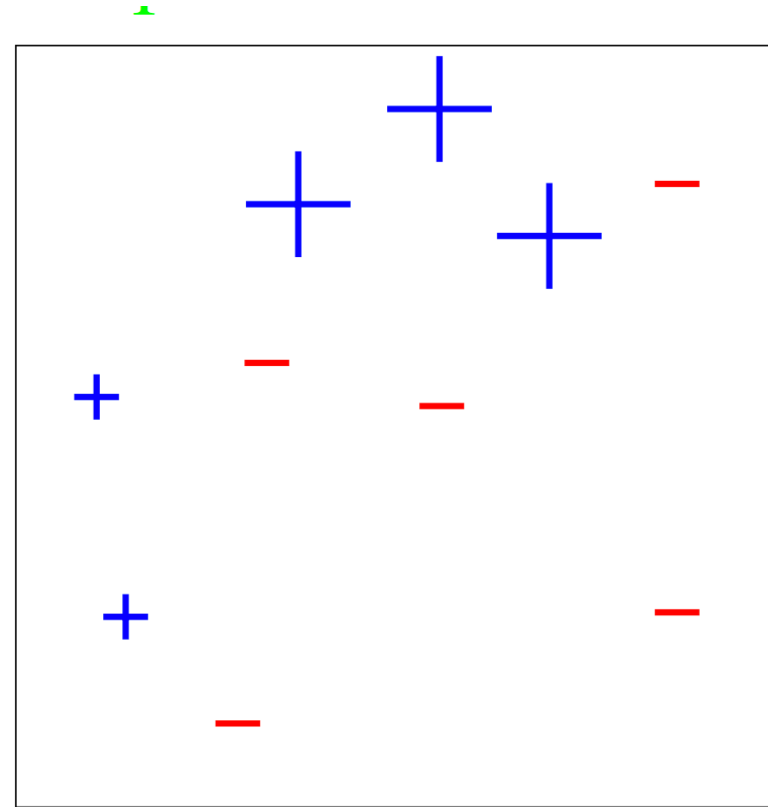


Round 1

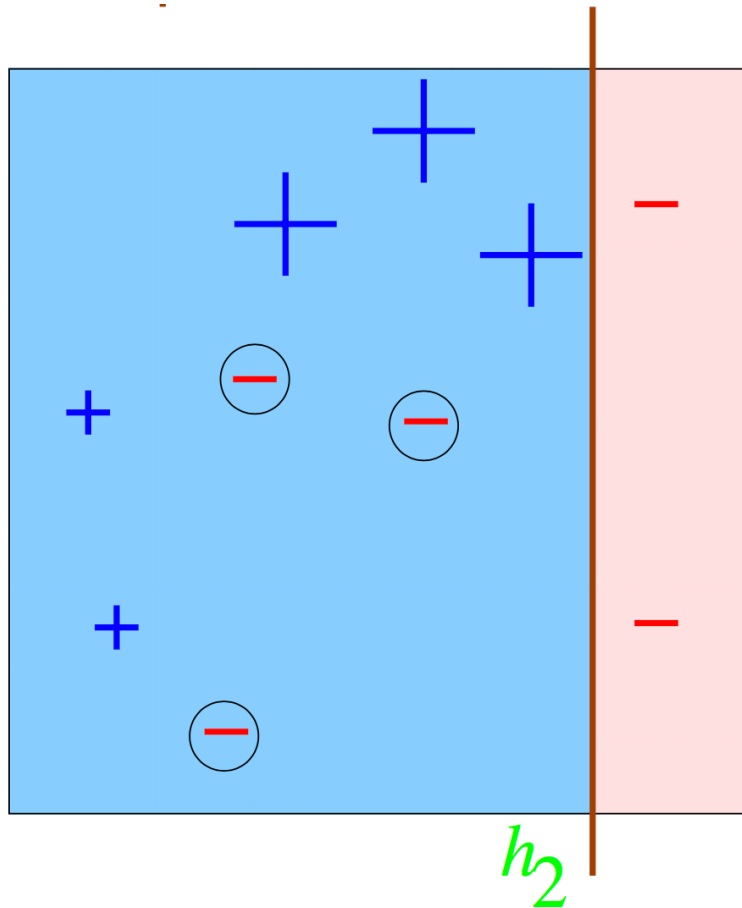


$$\epsilon_1 = 0.30$$
$$\alpha_1 = 0.42$$

D_2

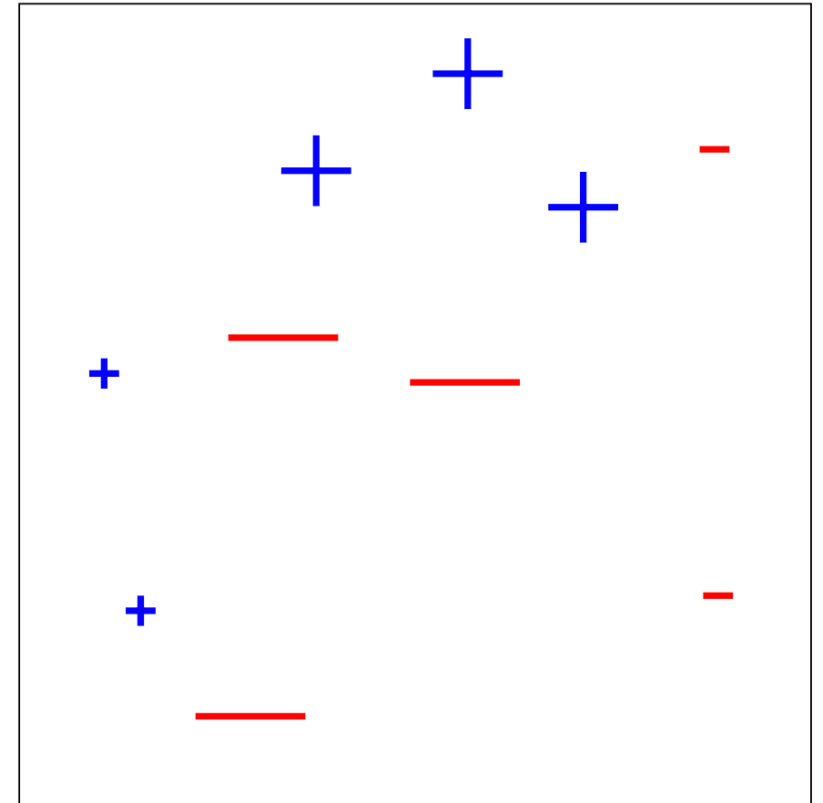


Round 2

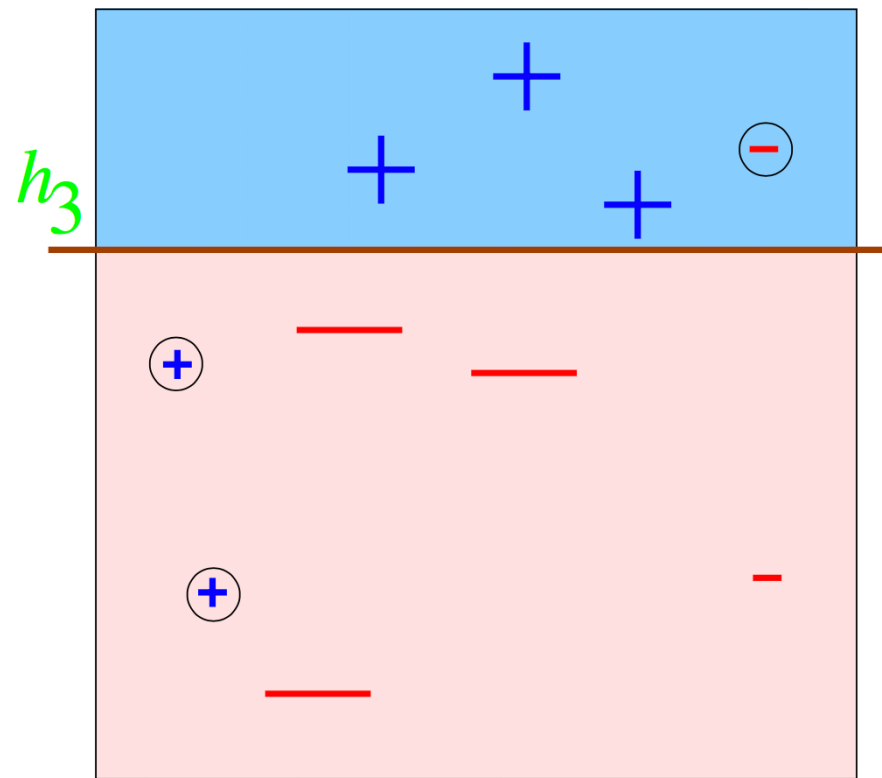


$$\epsilon_2 = 0.21$$
$$\alpha_2 = 0.65$$

D_3

