

CSCI567 Machine Learning (Fall 2020)

Prof. Haipeng Luo

U of Southern California

Oct 15, 2020

1 / 47

Outline

- 1 Review of last lecture
- 2 Decision tree
- 3 Boosting

3 / 47

Administration

Quiz 1 grading is done:

- Mean: 84.7, Median: 92
- discuss solutions today
- regrade requests by the end of Sat (10/17)

HW3 is due on Tue (10/20)

2 / 47

Review of last lecture

Outline

- 1 Review of last lecture
- 2 Decision tree
- 3 Boosting

4 / 47

Support Vector Machine

SVM: **max-margin linear classifier**

Primal (equivalent to minimizing L2 regularized hinge loss):

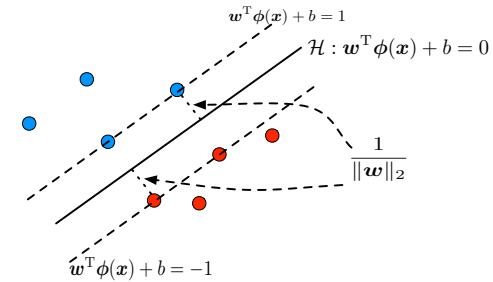
$$\begin{aligned} \min_{\mathbf{w}, b, \{\xi_n\}} \quad & C \sum_n \xi_n + \frac{1}{2} \|\mathbf{w}\|_2^2 \\ \text{s.t.} \quad & 1 - y_n(\mathbf{w}^T \phi(\mathbf{x}_n) + b) \leq \xi_n, \quad \forall n \\ & \xi_n \geq 0, \quad \forall n \end{aligned}$$

Dual (kernelizable, reveals what training points are support vectors):

$$\begin{aligned} \max_{\{\alpha_n\}} \quad & \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n \phi(\mathbf{x}_m)^T \phi(\mathbf{x}_n) \\ \text{s.t.} \quad & \sum_n \alpha_n y_n = 0 \quad \text{and} \quad 0 \leq \alpha_n \leq C, \quad \forall n \end{aligned}$$

5 / 47

Separable Case



6 / 47

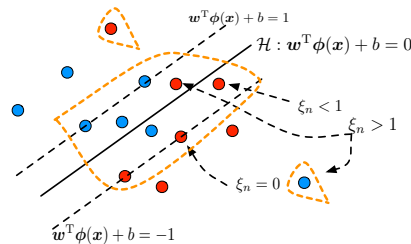
Geometric interpretation of support vectors

A support vector satisfies $\alpha_n^* \neq 0$ and

$$1 - \xi_n^* - y_n(\mathbf{w}^{*T} \phi(\mathbf{x}_n) + b^*) = 0$$

When

- $\xi_n^* = 0$, $y_n(\mathbf{w}^{*T} \phi(\mathbf{x}_n) + b^*) = 1$ and thus the point is $1/\|\mathbf{w}^*\|_2$ away from the hyperplane.
- $\xi_n^* < 1$, the point is classified correctly but does not satisfy the large margin constraint.
- $\xi_n^* > 1$, the point is misclassified.



Support vectors (circled with the orange line) are **the only points that matter!**

7 / 47

The Karush-Kuhn-Tucker (KKT) conditions

If \mathbf{w}^* and $\{\lambda_j^*\}$ are the primal and dual solution respectively, then:

Stationarity:

$$\nabla_{\mathbf{w}} L(\mathbf{w}^*, \{\lambda_j^*\}) = \nabla F(\mathbf{w}^*) + \sum_{j=1}^J \lambda_j^* \nabla h_j(\mathbf{w}^*) = \mathbf{0}$$

Complementary slackness:

$$\lambda_j^* h_j(\mathbf{w}^*) = 0 \quad \text{for all } j \in [J]$$

Feasibility:

$$h_j(\mathbf{w}^*) \leq 0 \quad \text{and} \quad \lambda_j^* \geq 0 \quad \text{for all } j \in [J]$$

These are **necessary conditions**. They are also **sufficient** when F is convex and h_1, \dots, h_J are continuously differentiable convex functions.

