CS 4602

Introduction to Machine Learning

Decision Trees

(Bagging and Boosting)

Instructor: Po-Chih Kuo

Roadmap

- Introduction and Basic Concepts
- Regression (Error-Based Learning)
- Bayesian Classifiers (Probability-Based Learning)
- Decision Trees (Information-Based Learning)
- KNN (Similarity-Based Learning)
- Linear Classifier
- Neural Networks
- Deep learning
- Convolutional Neural Networks
- RNN/Transformer
- Reinforcement Learning
- Model Selection and Evaluation
- Clustering
- Data Exploration & Dimensionality reduction

Make a Decision

Problem: decide whether to wait for a table at a restaurant based on the following attributes:

1. Alternate: is there an alternative restaurant nearby?
2. Bar: is there a comfortable bar area to wait in?
3. Fri/Sat: is today Friday or Saturday?
4. Hungry: are we hungry?
5. Patrons: number of people in the restaurant (None, Some, Full)
6. Price: price range ($, $$, $$$)
7. Raining: is it raining outside?
8. Reservation: have we made a reservation?
9. Type: kind of restaurant (French, Italian, Thai, Burger)
10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)

Attribute-based representations

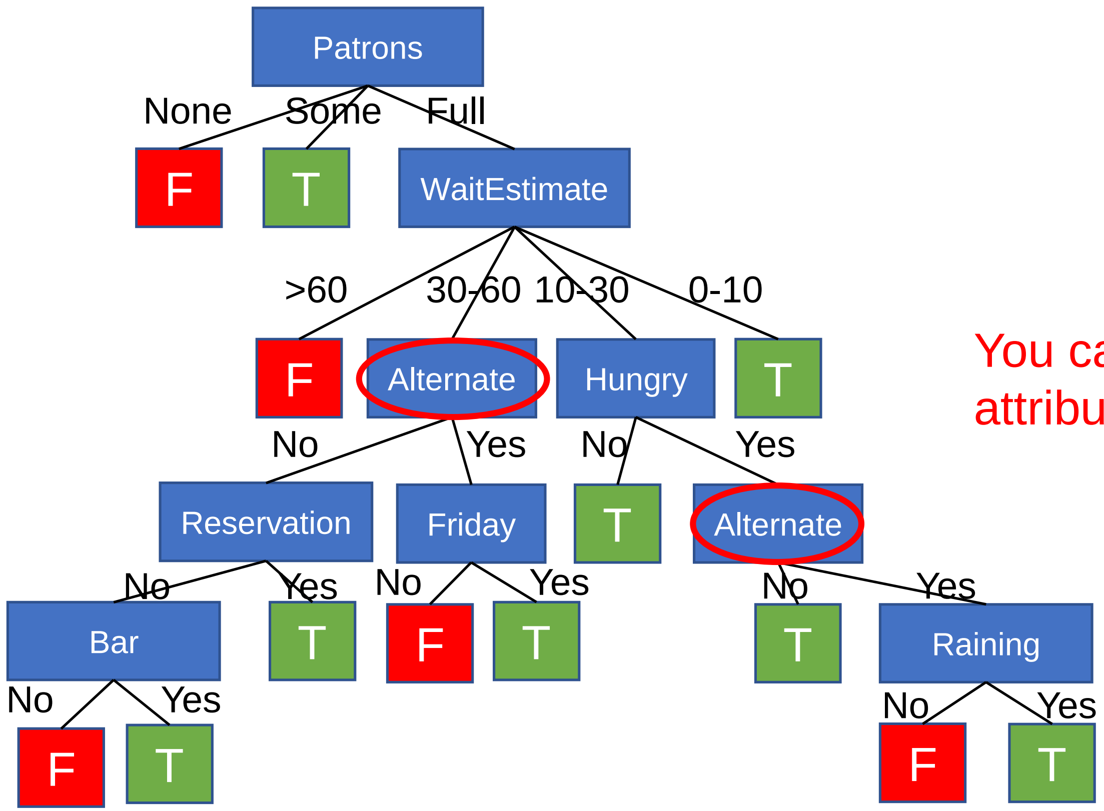
- Examples described by attribute values (Boolean, discrete, continuous)
- E.g., situations where I will/won't wait for a table:

Example	Attributes										Target
	Alt.	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est.	Wait
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

- Classification of examples is positive (T) or negative (F)
- A number N of instances, each with attributes $(x_1,x_2,x_3,...x_d)$ and target value y.

Decision trees

- One possible representation for hypotheses
- We imagine someone taking a sequence of decisions.
- E.g., here is the **TRUE** tree for deciding whether to wait:

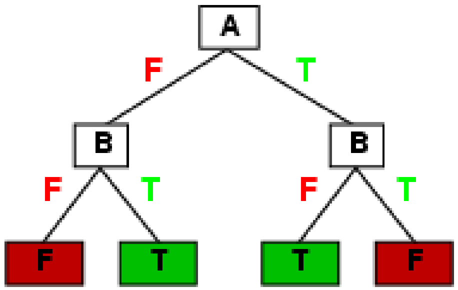


You can use the same attribute more than once.

Expressiveness

- Decision trees can express any function of the input attributes.
- E.g., for Boolean functions, truth table row → path to leaf:

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



- Trivially, there is a consistent decision tree for any training set with one path to leaf for each example (unless nondeterministic in x) but **it probably won't generalize to new examples**
- Prefer to find more compact decision trees: we don't want to memorize the data, we want to find structure in the data!

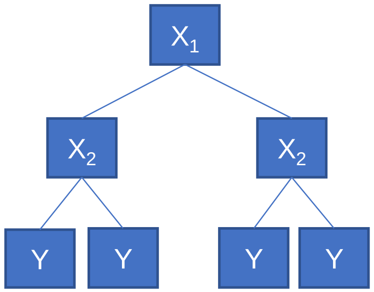
Hypothesis spaces

How many distinct decision trees with *n* Boolean attributes?

= number of Boolean functions

= number of distinct truth tables with 2^n rows = 2^{2^n}

X_1	X_2	Y
0	0	?
0	1	?
1	0	?
1	1	?



n=2: 2^2 = 4 rows. For each row we can choose T or F: 2^4 functions.

- E.g., with 6 Boolean attributes, how many trees?

There are 18,446,744,073,709,551,616 trees !

Decision tree learning

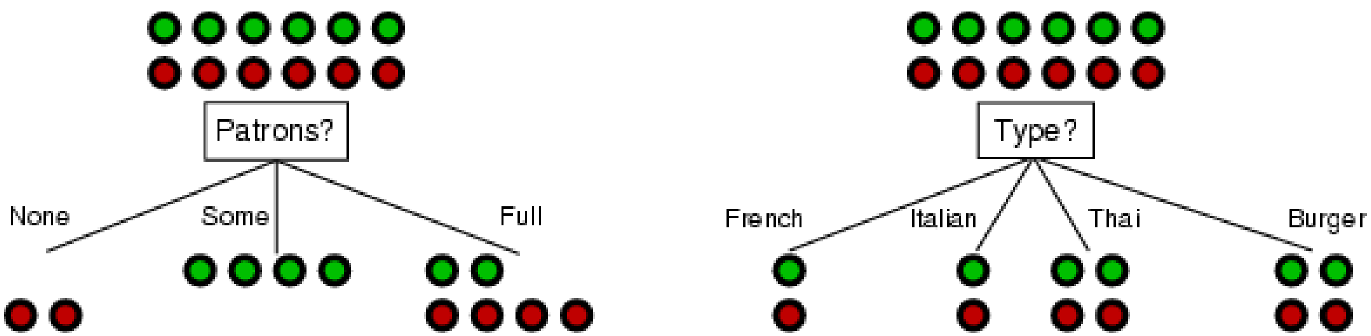
- If there are so many possible trees, can we actually search this space? (solution: greedy search).

- Aim: find a small tree consistent with the training examples

- Idea: Recursively choose "most significant" attribute as root of (sub-) tree.