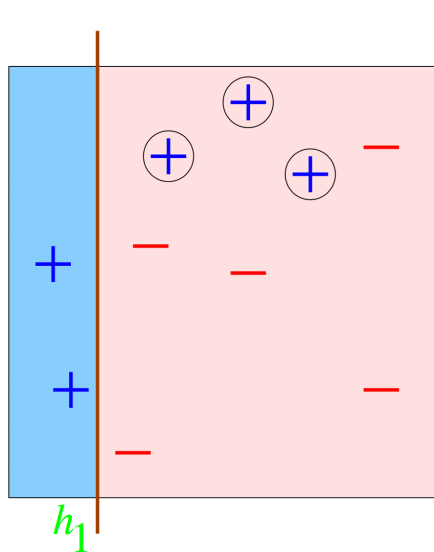


Round 3



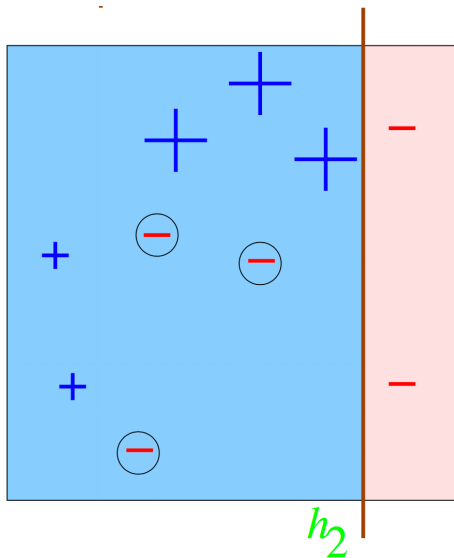
$$\epsilon_3 = 0.14$$

$$\alpha_3 = 0.92$$



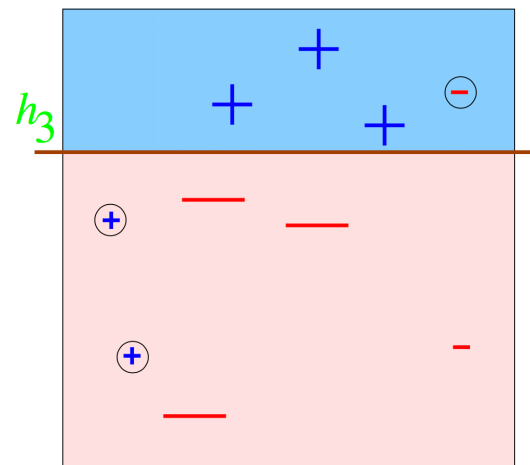
$$\varepsilon_1 = 0.30$$

$$\alpha_1 = 0.42$$



$$\varepsilon_2 = 0.21$$

$$\alpha_2 = 0.65$$



$$\varepsilon_3 = 0.14$$