8 / 47

Outline

- 1 Review of last lecture
- 2 Decision tree
 - The model
 - Learning a decision tree
- 3 Boosting

Decision tree

We have seen different ML models for classification/regression:

- linear models, neural nets and other nonlinear models induced by kernels

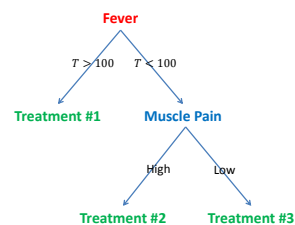
Decision tree is yet another one:

- **nonlinear** in general
- works for both classification and regression; we focus on **classification**
- one key advantage is good **interpretability**
- used to be very popular; ensemble of trees (i.e. “**forest**”) can still be very effective
- not to be confused with the “tree reduction” in Lec 4

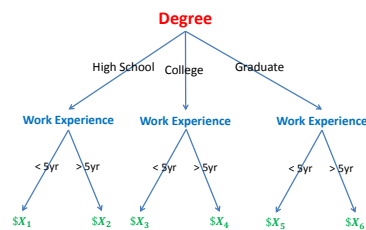
Example

Many decisions are made based on some tree structure

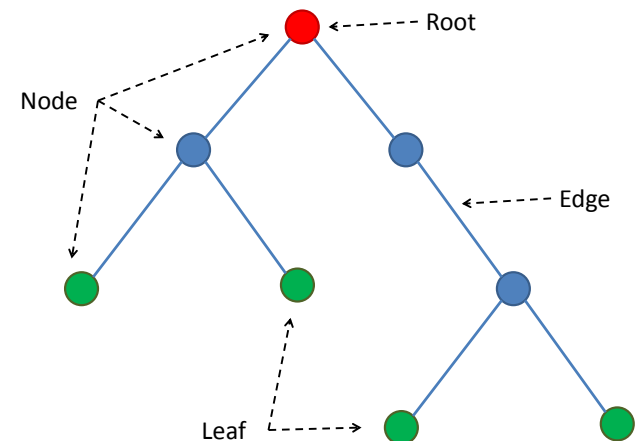
Medical treatment



Salary in a company



Tree terminology

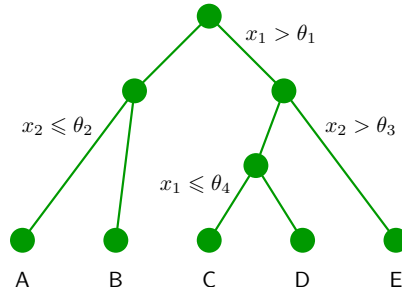


A more abstract example of decision trees

Input: $\mathbf{x} = (x_1, x_2)$

Output: $f(\mathbf{x})$ determined naturally by **traversing** the tree

- start from the root
- test at each node to decide which child to visit next
- finally the leaf gives the prediction $f(\mathbf{x})$

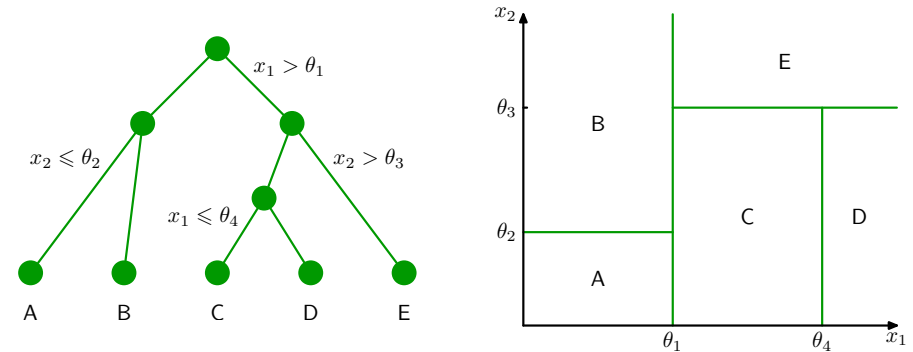


For example, $f((\theta_1 - 1, \theta_2 + 1)) = B$

Complex to formally write down, but **easy to represent pictorially or as codes**.

