

CS 4602

Introduction to Machine Learning

Decision Trees
(Bagging and Boosting)

Instructor: Po-Chih Kuo

Roadmap

- Introduction and Basic Concepts
- Regression (Error-Based Learning)
- Decision Trees (Information-Based Learning)
- Bayesian Classifiers (Probability-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)



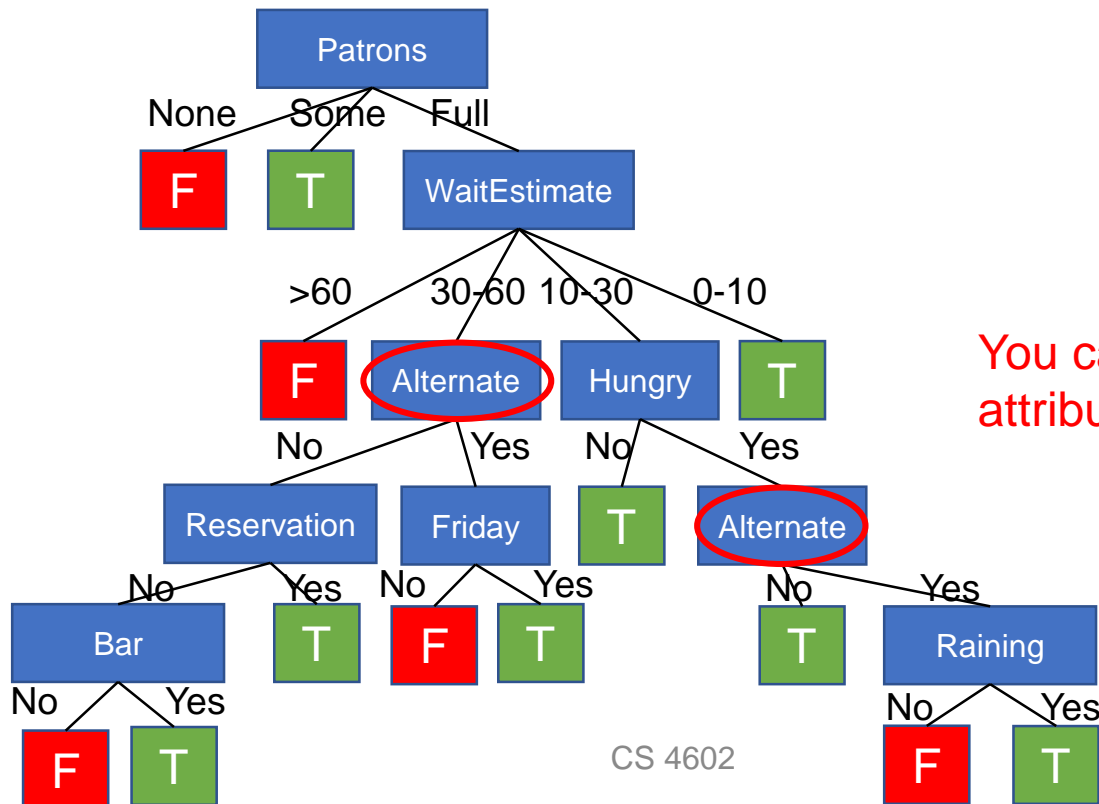
Example	Attributes (X_1, X_2, \dots)										Ans (Y)
	Alt.	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est.	Wait
X^1	T	F	F	T	Some	\$\$\$	F	T	French	0-10	T
X^2	T	F	F	T	Full	\$	F	F	Thai	30-60	F
X^3	F	T	F	F	Some	\$	F	F	Burger	0-10	T
X^4	T	F	T	T	Full	\$	F	F	Thai	10-30	T
X^5	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
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: wait) or **negative** (F: Not to wait)
- A number N of instances, each with attributes $(x^1, x^2, x^3, \dots, x^d)$ and target value y .

Decision trees



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

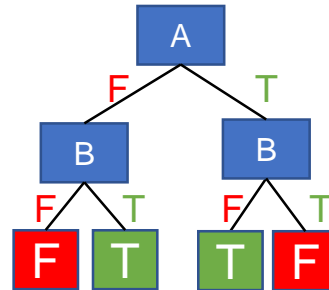


You can use the same attribute more than once.

Expressiveness

- Decision trees can express any function of the input attributes.
- E.g., for Boolean functions, truth table row \rightarrow 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, but **it probably won't generalize to new examples.**
OVERFITTING!
- Prefer to find more **compact** decision trees: we don't want to memorize the data. We want to find **structure** in the data!

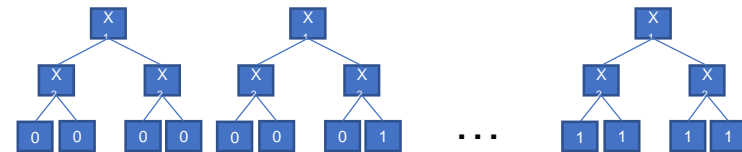
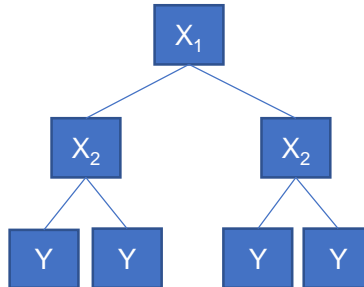
Hypothesis spaces

How many distinct decision trees with d Boolean attributes?