$$\alpha_3 = 0.92$$

H_{final}

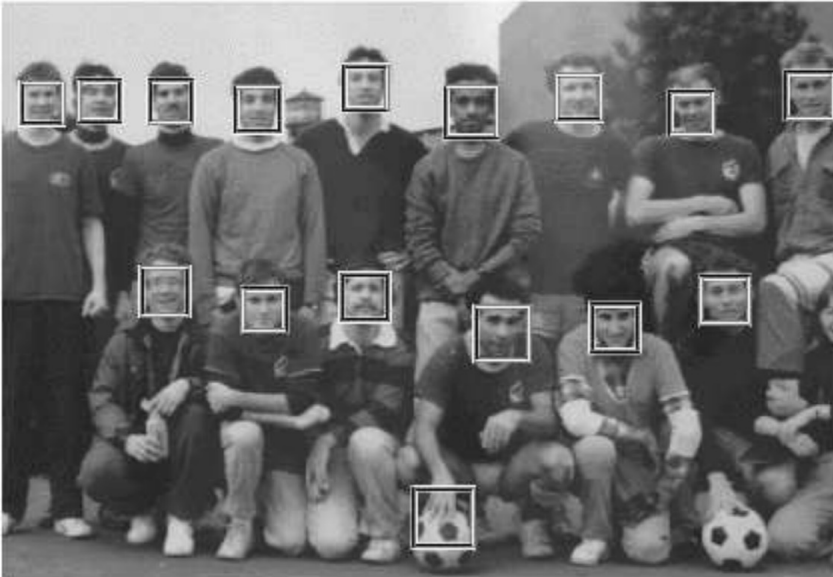
$$= \text{sign} \left(\begin{array}{c} 0.42 \\ + 0.65 \\ + 0.92 \end{array} \right) =$$

The final diagram shows a 2D space partitioned by two vertical lines and one horizontal line. The regions are colored blue or