Choosing a significant attribute

- Idea: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"



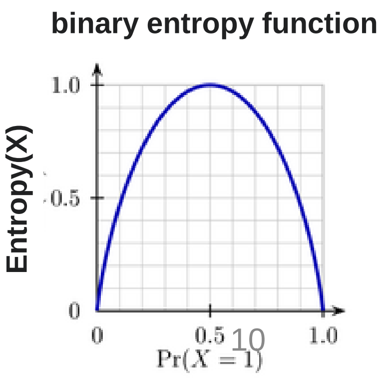
To wait or not to wait is still at 50%!

- *Patrons or type?*

Information theory

- Entropy measures the amount of uncertainty in a probability distribution:
  - Consider tossing a biased coin.
  - If you toss the coin VERY often, the frequency of heads is p and the frequency of tails is 1-p. (fair coin p=0.5).
  - The uncertainty in any actual outcome is given by the entropy.
  - The uncertainty is zero if p=0 or 1 and maximal if we have p=0.5.

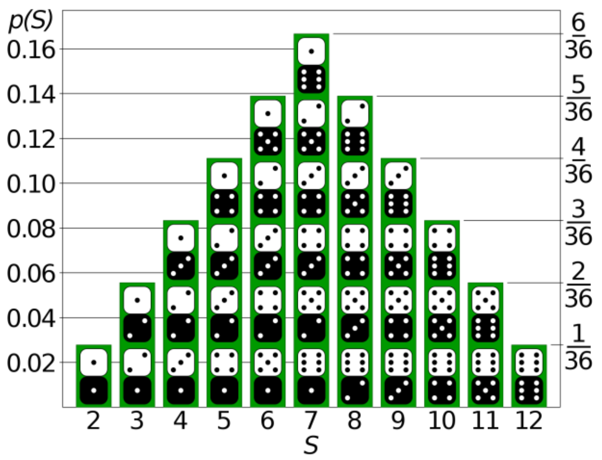
$$\mathrm{Entropy(X)} = -\mathrm{p}\log_2\mathrm{p} - (1-\mathrm{p})\log_2(1-\mathrm{p})$$



Information theory

- If there are more than two states s=1,2,..n we have (e.g. a die):

$$
\begin{aligned}
&\mathrm{Entropy}(X) = \\
&-p(s=1)\log_2[p(s=1)] \\
&-p(s=2)\log_2[p(s=2)] \\
&\ldots \\
&-p(s=n)\log_2[p(s=n)]
\end{aligned}
$$



$$\sum_{s=1}^{n} p(s) = 1$$

Information theory

- Imagine we have training data: *p* positive examples and *n* negative examples.

- Our best estimate of true or false is given by:

$$\mathrm{P(positive)} \approx \frac{\mathrm{p}}{\mathrm{p+n}}$$
$$\mathrm{P(negative)} \approx \frac{\mathrm{n}}{\mathrm{p+n}}$$

- Hence the entropy is given by:

$$Entropy\left(\frac{p}{p+n}, \frac{n}{p+n}\right) \approx -\frac{p}{p+n}\log\frac{p}{p+n} - \frac{n}{p+n}\log\frac{n}{p+n}$$

What about Cross-entropy?

- **Entropy** is a measure of the uncertainty in **one** probability distribution.
- **Cross-entropy** is a measure of the difference between **two** probability distributions.

We’ll talk about this more in the neural network section

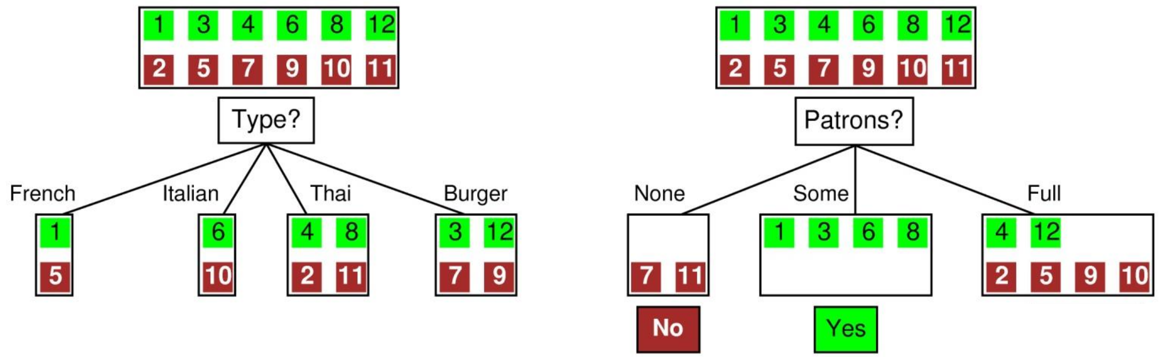
Information gain

- How much information do we gain if we disclose the value of some attribute?

- Answer:

  Decrease of uncertainty
  = uncertainty before - uncertainty after

Example



Before: Entropy = - ½ log(1/2) – ½ log(1/2)=log(2) = 1 bit:
There is "1 bit of information to be discovered".

After: for Type: If we go into branch "French" we have 1 bit, similarly for the others.

French: 1bit
Italian: 1 bit
Thai: 1 bit
Burger: 1bit

On average: 1 bit ! We gained nothing!

After: for Patrons: In branch "None" and "Some" entropy = 0
In branch "Full" entropy = -1/3log(1/3)-2/3log(2/3) = 0.918....

So Patrons gains more information!

Information gain

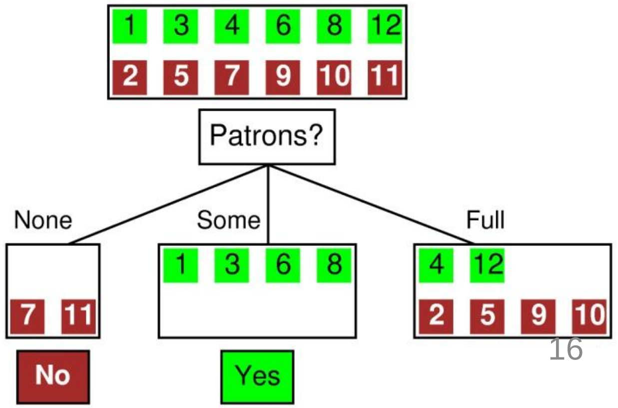
- How do we combine branches:

1/6 of the time we enter “None”, so we weight “None” with 1/6.
Similarly: “Some” has weight: 1/3 and “Full” has weight ½.

$$Entropy(A) = \sum_{i=1}^{n} \frac{p_i + n_i}{p + n} Entropy(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i})$$

entropy for each branch.

weight for each branch



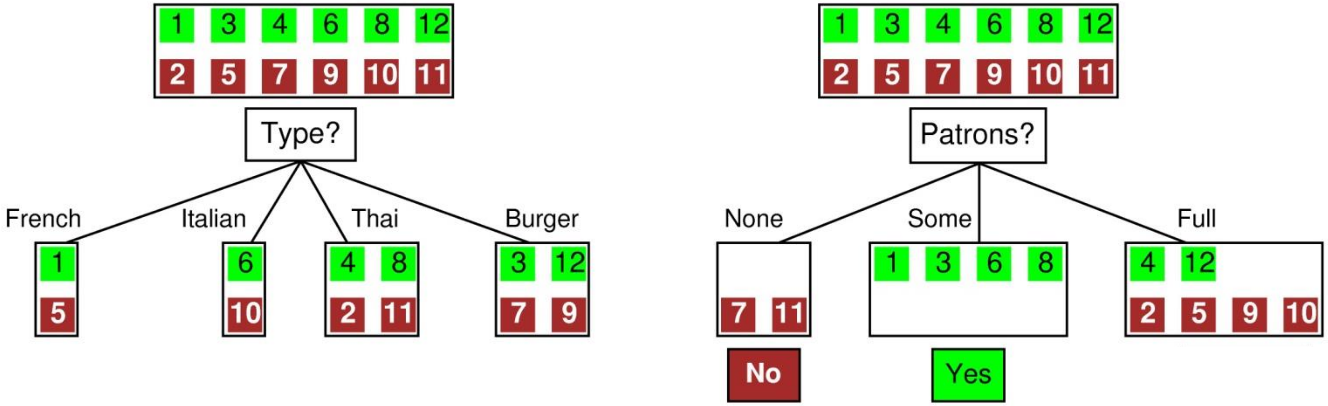
Information gain

- **Information Gain (IG)** or reduction in entropy from the attribute test:

$$IG(A) = Entropy\ before - Entropy\ after$$

- Choose the attribute with the **largest** IG

Information gain



For the training set, p= n = 6, E(6/12, 6/12) = 1 bit

$$IG(Patrons) = 1 - [\frac{2}{12}E(0,1) + \frac{4}{12}E(1,0) + \frac{6}{12}E(\frac{2}{6},\frac{4}{6})] = 0.541 \text{ bits}$$