= number of Boolean functions

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

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



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

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

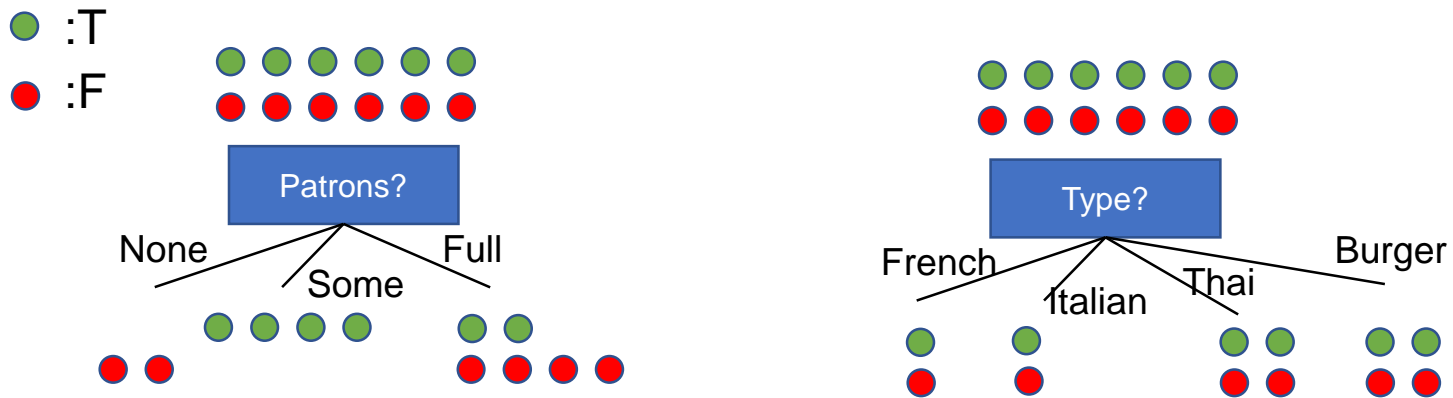
There are 18,446,744,073,709,551,616 trees !

Learn to build decision tree

- 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

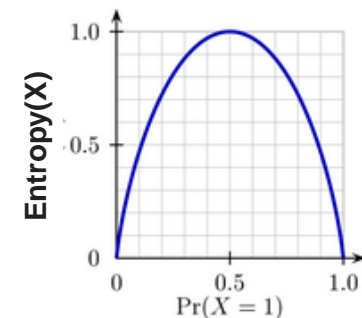


- **Entropy** measures the amount of uncertainty in a **probability** distribution:
 - Consider tossing a **biased** coin.
 - If you toss the coin VERY often, the frequency of heads is p and the frequency of tails is $1-p$. (fair coin $p=0.5$).
 - The uncertainty in any actual outcome is given by the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



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

binary entropy function



Information theory

- Imagine we have training data: p positive examples and n negative examples.
- Our best estimate of true or false is given by:

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

- Hence the entropy is given by:

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

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.

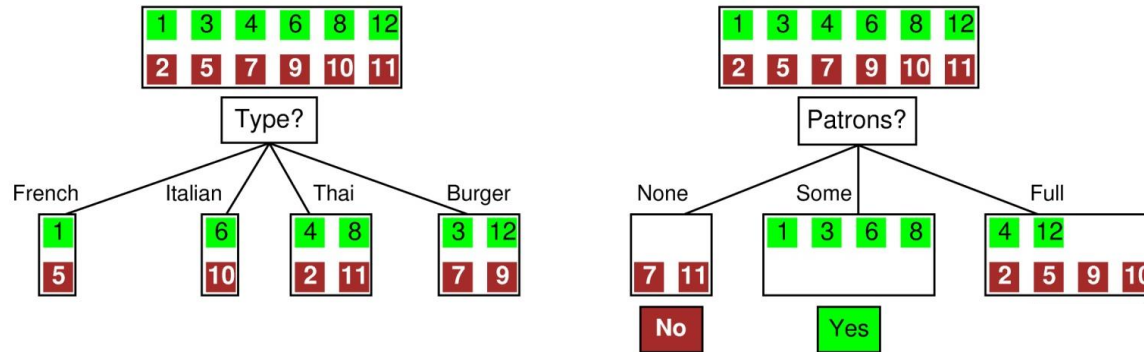


We'll talk about this more in the “neural network” lecture

Information gain

- How much information do we gain if we disclose the value of some attribute?
- Answer:
 - Decrease of uncertainty
 - =uncertainty before - uncertainty after
 - =entropy before - entropy after

Example



Before: Entropy = $-\frac{1}{2} \log(1/2) - \frac{1}{2} \log(1/2) = \log(2) = 1$ 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 = $-\frac{1}{3} \log(1/3) - \frac{2}{3} \log(2/3) = 0.918....$

So Patrons gains more information!

Information gain

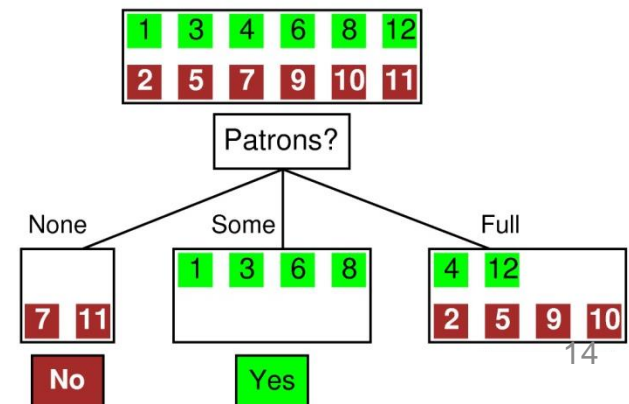
- How do we combine branches:

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 1/2.

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

weight for each branch

entropy for each branch.