pink based on the sign of the sum of the three terms in the parentheses. The resulting partitioning is identical to the one in Diagram 3.

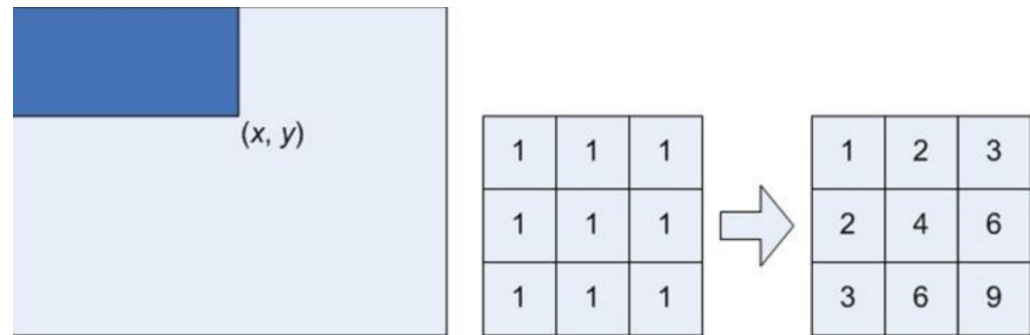
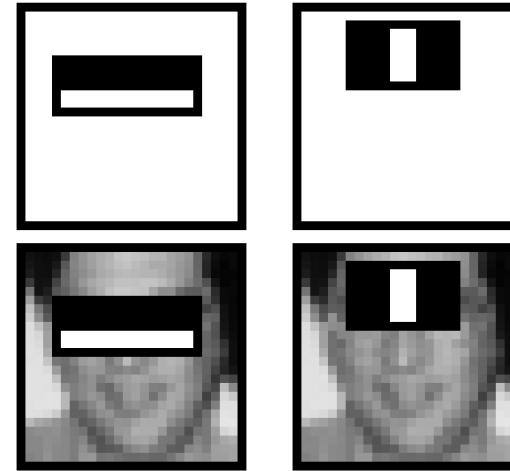
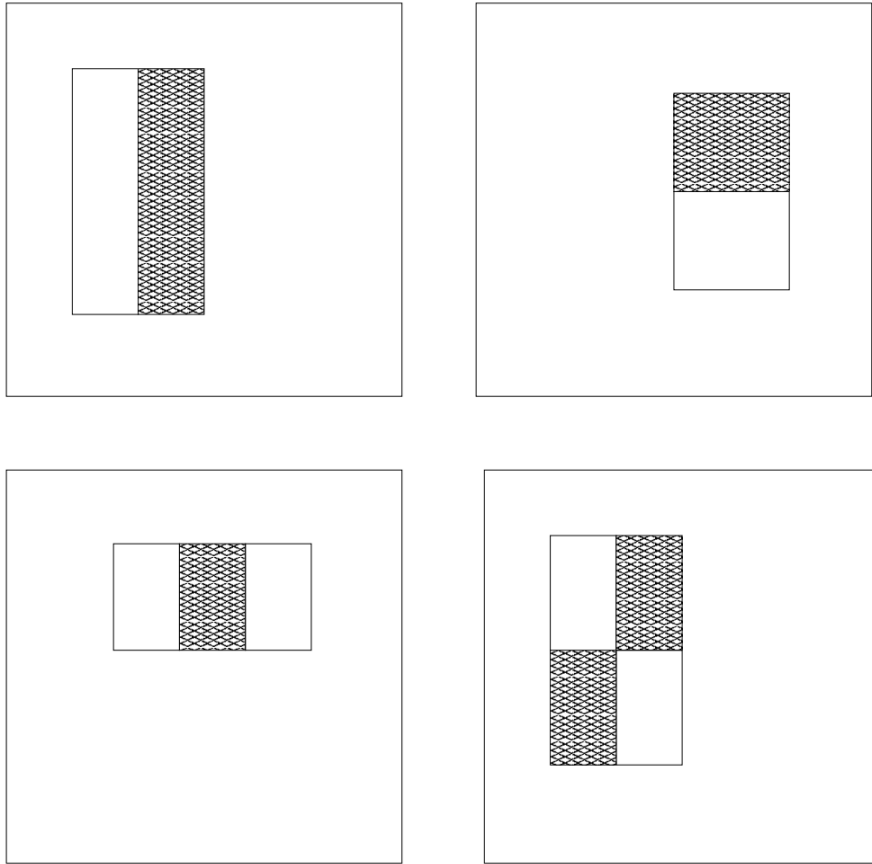
Practical Success of AdaBoost