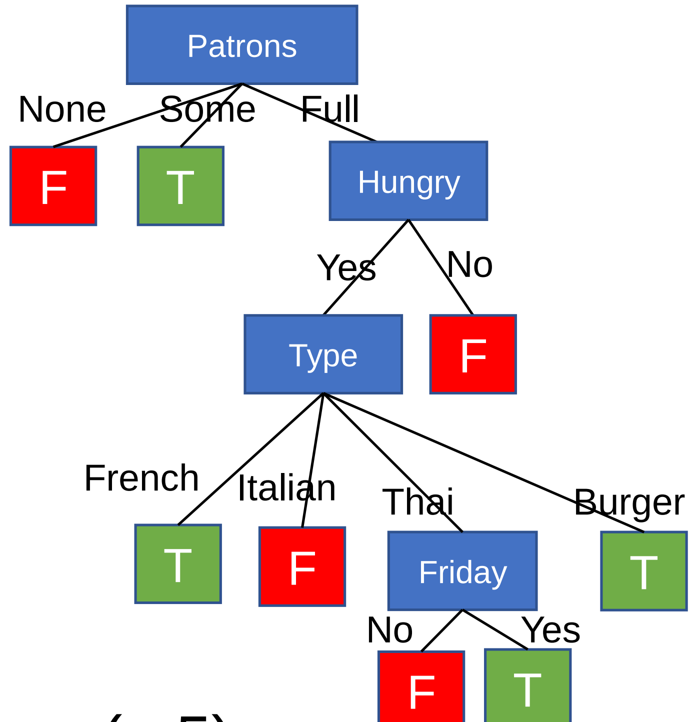
$$IG(Type) = 1 - [\frac{2}{12}E(\frac{1}{2},\frac{1}{2}) + \frac{2}{12}E(\frac{1}{2},\frac{1}{2}) + \frac{4}{12}E(\frac{2}{4},\frac{2}{4}) + \frac{4}{12}E(\frac{2}{4},\frac{2}{4})] = 0 \text{ bits}$$

Patrons has the highest IG of all attributes and so is chosen by the Decision Tree Learning algorithm as the root

Is it sensible to have the same attribute on a single branch of the tree (why)?

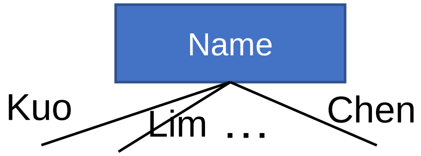
Example contd.

- Decision tree learned from the 12 examples:



- Simpler than **TRUE** tree (p.5)
  - a more complex hypothesis isn't justified by small amount of data

Gain-Ratio



- If 1 attribute splits in many more classes than another, it has an (unfair) advantage if we use information gain.

- The gain-ratio is designed to compensate for this problem,

$$GainRatio = \frac{IG}{-\sum_{i=1}^{n} \frac{p_i + n_i}{p + n} \log \frac{p_i + n_i}{p + n}}$$

- if we have n uniformly populated classes the denominator is log2(n) which penalized relative to 1 for 2 uniformly populated classes.

Gain-Ratio (Example)

$$\textbf{GainRatio}(T,X) = \frac{\textbf{Gain}(T,X)}{\textbf{SplitInformation}(T,X)}$$

$$Split(T,X) = -\sum_{c \in A} P(c) \log_2 P(c)$$

		Play Golf		
		Yes	No	*total*
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
Gain = 0.247				

Split (Play,Outlook) = - $(5/14*\log_2(5/14) + 4/14*\log_2(4/14) + 5/14*\log_2(5/14))$
= 1.577

Gain Ratio (Play,Outlook) = 0.247/1.577 = **0.156**

What to Do if...

- In some leaf there are no examples:

  Choose True or False according to the number of positive/negative examples at your parent.

- There are no attributes left

  Two or more examples have the same attributes but different label: we have an error/noise. Stop and use majority vote.

Continuous variables

- If variables are continuous we can
  - bin them
  - learn a simple classifier on a single dimension (e.g. logistic regression classifier).

A practical way for continuous variables

Ex.	Est	Wait
X_1	8	T
X_2	40	F
X_3	8	T
X_4	12	T
X_5	70	F
X_6	3	T
X_7	7	F
X_8	6	T
X_9	80	F
X_{10}	20	F
X_{11}	8	F
X_{12}	40	T

sort

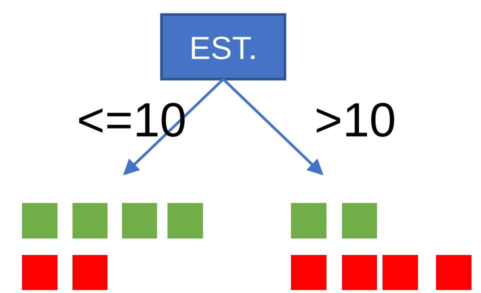
Ex.	Est	Wait
X_6	3	T
X_8	6	T
X_7	7	F
X_1	8	T
X_3	8	T
X_{11}	8	F
X_4	12	T
X_{10}	20	F
X_2	40	F
X_{12}	40	T
X_5	70	F
X_9	80	F

Middle point (what else?)

- 4.5 → IG
- 6.5 → IG
- 7.5 → IG
- 10 → IG
- 16 → IG
- 30 → IG
- 55 → IG
- 75 → IG

if there are **N** possible values, we would have at most **N-1** possible splits.

→ Max(IGs) is the IG for this attribute (EST.)



When to Stop ?

- If we keep going until perfect classification we might over-fit.
- Heuristics:
  - Stop when Info-Gain (Gain-Ratio) is smaller than threshold
  - Stop when there are M examples in each leaf node
- Penalize complex trees by minimizing with "complexity" = # nodes.
  Note: if tree grows, complexity grows but entropy shrinks.

$$\alpha \times complexity + \sum_{all\ leafs} entropy(leaf)$$

- Compute many full grown trees on subsets of data and test them on hold-out data. Pick the best or average their prediction.
- Do a statistical test: is the increase in information significant.

How to improve Decision Trees?

“Unity is strength!”

Ensemble methods!

Ensemble methods: bagging, boosting

Aims

- **Bagging (b**ootstrap **agg**regat**ing)** algorithms aim to reduce the complexity of models that **overfit** the training data.
- **Boosting** is an approach to increase the complexity of models that **underfit** the training data.

How do they work?

- **Bagging:** learn homogeneous weak learners **independently** from each other in **parallel** and combine them following some kind of deterministic averaging process.
- **Boosting:** lean homogeneous weak learners **sequentially** in an adaptative way (a base model depends on the previous ones) and combine them following a deterministic strategy.
- **Stacking:** learn heterogeneous weak learners in **parallel** and combines them by **training a meta-model** to output a prediction based on the different weak models predictions.