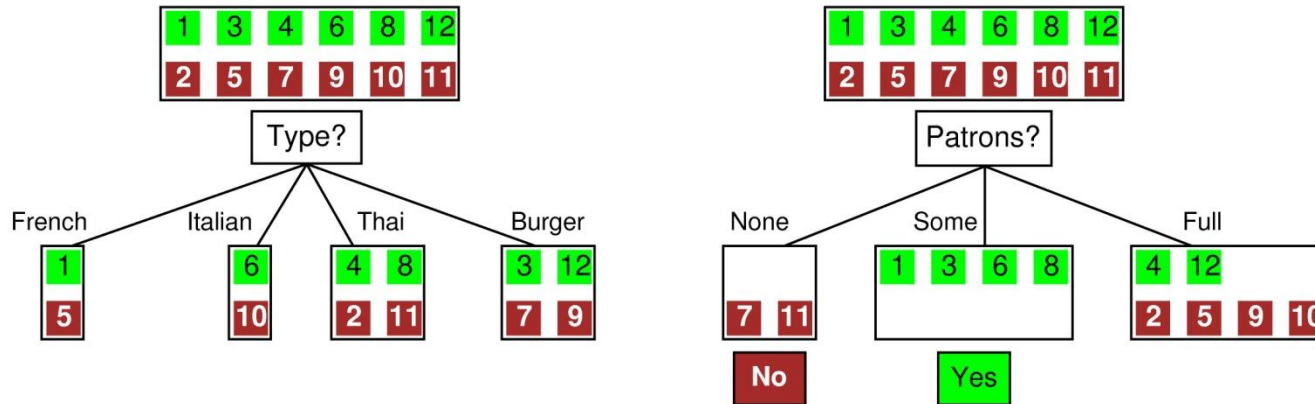
Information gain

- **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

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

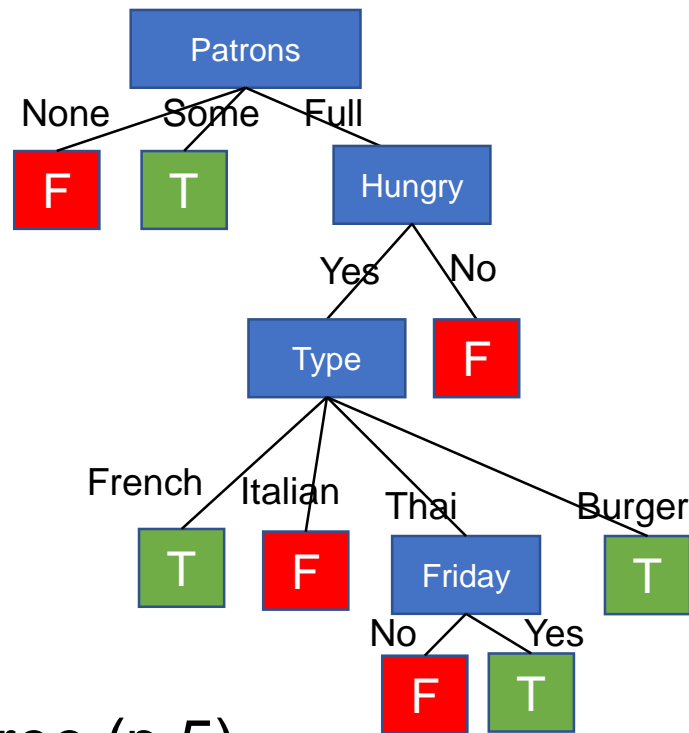
$$IG(Type) = 1 - \left[\frac{2}{12} E\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{2}{12} E\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{4}{12} E\left(\frac{2}{4}, \frac{2}{4}\right) + \frac{4}{12} E\left(\frac{2}{4}, \frac{2}{4}\right) \right] = 0 \text{ bits}$$

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

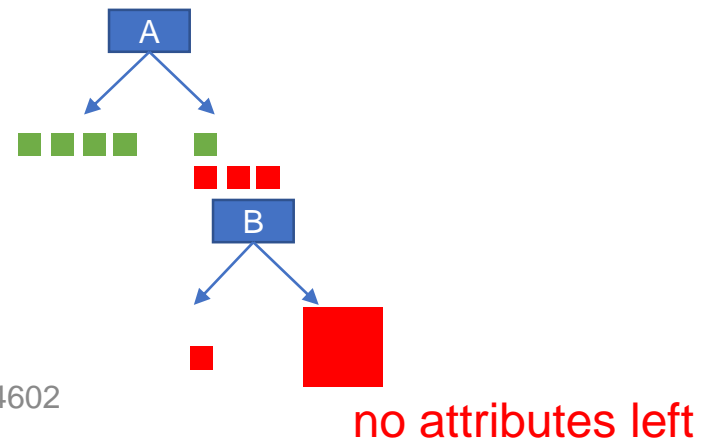
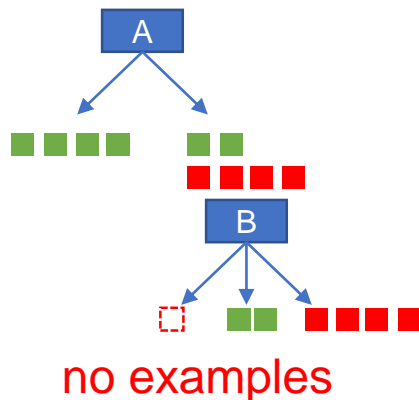
What to Do if...