The decision boundary

Corresponds to a classifier with boundaries:



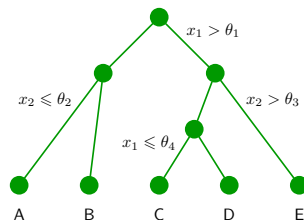
Parameters

Parameters to learn for a decision tree:

- the **structure** of the tree, such as the depth, #branches, #nodes, etc
 - some of them are sometimes considered as hyperparameters
 - unlike typical neural nets, the structure of a tree is **not fixed in advance, but learned from data**

- the **test** at each internal node

- which **feature(s)** to test on?
- if the feature is continuous, what **threshold** ($\theta_1, \theta_2, \dots$)?



- the **value/prediction** of the leaves (A, B, ...)

Learning the parameters

So how do we **learn all these parameters**?

Recall typical approach is to find the parameters that **minimize some loss**.

This is unfortunately **not feasible for trees**

- suppose there are Z nodes, there are roughly $\#features^Z$ different ways to decide “which feature to test on each node”, which is **a lot**.
- enumerating all these configurations to find the one that minimizes some loss is too computationally expensive.

Instead, we turn to some **greedy top-down approach**.

A running example

[Russell & Norvig, AIMA]

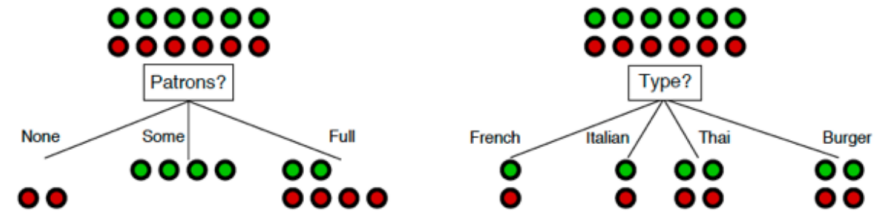
- predict whether a customer will wait for a table at a restaurant
- 12 training examples
- 10 features (all discrete)

Example	Attributes										Target
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	WillWait
X_1	T	F	F	T	Some	\$\$\$	F	T	French	0-10	T
X_2	T	F	F	T	Full	\$	F	F	Thai	30-60	F
X_3	F	T	F	F	Some	\$	F	F	Burger	0-10	T
X_4	T	F	T	T	Full	\$	F	F	Thai	10-30	T
X_5	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
X_6	F	T	F	T	Some	\$\$	T	T	Italian	0-10	T
X_7	F	T	F	F	None	\$	T	F	Burger	0-10	F
X_8	F	F	F	T	Some	\$\$	T	T	Thai	0-10	T
X_9	F	T	T	F	Full	\$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	\$\$\$	F	T	Italian	10-30	F
X_{11}	F	F	F	F	None	\$	F	F	Thai	0-10	F
X_{12}	T	T	T	T	Full	\$	F	F	Burger	30-60	T

17 / 47

First step: how to build the root?

I.e., which feature should we test at the root? Examples:



Which split is better?

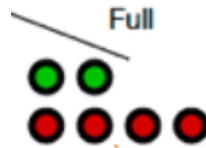
- intuitively “patrons” is a better feature since it leads to “more pure” or “more certain” children
- how to quantify this intuition?

18 / 47

Measure of uncertainty of a node

It should be a function of the distribution of classes

- e.g. a node with 2 positive and 4 negative examples can be summarized by a distribution P with $P(Y = +1) = 1/3$ and $P(Y = -1) = 2/3$



One classic uncertainty measure of a distribution is its (Shannon) entropy:

$$H(P) = - \sum_{k=1}^C P(Y = k) \log P(Y = k)$$

19 / 47

Properties of entropy

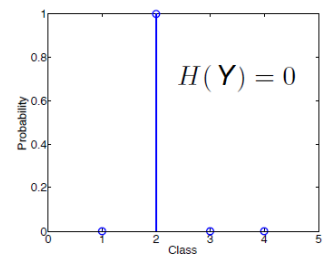
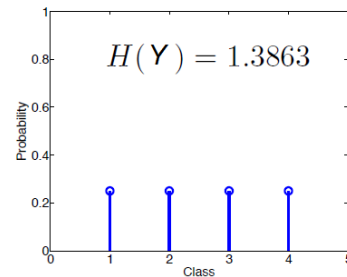
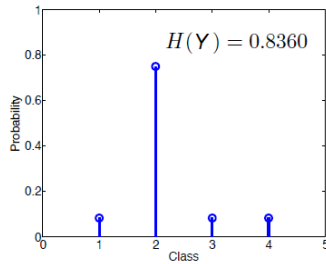
$$H(P) = - \sum_{k=1}^C P(Y = k) \log P(Y = k)$$