Ref: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

Bagging

- “bagging” (standing for “bootstrap aggregating”) that aims at producing an ensemble model that is **more robust** than the individual models composing it.
- The low correlation between models is the key.
- Bootstrapping

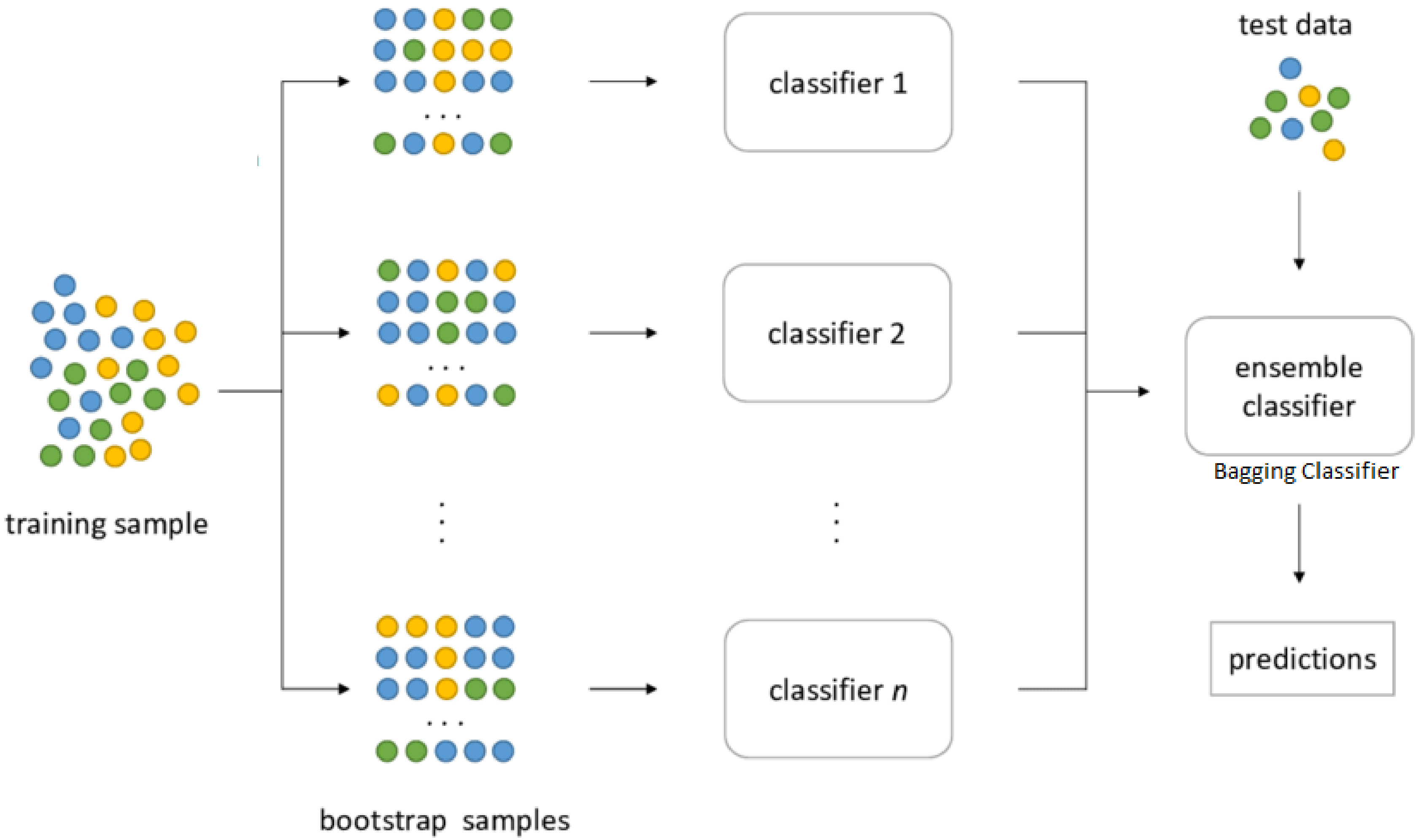
Bagging

- Assuming that we have L bootstrap samples (approximations of L independent datasets)
- We can fit L almost independent weak learners (one on each dataset)

$$w_1(.), w_2(.), ..., w_L(.)$$

- Then aggregate them into some kind of averaging or voting process in order to get an ensemble model with a lower variance.

$$s_L(.) = \frac{1}{L} \sum_{l=1}^{L} w_l(.)$$

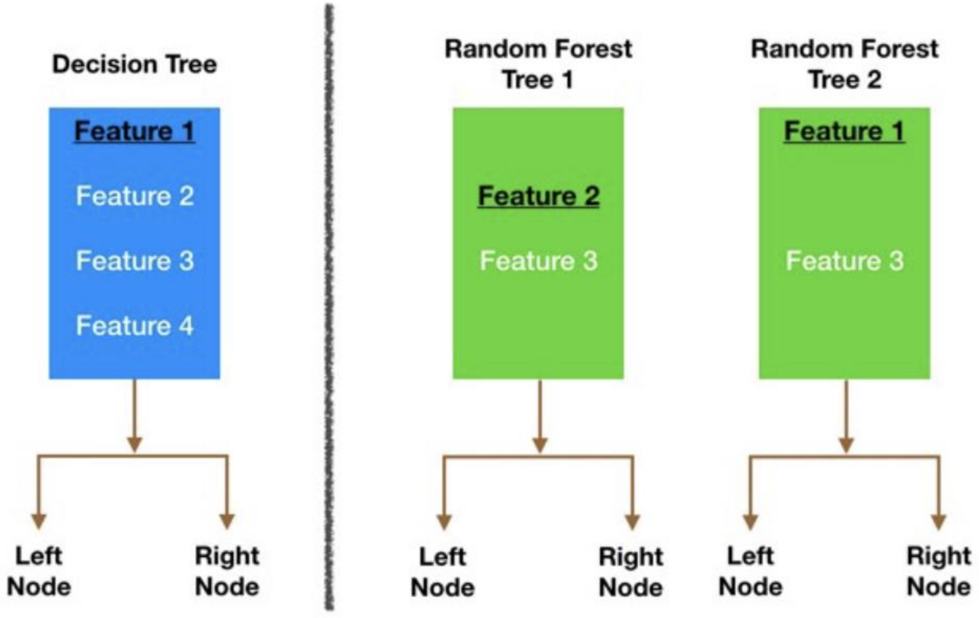


Bagging Classifier Process Flow

Random forests

- When growing each tree, instead of only sampling over the observations in the dataset to generate a bootstrap sample, it also **sample over features** and keep only a random subset of them to build the tree.
  - It forces even more variation amongst the trees in the model.
  - It reduces the correlation between the different returned outputs.
  - It makes the decision making process more robust to missing data.