- In some leaf there are no examples:

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

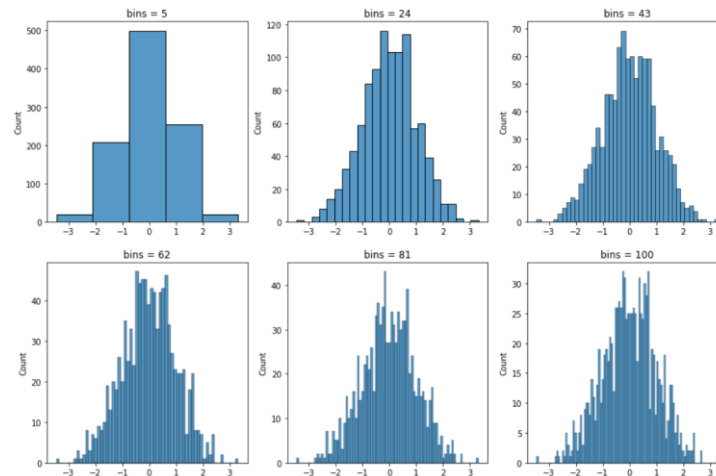
- There are no attributes left

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



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?)

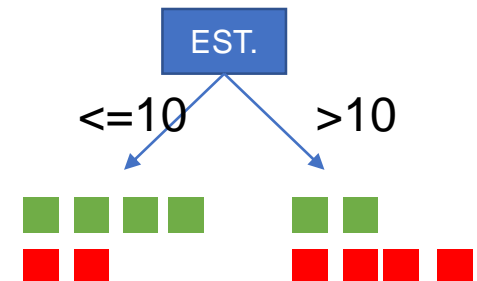
4.5 → IG
6.5 → IG
7.5 → IG

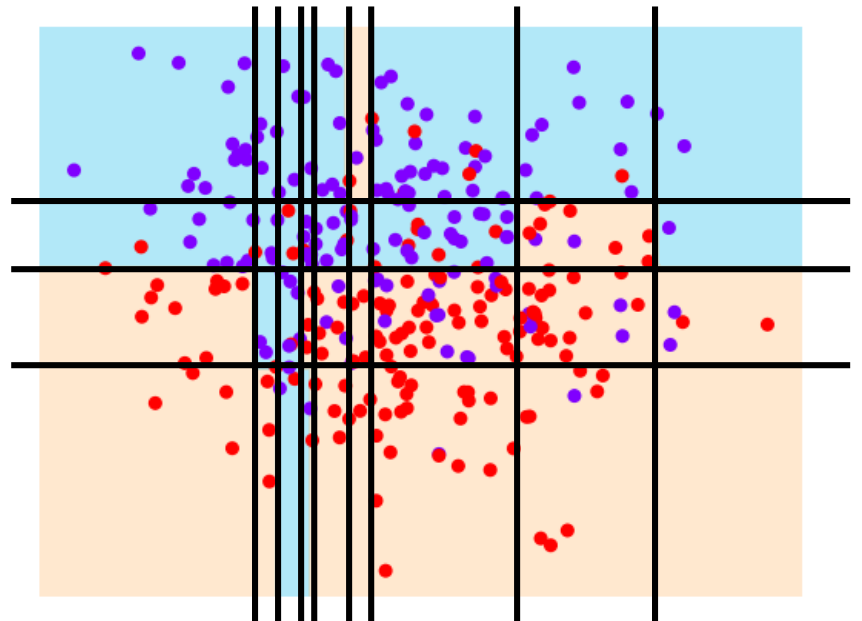
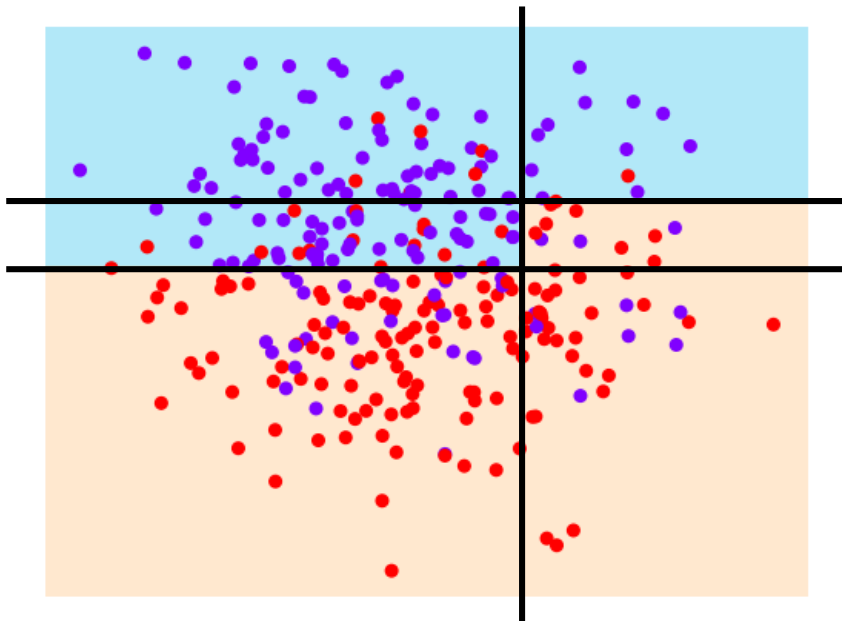
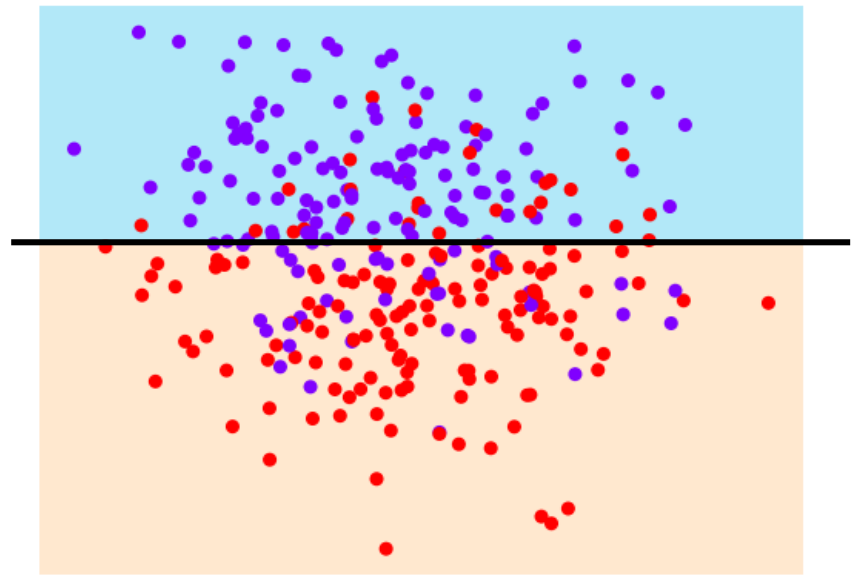
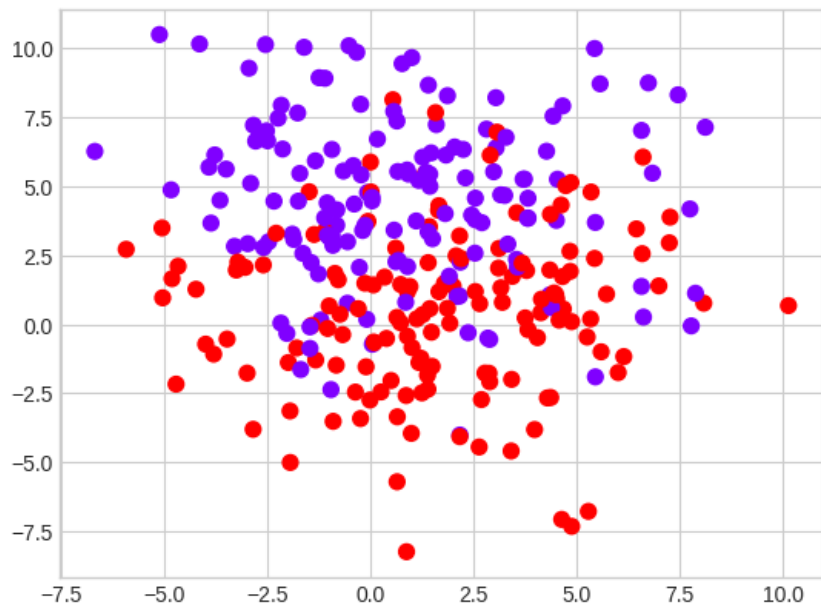
10 → IG
16 → IG
30 → IG