- the base of log can be 2, e or 10
- always non-negative
- it's the *smallest codeword length to encode symbols drawn from P*
- maximized if P is uniform (max = $\ln C$): most uncertain case
- minimized if P focuses on one class (min = 0): most certain case
 - e.g. $P = (1, 0, \dots, 0)$
 - $0 \log 0$ is defined naturally as $\lim_{z \rightarrow 0^+} z \log z = 0$

20 / 47

Examples of computing entropy

With base e and 4 classes:



21 / 47

Another example

Entropy in each child if root tests on “patrons”

For “None” branch

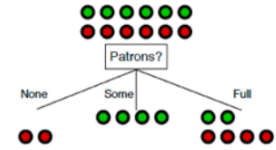
$$-\left(\frac{0}{0+2} \log \frac{0}{0+2} + \frac{2}{0+2} \log \frac{2}{0+2}\right) = 0$$

For “Some” branch

$$-\left(\frac{4}{4+0} \log \frac{4}{4+0} + \frac{0}{4+0} \log \frac{0}{4+0}\right) = 0$$

For “Full” branch

$$-\left(\frac{2}{2+4} \log \frac{2}{2+4} + \frac{4}{2+4} \log \frac{4}{2+4}\right) \approx 0.9$$



So how good is choosing “patrons” overall?

Very naturally, we take the **weighted average of entropy**:

$$\frac{2}{12} \times 0 + \frac{4}{12} \times 0 + \frac{6}{12} \times 0.9 = 0.45$$

22 / 47

Measure of uncertainty of a split

Suppose we split based on a discrete feature A , the uncertainty can be measured by the **conditional entropy**:

$$\begin{aligned} H(Y | A) &= \sum_a P(A = a) H(Y | A = a) \\ &= \sum_a P(A = a) \left(- \sum_{k=1}^C P(Y | A = a) \log P(Y | A = a) \right) \\ &= \sum_a \text{“fraction of example at node } A = a\text{”} \times \text{“entropy at node } A = a\text{”} \end{aligned}$$

Pick the feature that leads to the smallest conditional entropy.

23 / 47

Deciding the root

For “French” branch

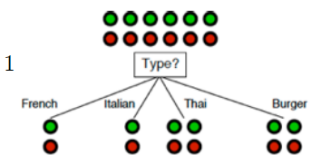
$$-\left(\frac{1}{1+1} \log \frac{1}{1+1} + \frac{1}{1+1} \log \frac{1}{1+1}\right) = 1$$

For “Italian” branch

$$-\left(\frac{1}{1+1} \log \frac{1}{1+1} + \frac{1}{1+1} \log \frac{1}{1+1}\right) = 1$$

For “Thai” and “Burger” branches

$$-\left(\frac{2}{2+2} \log \frac{2}{2+2} + \frac{2}{2+2} \log \frac{2}{2+2}\right) = 1$$



The conditional entropy is $\frac{2}{12} \times 1 + \frac{2}{12} \times 1 + \frac{4}{12} \times 1 + \frac{4}{12} \times 1 = 1 > 0.45$

So splitting with “patrons” is better than splitting with “type”.

In fact by similar calculation **“patrons” is the best split** among all features.

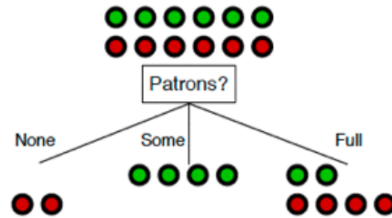
We are now done with building the root (this is also called a **stump**).