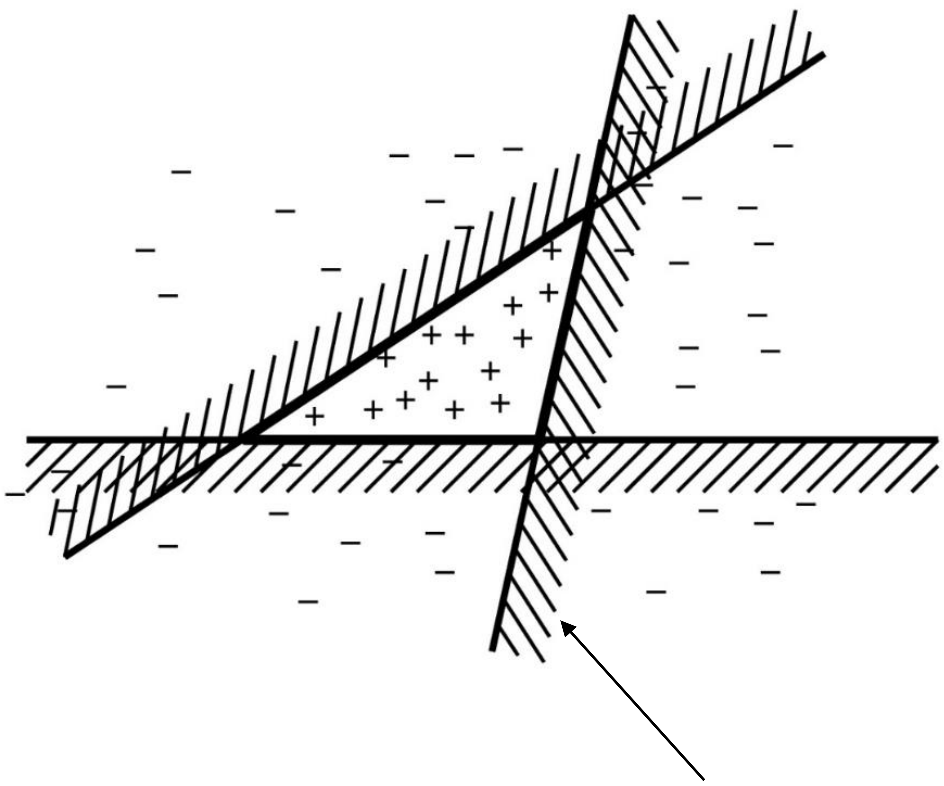
Random forests



Boosting

- Main idea:
  - Train classifiers (e.g. decision trees) in a sequence.
  - A new classifier should focus on those cases which were incorrectly classified in the last round.
  - Combine the classifiers by letting them vote on the final prediction (like bagging).
  - Each classifier could be (should be) very “weak”, e.g. a decision stump.

Example



this line is one simple classifier saying that everything to the left "+" and everything to the right is "–"

Boosting intuition

- We adaptively weigh each data case.
- Data cases which are wrongly classified get high weight (the algorithm will focus on them)
- Each boosting round learns a new (simple) classifier on the weighed dataset.
- These classifiers are weighed to combine them into a single powerful classifier.
- Classifiers that obtain low training error rate have high weight.

Bagging

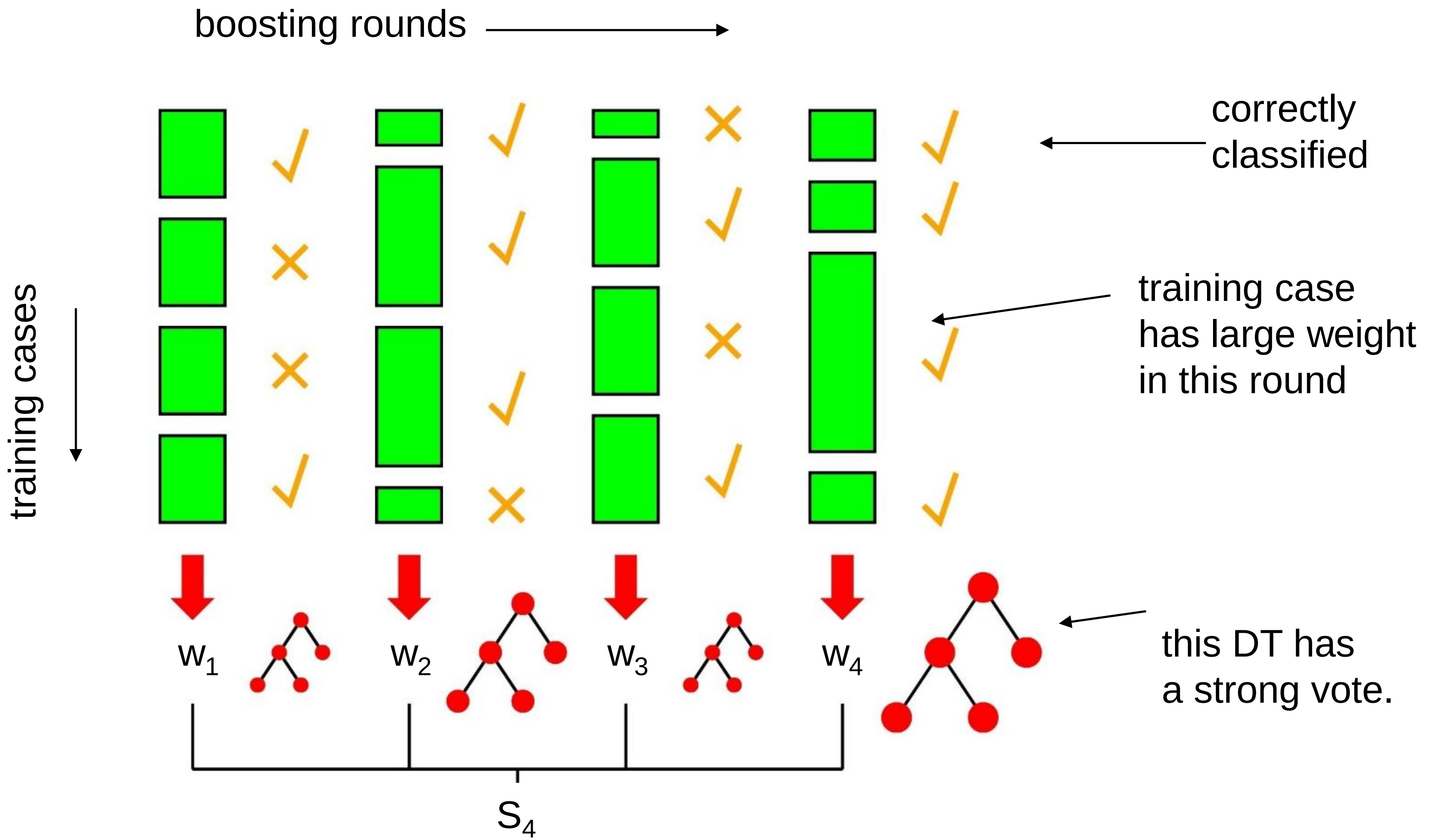
$$s_L(.) = \sum_{l=1}^{L} c_l \times w_l(.)$$

$\longrightarrow$

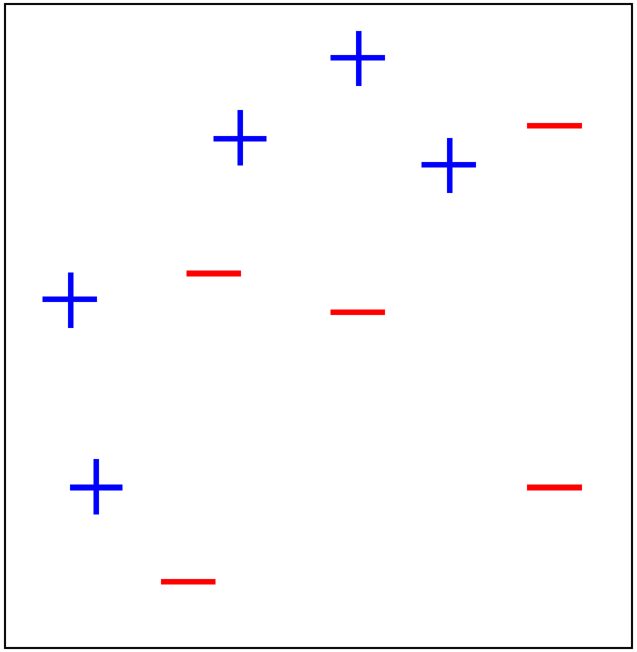
Boosting

$$s_l(.) = s_{l-1}(.) + c_l \times w_l(.)$$

Boosting in a picture



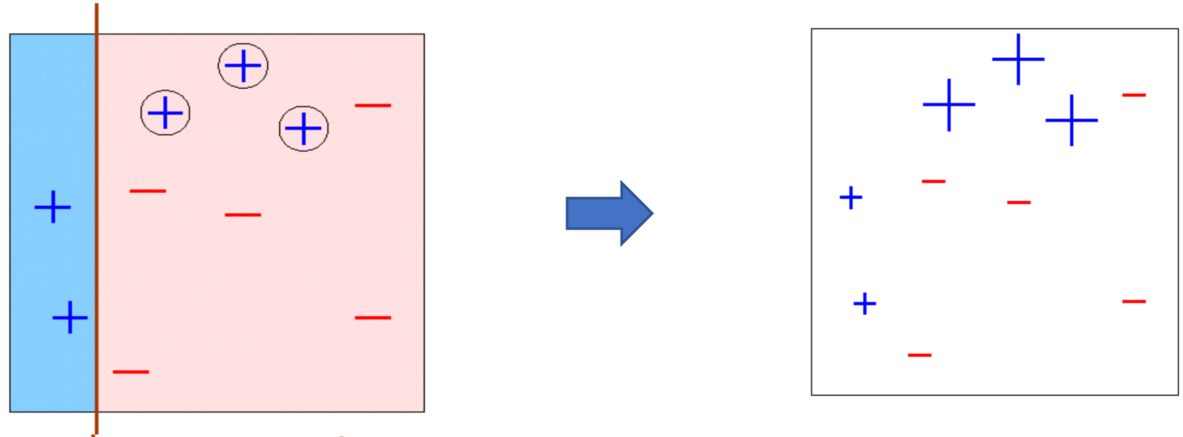
AdaBoost (Example)