Viola-Jones Face Detection (2001)

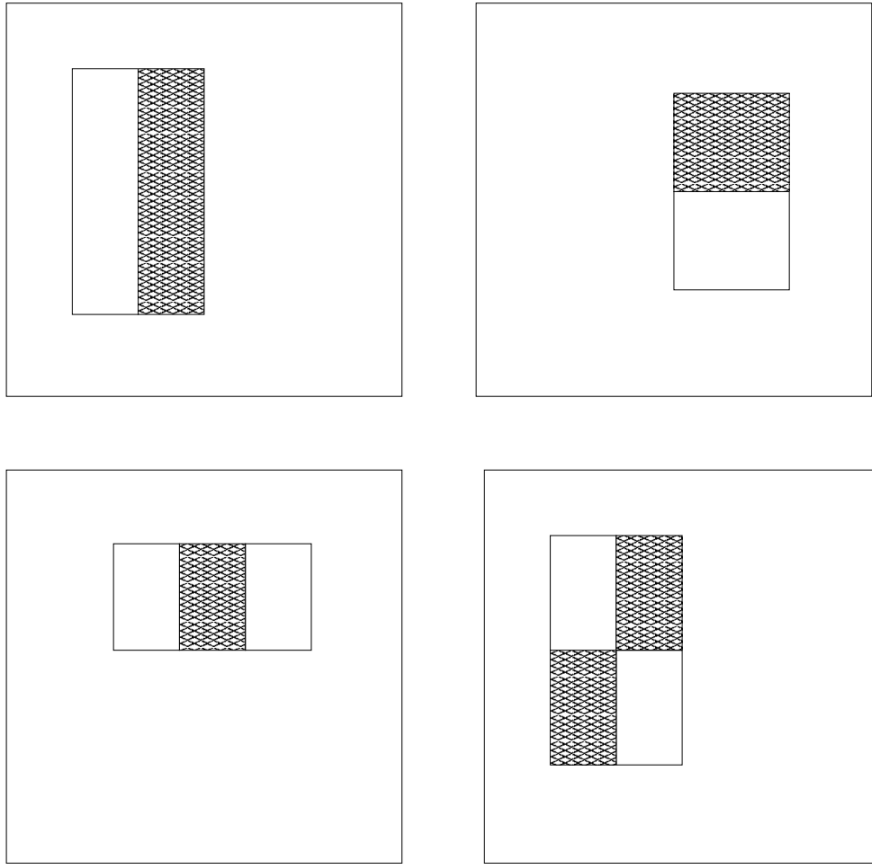
- First real-time object detection framework
- Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. CVPR 2001.