24 / 47

Repeat recursively

Split each child in the same way.

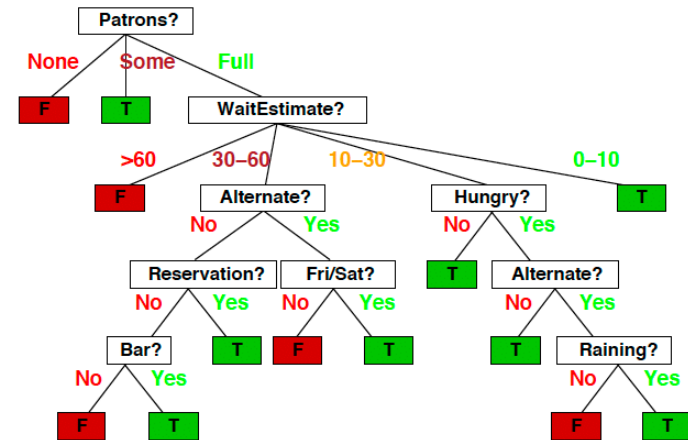
- but no need to split children “none” and “some”: they are pure already and become leaves
- for “full”, repeat, focusing on those 6 examples:



	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	WillWait
X ₁	T	F	F	T	Some	\$\$\$	F	T	French	0-10	T
X ₂	T	F	F	T	Full	\$	F	F	Thai	30-60	F
X ₃	F	T	F	F	Some	\$	F	F	Burger	0-10	T
X ₄	T	F	T	T	Full	\$	F	F	Thai	10-30	T
X ₅	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
X ₆	F	T	F	T	Some	\$\$	T	T	Italian	0-10	T
X ₇	F	T	F	F	None	\$	T	F	Burger	0-10	F
X ₈	F	F	F	T	Some	\$\$	T	T	Thai	0-10	T
X ₉	F	T	T	F	Full	\$	T	F	Burger	>60	F
X ₁₀	T	T	T	T	Full	\$\$\$	F	T	Italian	10-30	F
X ₁₁	F	F	F	F	None	\$	F	F	Thai	0-10	F
X ₁₂	T	T	T	T	Full	\$	F	F	Burger	30-60	T

25 / 47

Greedily we build the tree and get this



Again, very easy to interpret.

26 / 47

Putting it together

DecisionTreeLearning(Examples, Features)

- if **Examples** have the same class, return a leaf with this class
- else if **Features** is empty, return a leaf with the majority class
- else if **Examples** is empty, return a leaf with majority class of parent
- else

find the best feature A to split (e.g. based on conditional entropy)

Tree \leftarrow a root with test on A

For each value a of A :

Child \leftarrow **DecisionTreeLearning**(Examples with $A = a$, $\text{Features} \setminus \{A\}$)
add **Child** to **Tree** as a new branch

- return **Tree**

27 / 47

Variants

Popular decision tree algorithms (e.g. C4.5, CART, etc) are all based on this framework.

Variants:

- replace entropy by **Gini impurity**:

$$G(P) = \sum_{k=1}^C P(Y = k)(1 - P(Y = k))$$

meaning: *how often a randomly chosen example would be incorrectly classified if we predict according to another randomly picked example*

- if a feature is continuous, we need to find a **threshold** that leads to minimum conditional entropy or Gini impurity. *Think about how to do it efficiently.*

28 / 47

Regularization

If the dataset has no contradiction (i.e. same x but different y), the training error of a tree is always zero, which might indicate **overfitting**.

Pruning is a typical way to prevent overfitting for a tree:

- restrict the depth or #nodes
- other more principled approaches
- all make use of a validation set

29 / 47

Outline

- 1 Review of last lecture
- 2 Decision tree
- 3 Boosting
 - Examples
 - AdaBoost
 - Derivation of AdaBoost

30 / 47

Introduction

Boosting

- is a **meta-algorithm**, which takes a base algorithm (classification, regression, ranking, etc) as input and **boosts** its accuracy
- main idea: combine **weak “rules of thumb”** (e.g. 51% accuracy) to form a **highly accurate predictor** (e.g. 99% accuracy)
- works very well in practice (especially in combination with trees)
- often is **resistant to overfitting**
- has strong theoretical guarantees

We again focus on **binary classification**.