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 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.

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

- Do a statistical test: is the increase in information significant.

How to improve Decision Trees?



Ensemble methods: bagging, boosting

Aims

- **Bagging (bootstrap aggregating)** 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:** learn 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

- 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

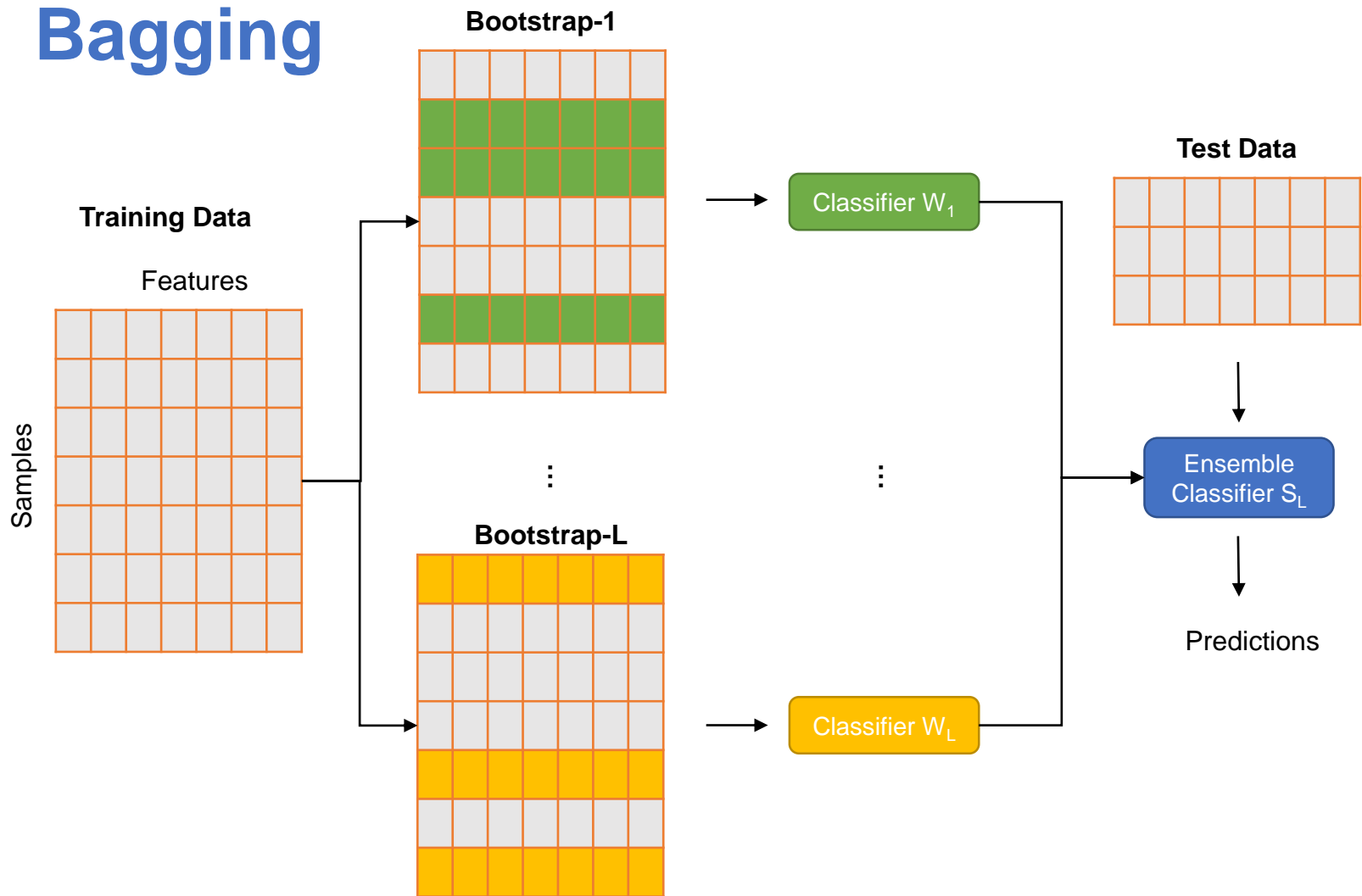
- Assuming that we have L bootstrap samples (approximations of L independent datasets)
- We can fit L almost independent weak learners (one on each dataset)