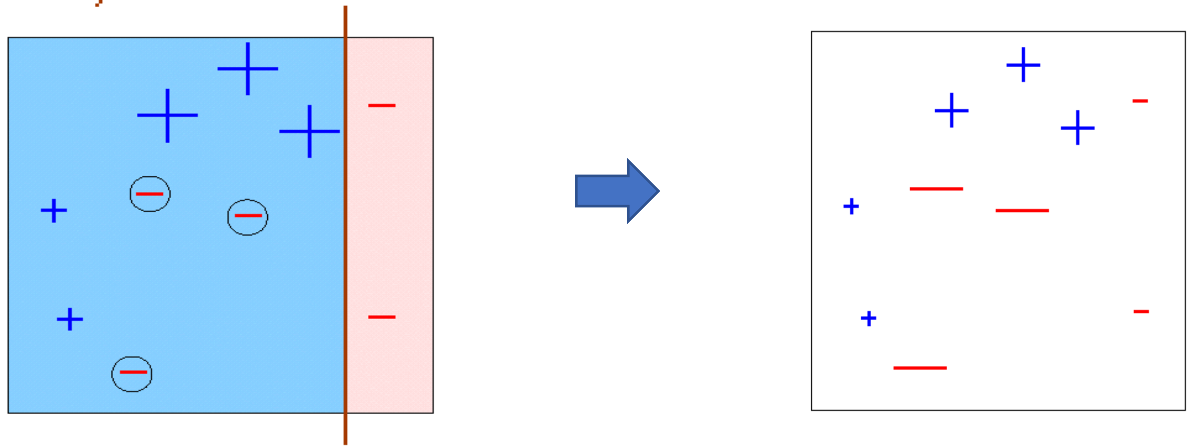
Original Training set : Equal Weights to all training samples

Taken from "**A Tutorial on Boosting" by** Yoav Freund and Rob Schapire

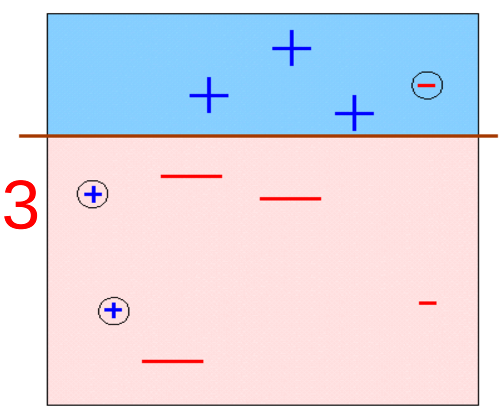
ROUND 1



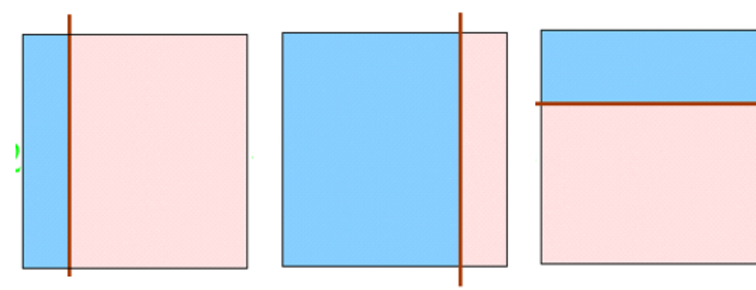
ROUND 2



ROUND 3



FINAL



AdaBoost (Algorithm)

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$

Initialize $D_1(i) = \frac{1}{m}$

For $t = 1, \ldots, T$:

1. Find the classifier $h_t : X \to \{-1, +1\}$ that minimizes the error with respect to the distribution D_t:

$$h_t = \arg\min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y(i) \neq h_j(x_i)]$$

2. Prerequisite: $\varepsilon_t < 0.5$, otherwise stop.
3. Choose $\alpha_t \in \mathbf{R}$, typically $\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$ where ε_t is the weighted error rate of classifier h_t.

4. Update: $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$ where Z_t is a normalization factor

$$\boxed{\exp(-\alpha_t y_i h_t(x_i)) \begin{cases} < 1, & y(i) = h_t(x_i) \\ > 1, & y(i) \neq h_t(x_i) \end{cases}}$$

Output the final classifier: $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

AdaBoost (Algorithm)

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X,\ y_i \in Y = \{-1, +1\}$

Initialize $D_1(i) = \frac{1}{m}$

For $t = 1, \ldots, T$:

1. Find the classifier $h_t : X \to \{-1, +1\}$ that minimizes the error with respect to the distribution D_t:

$$h_t = \arg\min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y(i) \neq h_j(x_i)]$$

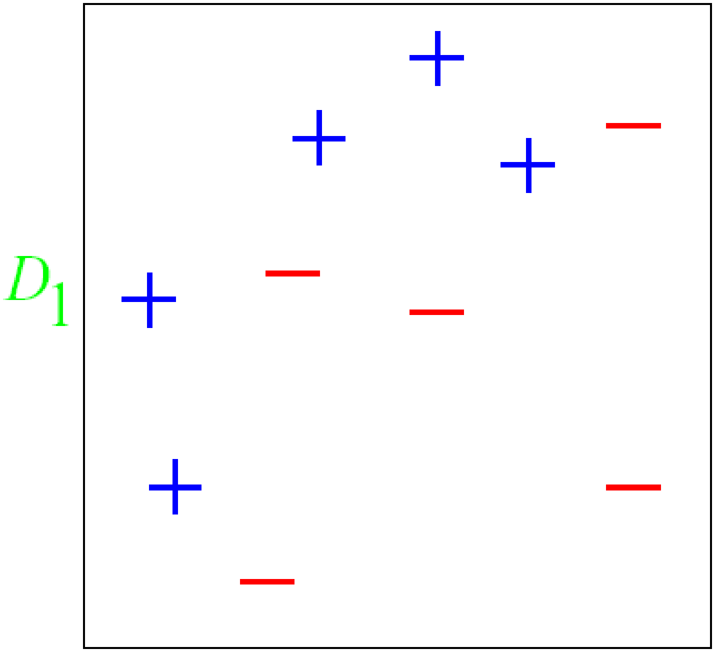
2. Prerequisite: $\epsilon_t < 0.5$, otherwise stop.
3. Choose $\alpha_t \in \mathbf{R}$, typically $\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$ where ϵ_t is the weighted error rate of classifier h_t.

4. Update: $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$ where Z_t is a normalization factor

$$\exp(-\alpha_t y_i h_t(x_i)) \begin{cases} < 1, & y(i) = h_t(x_i) \\ > 1, & y(i) \neq h_t(x_i) \end{cases}$$

Output the final classifier: $H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

AdaBoost (Example)



Original Training set : Equal Weights to all training samples

Taken from “**A Tutorial on Boosting” by** Yoav Freund and Rob Schapire

AdaBoost (Algorithm)

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X,\ y_i \in Y = \{-1, +1\}$

Initialize $D_1(i) = \frac{1}{m}$

For $t = 1, \ldots, T$:

1. Find the classifier $h_t : X \to \{-1, +1\}$ that minimizes the error with respect to the distribution D_t:

$$h_t = \arg\min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y(i) \neq h_j(x_i)]$$

2. Prerequisite: $\varepsilon_t < 0.5$, otherwise stop.
3. Choose $\alpha_t \in \mathbf{R}$, typically $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$ where ε_t is the weighted error rate of classifier h_t.