31 / 47

A simple example

Email spam detection:

- given a training set like:
 - (“Want to make money fast? ...”, **spam**)
 - (“Viterbi Research Gist ...”, **not spam**)
- first obtain a classifier by applying a **base algorithm**, which can be a rather simple/weak one, like decision stumps:
 - e.g. contains the word “money” \Rightarrow spam
- **reweight** the examples so that “**difficult**” ones get more attention
 - e.g. spam that doesn’t contain the word “money”
- obtain **another classifier** by applying the same base algorithm:
 - e.g. empty “to address” \Rightarrow spam
- repeat ...
- final classifier is the **(weighted) majority vote** of all weak classifiers

32 / 47

The base algorithm

A **base algorithm** \mathcal{A} (also called weak learning algorithm/oracle) takes a **training set** S **weighted by** D as input, and outputs classifier $h \leftarrow \mathcal{A}(S, D)$

- this can be **any off-the-shelf classification algorithm** (e.g. decision trees, logistic regression, neural nets, etc)
- many algorithms can deal with a **weighted training set** (e.g. for algorithm that minimizes some loss, we can simply **replace** “total loss” by “weighted total loss”)
- even if it's not obvious how to deal with weight directly, we can always **resample according to** D to create a new unweighted dataset

33 / 47

Boosting Algorithms

Given:

- a training set S
- a base algorithm \mathcal{A}

Two things to specify a boosting algorithm:

- how to **reweight** the examples?
- how to **combine** all the weak classifiers?

AdaBoost is one of the most successful boosting algorithms.

34 / 47

The AdaBoost Algorithm

Given a training set S and a base algorithm \mathcal{A} , **initialize** D_1 to be uniform

For $t = 1, \dots, T$

- obtain a weak classifier $h_t \leftarrow \mathcal{A}(S, D_t)$
- **calculate the importance** of h_t as

$$\beta_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \quad (\beta_t > 0 \Leftrightarrow \epsilon_t < 0.5)$$

where $\epsilon_t = \sum_{n: h_t(x_n) \neq y_n} D_t(n)$ is the **weighted error of** h_t .

- **update weights**

$$D_{t+1}(n) \propto D_t(n) e^{-\beta_t y_n h_t(x_n)} = \begin{cases} D_t(n) e^{-\beta_t} & \text{if } h_t(x_n) = y_n \\ D_t(n) e^{\beta_t} & \text{else} \end{cases}$$

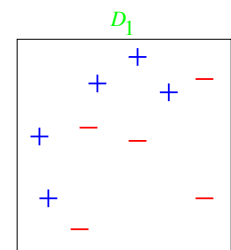
Output the **final classifier** $H(x) = \text{sgn} \left(\sum_{t=1}^T \beta_t h_t(x) \right)$

35 / 47

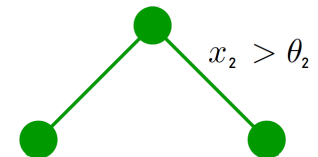
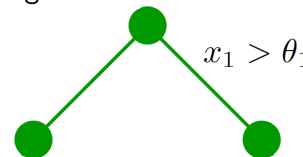
Example

10 data points in \mathbb{R}^2

The size of + or - indicates the **weight**, which starts from uniform D_1

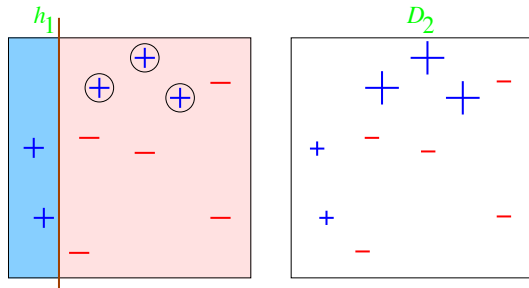


Base algorithm is decision stump:



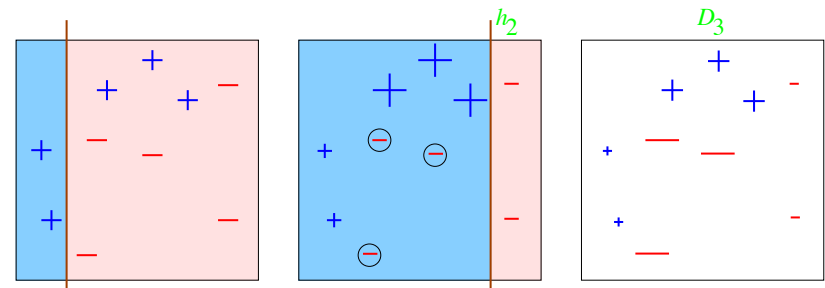
Observe that **no stump can predict very accurately for this dataset**

36 / 47

Round 1: $t = 1$ 

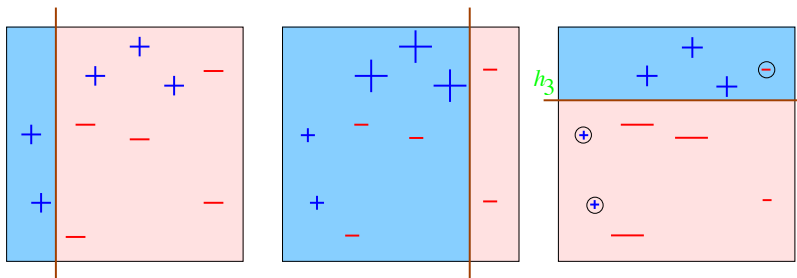
- 3 misclassified (circled): $\epsilon_1 = 0.3 \rightarrow \beta_1 = \frac{1}{2} \ln \left(\frac{1-\epsilon_1}{\epsilon_1} \right) \approx 0.42$.
- D_2 puts more weights on those examples

37 / 47

Round 2: $t = 2$ 

- 3 misclassified (circled): $\epsilon_2 = 0.21 \rightarrow \beta_2 = 0.65$.
- D_3 puts more weights on those examples

38 / 47

Round 3: $t = 3$ 

- again 3 misclassified (circled): $\epsilon_3 = 0.14 \rightarrow \beta_3 = 0.92$.

39 / 47

Final classifier: combining 3 classifiers

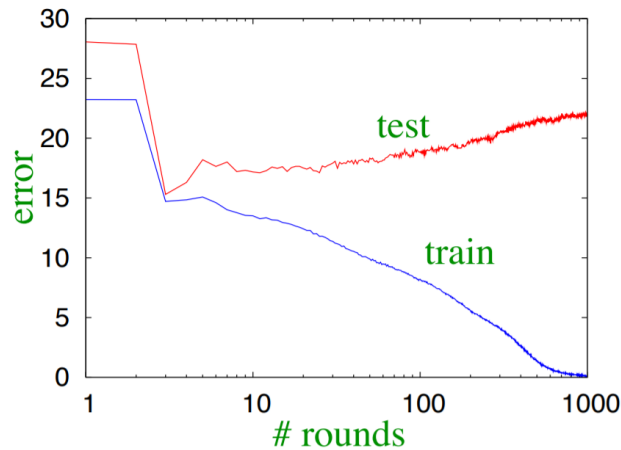
$$H_{\text{final}} = \text{sign} \left(0.42 \left(\text{blue region} \right) + 0.65 \left(\text{blue region} \right) + 0.92 \left(\text{blue region} \right) \right)$$

All data points are now classified correctly, even though each weak classifier makes 3 mistakes.

40 / 47

Overfitting

When T is large, the model is very complicated and overfitting can happen

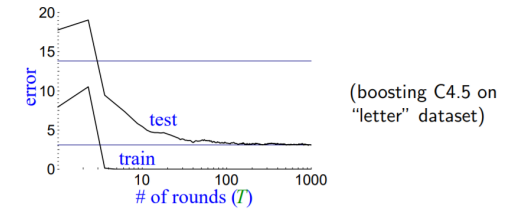


(boosting "stumps" on heart-disease dataset)

41 / 47

Resistance to overfitting

However, *very often AdaBoost is resistant to overfitting*



- test error does **not** increase, even after 1000 rounds
 - (total size > 2,000,000 nodes)
- test error continues to drop even after training error is zero!