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

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

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

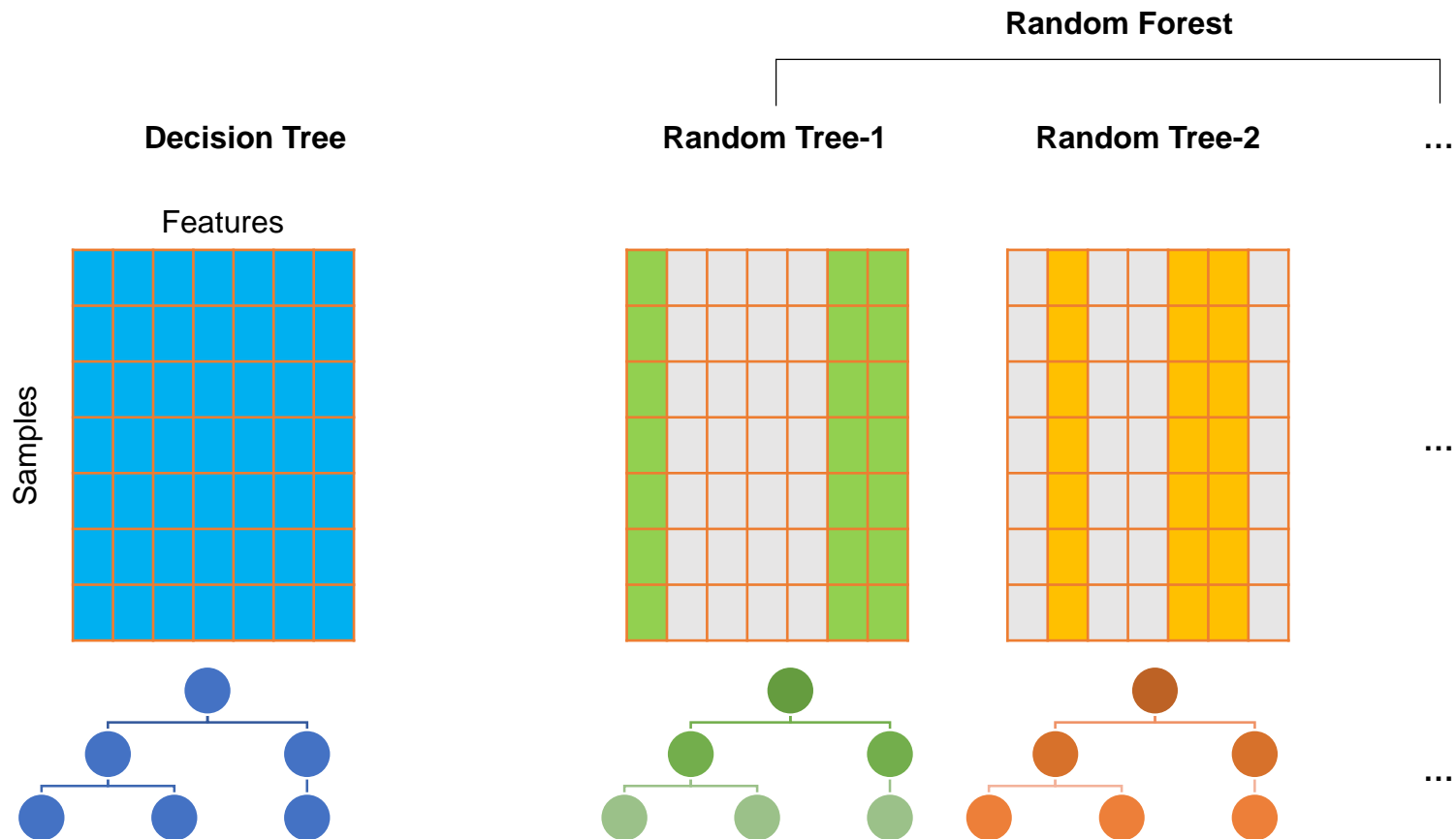
Bagging



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

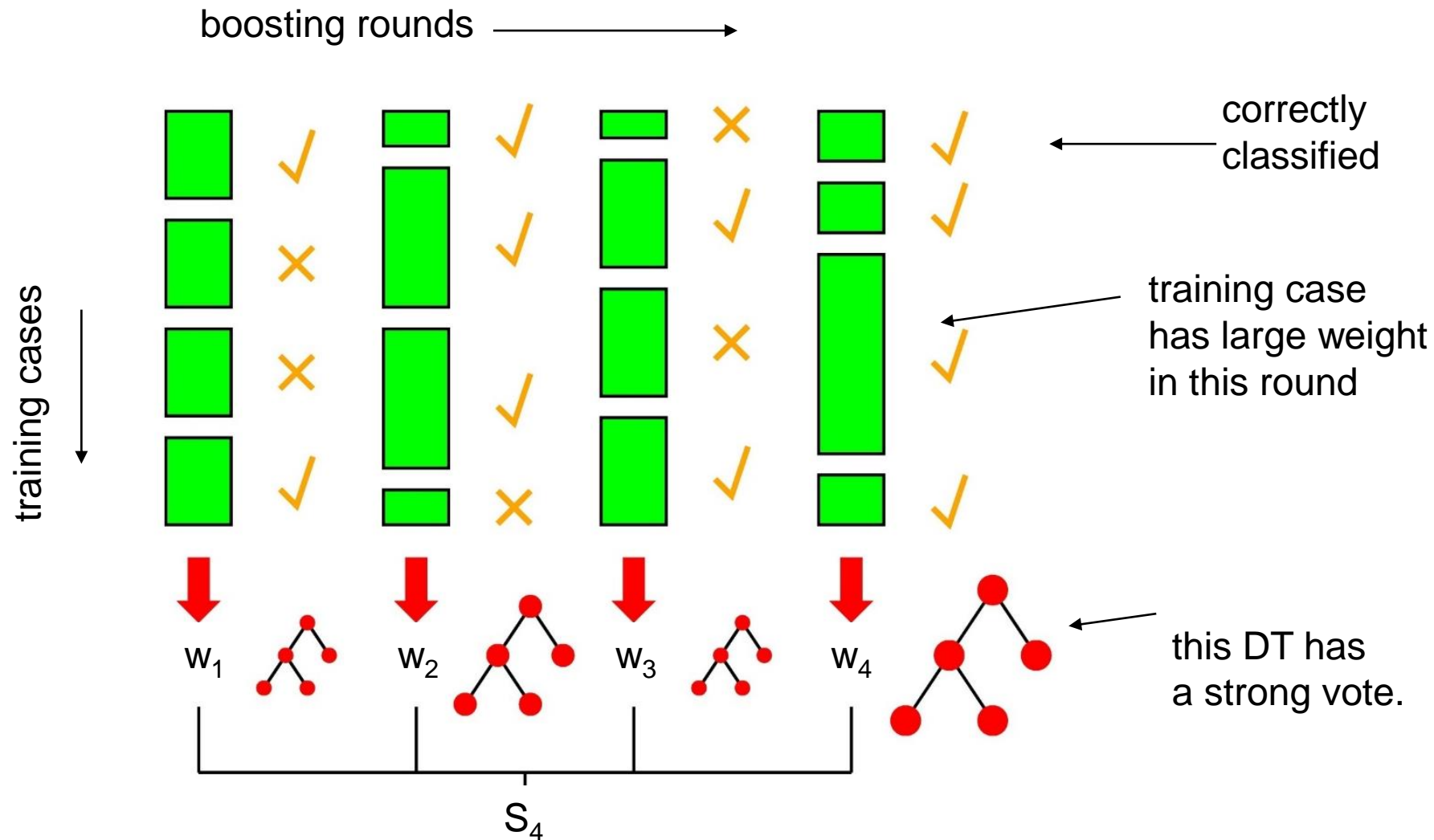
- Main idea:
 - Train classifiers (e.g. decision trees) in a sequence.
 - A new classifier should focus on those cases which were incorrectly classified in the last round.
 - Combine the classifiers by letting them vote on the final prediction (like bagging).
 - Each classifier could be (should be) very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