Weak Learners (Haar wavelet features)



Weak Learners (Haar wavelet features)



- Each hypothesis is very weak.
- There are many possible features.
 - For a 24x24 detection region, more than 160,000 features
- AdaBoost!
 - Training is slow
 - Testing is fast
 - (inherent feature selection)

Brief Discussion on Gradient Boosting

Gradient boosting is **safe to skip** for Exam 2

Look at the AdaBoost Algorithm Again

Given $D = \{(\vec{x}_1, y_1), \dots, (\vec{x}_N, y_N)\}$
Initialize $D_1(n) = 1/N$ for all $n = 1, \dots, N$
For $t = 1, \dots, T$
 Learn g_t from D_t (using decision stumps)
 Calculate $\epsilon_t = E_{in}^{(D_t)}(g_t)$
 Set $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
 Update $D_{t+1}(n) = \frac{1}{Z_t} D_t(n) e^{-\alpha_t y_n g_t(\vec{x}_n)}$
Output $G(\vec{x}) = \text{sign}(\sum_{t=1}^T \alpha_t g_t(\vec{x}))$



Initialize $G(\vec{x}) = 0$
For $t = 1, \dots, T$
 $G(\vec{x}) \leftarrow G(\vec{x}) + \alpha_t g_T(\vec{x})$
Output $\text{sign}(G(\vec{x}))$

- The format is similar to **gradient descent**!
 - If we consider the space of the weak learners (i.e., $g_t(\vec{x})$) as the space of “weights”
 - This observation leads to a general class of boosting algorithms: **gradient boosting**
 - XGBoost is one implementation of gradient boosting that is popular in practice
 - See CASI 17.4 and the reference in CASI P.350 for more discussion

[Safe to Skip]

Gradient Boosting

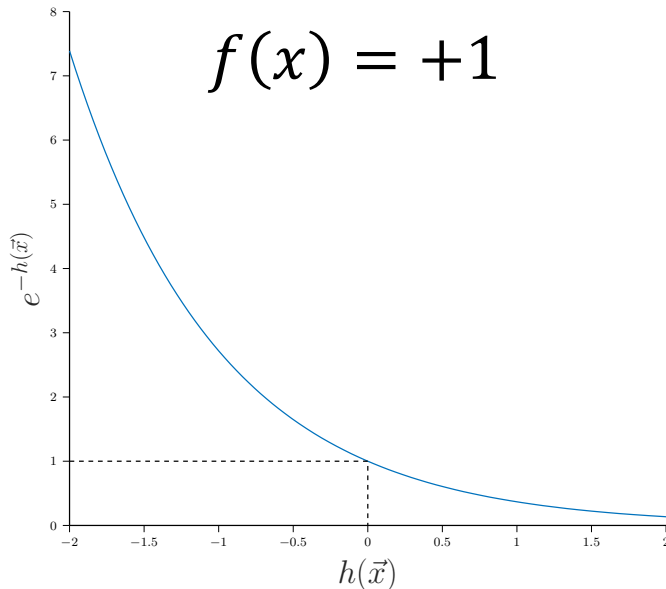
Initialize $G(\vec{x}) = 0$

For $t = 1, \dots, T$

$$G(\vec{x}) \leftarrow G(\vec{x}) + \alpha_t g_T(\vec{x})$$

Output $\text{sign}(G(\vec{x}))$

- AdaBoost is a special case of Gradient Boosting
 - minimizing the exponential **loss** ($e_{\text{exp}}(h(\vec{x}), y) = e^{-yh(\vec{x})}$)
 - using decision stump as the **weak learners**



- e_{exp} is a **surrogate loss function** of the binary classification error we care about
 - Minimizing an alternative error (loss function) is a common trick in ML, especially when the target loss function is hard to optimize.
 - There are some theoretical discussions on when doing this makes sense (“calibration”: whether minimizing the surrogate is consistent with minimizing the original loss).

[Safe to Skip]