	# rounds		
	5	100	1000
train error	0.0	0.0	0.0
test error	8.4	3.3	3.1

Used to be a mystery, but by now rigorous theory has been developed to explain this phenomenon.

42 / 47

Why AdaBoost works?

In fact, *AdaBoost also follows the general framework of minimizing some surrogate loss*.

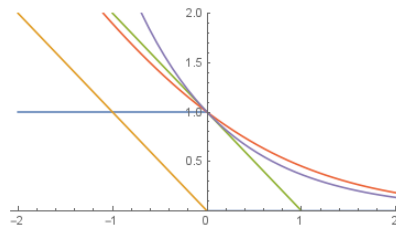
Step 1: the model that AdaBoost considers is

$$\left\{ \text{sgn}(f(\cdot)) \mid f(\cdot) = \sum_{t=1}^T \beta_t h_t(\cdot) \text{ for some } \beta_t \geq 0 \text{ and } h_t \in \mathcal{H} \right\}$$

where \mathcal{H} is the set of models considered by the base algorithm

Step 2: the loss that AdaBoost minimizes is the **exponential loss**

$$\sum_{n=1}^N \exp(-y_n f(\mathbf{x}_n))$$



43 / 47

Greedy minimization

Step 3: the way that AdaBoost minimizes exponential loss is by a **greedy approach**, that is, find β_t, h_t one by one for $t = 1, \dots, T$.

Specifically, let $f_t = \sum_{\tau=1}^t \beta_\tau h_\tau$. Suppose we have found f_{t-1} , *what should f_t be?* Greedily, we want to find β_t, h_t to minimize

$$\begin{aligned} \sum_{n=1}^N \exp(-y_n f_t(\mathbf{x}_n)) &= \sum_{n=1}^N \exp(-y_n f_{t-1}(\mathbf{x}_n)) \exp(-y_n \beta_t h_t(\mathbf{x}_n)) \\ &\propto \sum_{n=1}^N D_t(n) \exp(-y_n \beta_t h_t(\mathbf{x}_n)) \end{aligned}$$

where the last step is by the definition of weights

$$D_t(n) \propto D_{t-1}(n) \exp(-y_n \beta_{t-1} h_{t-1}(\mathbf{x}_n)) \propto \dots \propto \exp(-y_n f_{t-1}(\mathbf{x}_n))$$

44 / 47

Greedy minimization

So the goal becomes finding $\beta_t, h_t \in \mathcal{H}$ that minimize

$$\begin{aligned}
 & \sum_{n=1}^N D_t(n) \exp(-y_n \beta_t h_t(\mathbf{x}_n)) \\
 &= \sum_{n: y_n \neq h_t(\mathbf{x}_n)} D_t(n) e^{\beta_t} + \sum_{n: y_n = h_t(\mathbf{x}_n)} D_t(n) e^{-\beta_t} \\
 &= \epsilon_t e^{\beta_t} + (1 - \epsilon_t) e^{-\beta_t} \quad (\text{recall } \epsilon_t = \sum_{n: y_n \neq h_t(\mathbf{x}_n)} D_t(n)) \\
 &= \epsilon_t (e^{\beta_t} - e^{-\beta_t}) + e^{-\beta_t}
 \end{aligned}$$

It is now clear we should find h_t to minimize its the weighted classification error ϵ_t , *exactly what the base algorithm should do intuitively!*

This greedy step is abstracted out through a base algorithm.

Greedy minimization

When h_t (and thus ϵ_t) is fixed, we then find β_t to minimize

$$\epsilon_t (e^{\beta_t} - e^{-\beta_t}) + e^{-\beta_t}$$

In HW 3, you will verify that this exactly gives:

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

Keep doing this greedy minimization gives the AdaBoost algorithm.

Summary for boosting

Key idea of boosting is to **combine weak predictors into a strong one**.

There are many boosting algorithms; AdaBoost is the most classic one.

AdaBoost is **greedily minimizing the exponential loss**.

AdaBoost tends to **not overfit**.