Boosting intuition

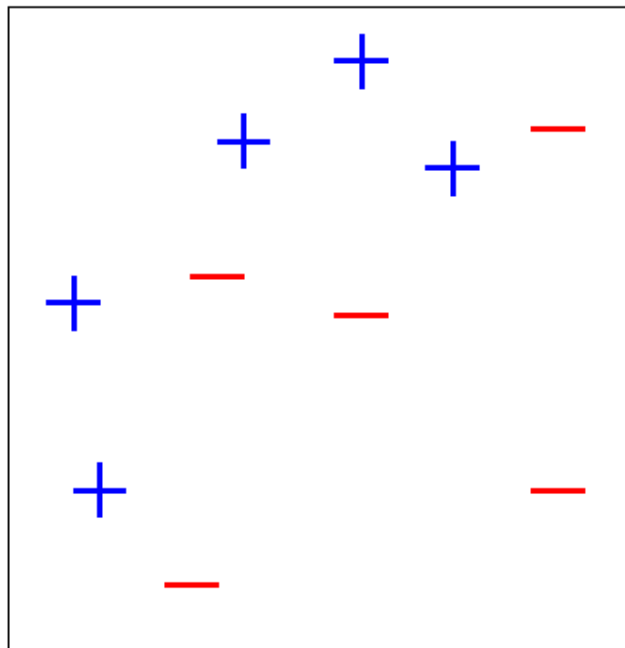
- We adaptively weigh each data case.
- Data cases which are wrongly classified get high weight (the algorithm will focus on them)
- Each boosting round learns a new (simple) classifier on the weighed dataset.
- We weigh classifiers to combine them into a single powerful classifier.
- Classifiers that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

$$\begin{array}{ccc} \text{Bagging} & & \text{Boosting} \\ s_L(.) = \sum_{l=1}^L c_l \times w_l(.) & \longrightarrow & s_l(.) = s_{l-1}(.) + c_l \times w_l(.) \end{array}$$