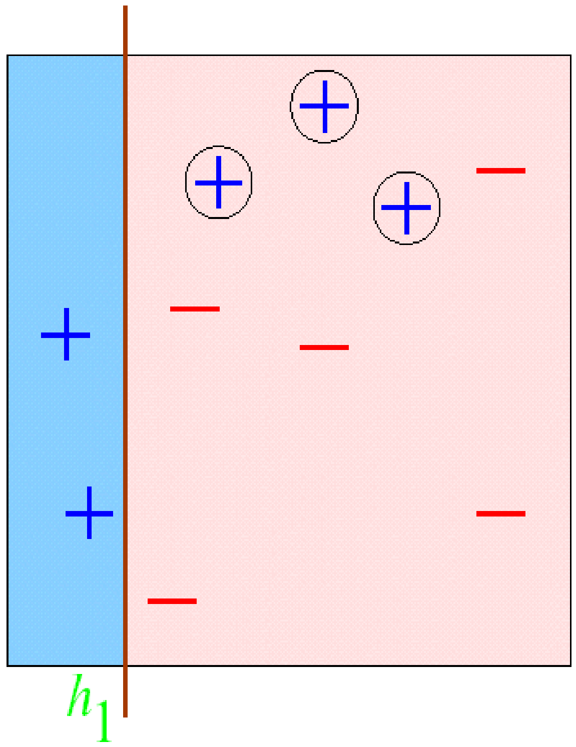
4. Update: $D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$ where Z_t is a normalization factor

$$\exp(-\alpha_t y_i h_t(x_i)) \begin{cases} < 1, & y(i) = h_t(x_i) \\ > 1, & y(i) \neq h_t(x_i) \end{cases}$$

Output the final classifier: $H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

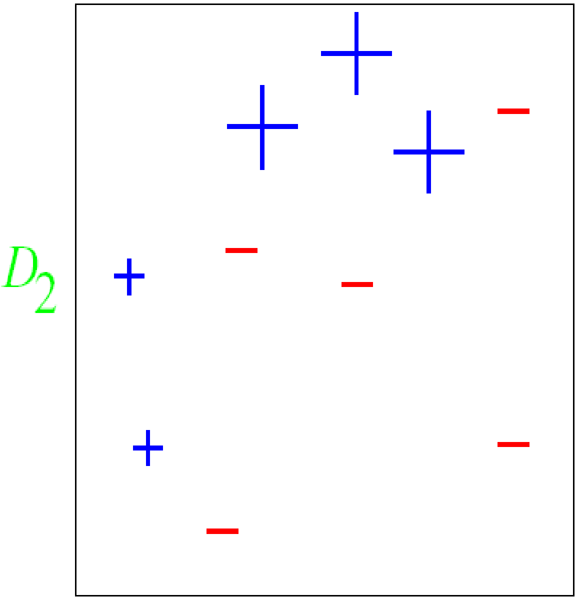
AdaBoost (Example)

ROUND 1



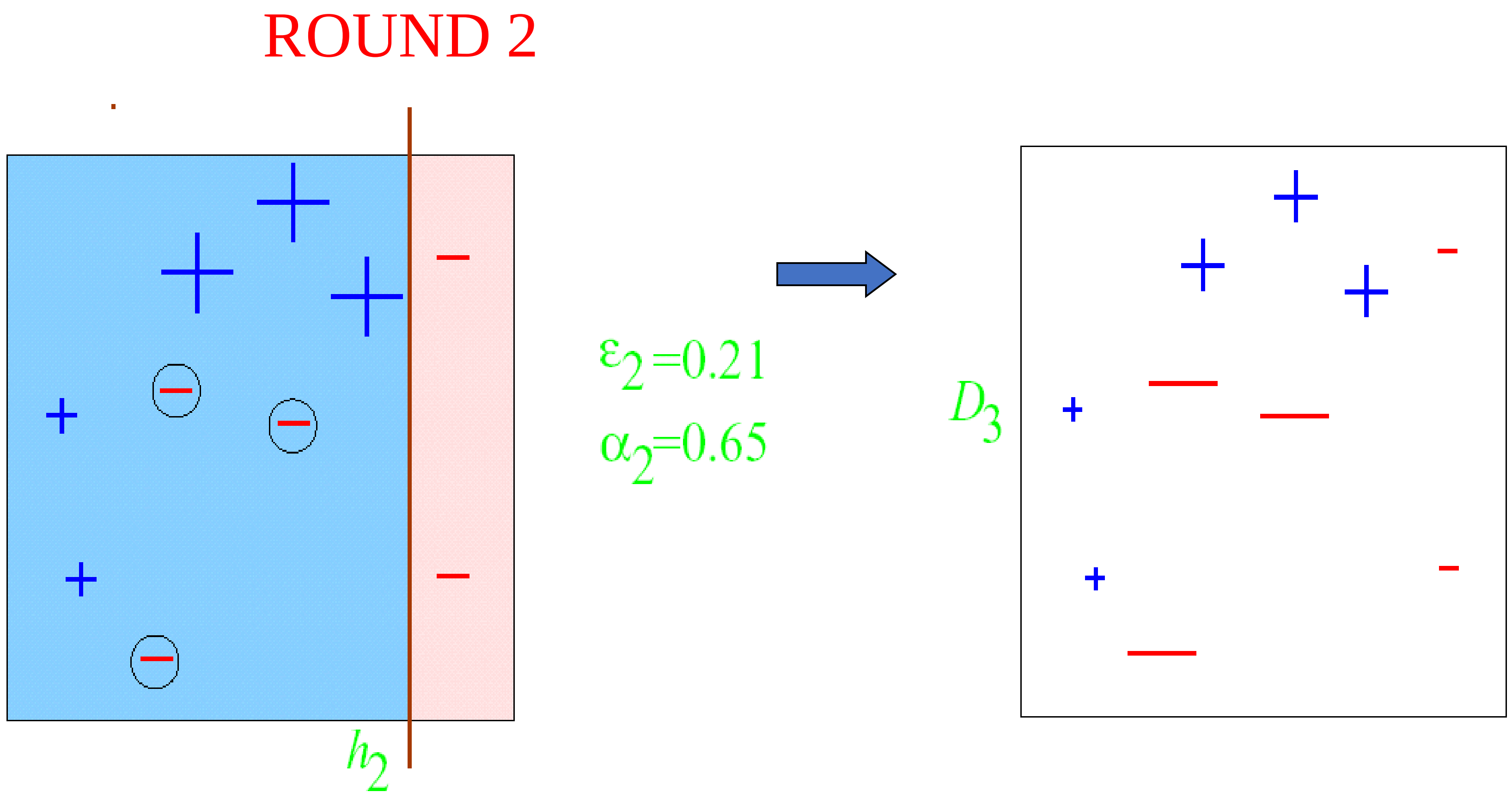
$\varepsilon_1 = 0.30$

$\alpha_1 = 0.42$



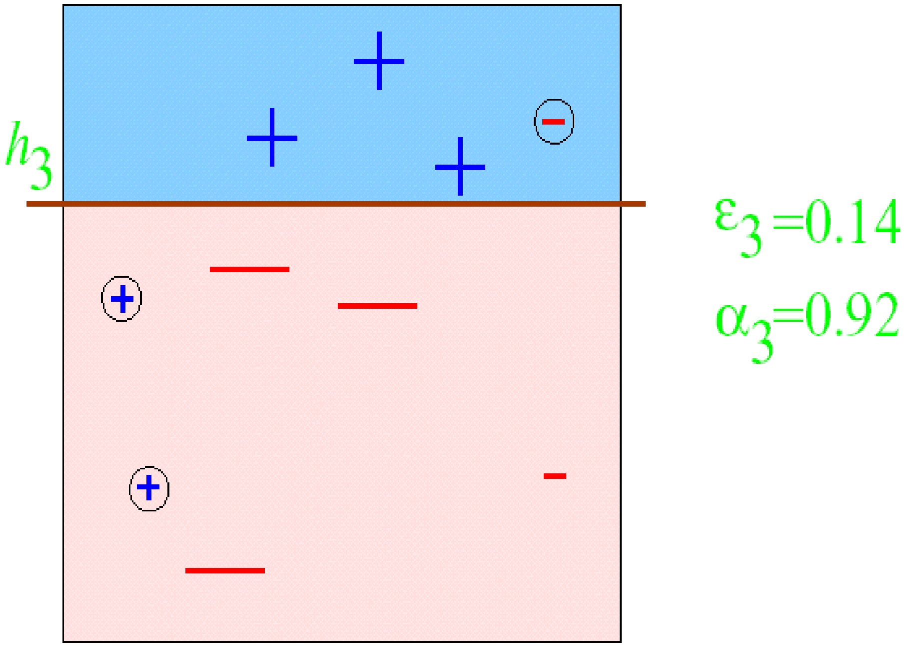
$$\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$$

AdaBoost (Example)



AdaBoost (Example)

ROUND 3



AdaBoost (Algorithm)

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$

Initialize $D_1(i) = \frac{1}{m}$

For $t = 1, \ldots, T$:

1. Find the classifier $h_t : X \to \{-1, +1\}$ that minimizes the error with respect to the distribution D_t:

$$h_t = \arg\min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y(i) \neq h_j(x_i)]$$

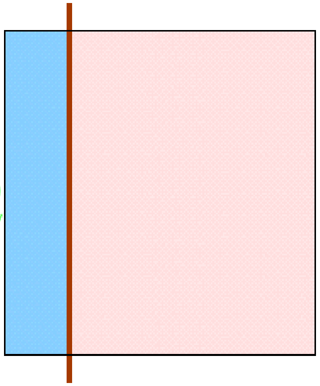
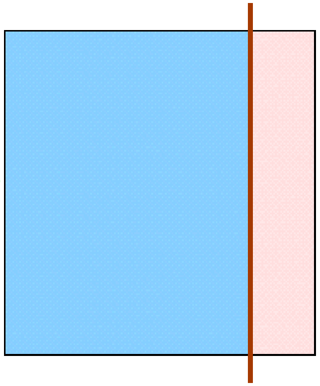
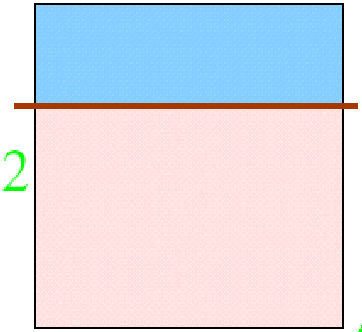
2. Prerequisite: $\varepsilon_t < 0.5$, otherwise stop.
3. Choose $\alpha_t \in \mathbf{R}$, typically $\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$ where ε_t is the weighted error rate of classifier h_t.

4. Update: $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$ where Z_t is a normalization factor

$$\exp(-\alpha_t y_i h_t(x_i)) \begin{cases} < 1, & y(i) = h_t(x_i) \\ > 1, & y(i) \neq h_t(x_i) \end{cases}$$

$\boxed{\text{Output the final classifier: } H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)}$

AdaBoost (Example)

$H_{\text{final}} = \text{sign}\Big(0.42$  $+\ 0.65$  $+\ 0.92$  $\Big)$

Gradient boosting

- Gradient boosting **casts the problem into a gradient descent one**: at each iteration we fit a weak learner to the opposite of the gradient of the current fitting error with respect to the current ensemble model.

Bagging

$$s_L(.) = \sum_{l=1}^{L} c_l \times w_l(.)$$

$\longrightarrow$

AdaBoost

$$s_l(.) = s_{l-1}(.) + c_l \times w_l(.)$$

$\downarrow$

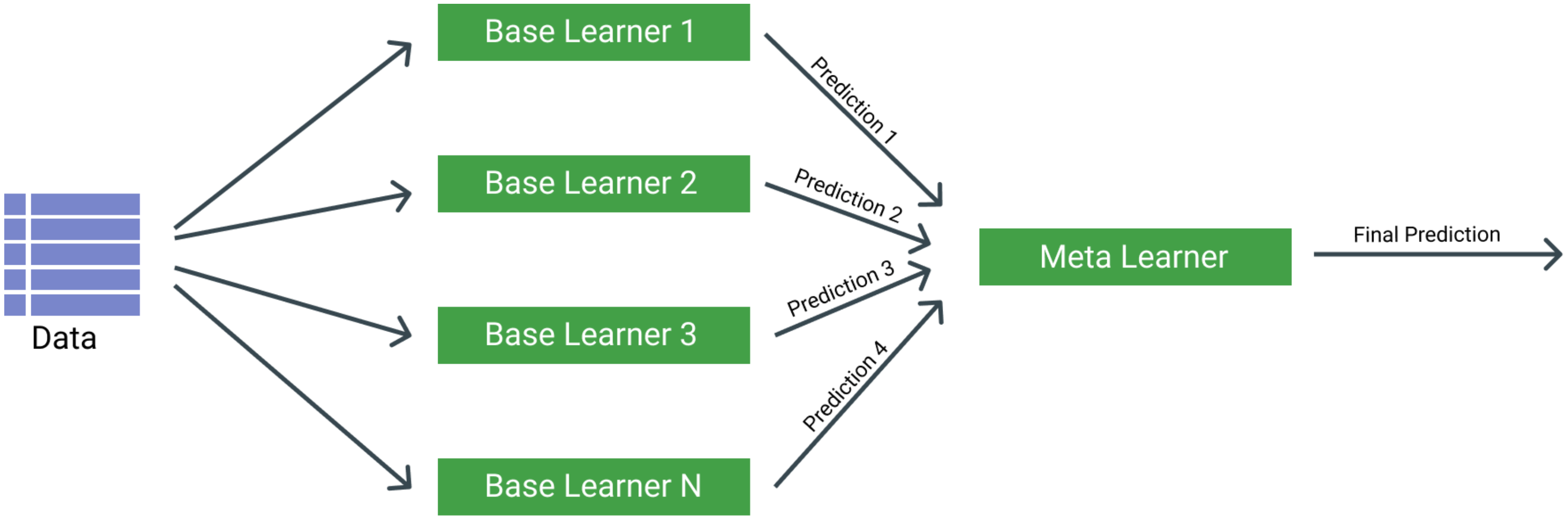
$$s_l(.) = s_{l-1}(.) - c_l \times \nabla_{s_{l-1}} E(s_{l-1})(.)$$

XGBoost(Extreme Gradient Boosting)

- Additional tricks that make learning much more efficient:
  - Implements **regularization** helping reduce overfit
  - Implements **parallel** processing being much faster (10x) than GB
  - Allows users to define **custom optimization objectives** and evaluation criteria
  - XGBoost has an in-built routine to handle **missing values**
  - XGBoost **prunes** the tree backwards and removes splits beyond which there is no positive gain
  - XGBoost allows a user to run a **cross-validation at each iteration** of the boosting process

Ref. https://medium.com/diogo-menezes-borges/boosting-with-adaboost-and-gradient-boosting-9cbab2a1af81

Stacking



Data
Base Learner 1
Base Learner 2
Base Learner 3
Base Learner N
Prediction 1
Prediction 2
Prediction 3
Prediction 4
Meta Learner
Final Prediction

What have we learnt

- **Decision tree**
  - Information Gain
- Ensemble methods
  - Bagging
    - **Random forest**
  - Boosting
    - XGBoost
  - Stacking

Questions?

https://colab.research.google.com/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/05.08-Random-Forests.ipynb