Boosting in a picture

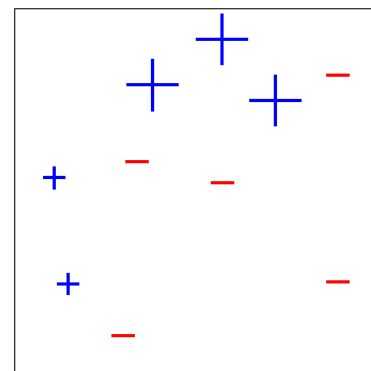
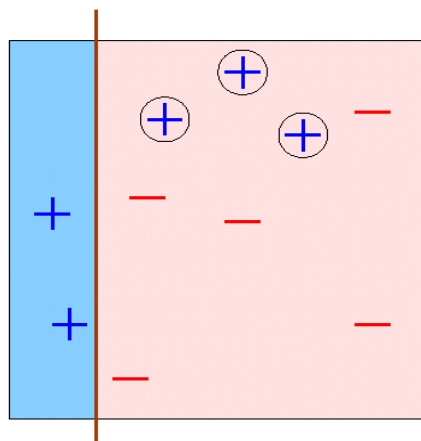


AdaBoost (Example)

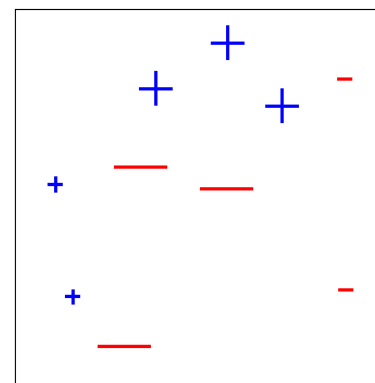
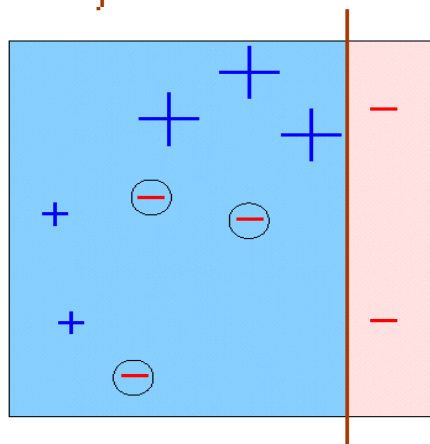


Original Training set : Equal Weights to all training samples

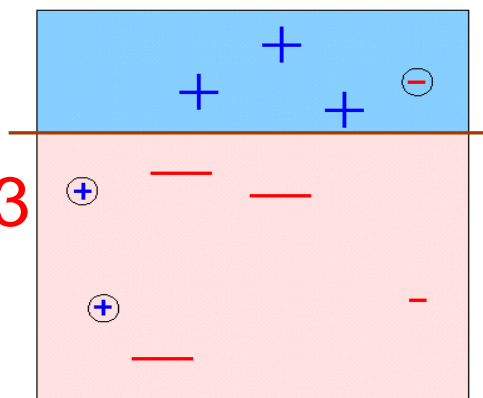
ROUND 1



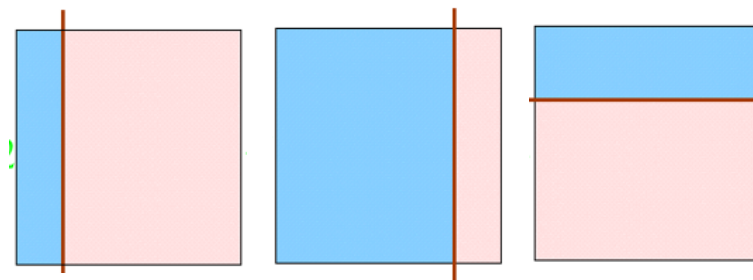
ROUND 2



ROUND 3



FINAL



AdaBoost (Algorithm)

Given: $(x_1, y_1), \dots, (x_N, y_N)$ where $x_i \in \mathbf{X}$, $y_i \in Y = \{-1, +1\}$

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

For $l = 1, \dots, L$

1. Find the classifier $w_l(x_i) \in \{-1, +1\}$ that minimizes the error with respect to the distribution D_l :

$$w_l = \arg \min_{w_j \in W} \epsilon_j = \sum_{i=1}^N D_l(i) [y(i) \neq w_j(x_i)]$$

2. Prerequisite: $\epsilon_l < 0.5$; otherwise, stop.

3. Choose $\alpha_l \in \mathbb{R}$, typically $\alpha_l = \frac{1}{2} \ln \frac{1-\epsilon_l}{\epsilon_l}$, where ϵ_l is the weighted error rate of classifier w_l .

4. Update: $D_{l+1}(i) = \frac{D_l(i) \exp(-\alpha_l y_i w_l(x_i))}{Z_l}$, where Z_l is a normalization factor

$$\exp(-\alpha_l y(i) w_l(x_i)) \begin{cases} < 1, y(i) = w_l(x_i) \\ > 1, y(i) \neq w_l(x_i) \end{cases}$$

Output the final classifier: $s_L(x) = \text{sign}(\sum_{l=1}^L \alpha_l w_l(x))$

AdaBoost (Algorithm)

Given: $(x_1, y_1), \dots, (x_N, y_N)$ where $x_i \in \mathbf{X}$, $y_i \in Y = \{-1, +1\}$

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

For $l = 1, \dots, L$

1. Find the classifier $w_l(x_i) \in \{-1, +1\}$ that minimizes the error with respect to the distribution D_l :

$$w_l = \arg \min_{w_j \in W} \epsilon_j = \sum_{i=1}^N D_l(i) [y(i) \neq w_j(x_i)]$$

2. Prerequisite: $\epsilon_l < 0.5$; otherwise, stop.

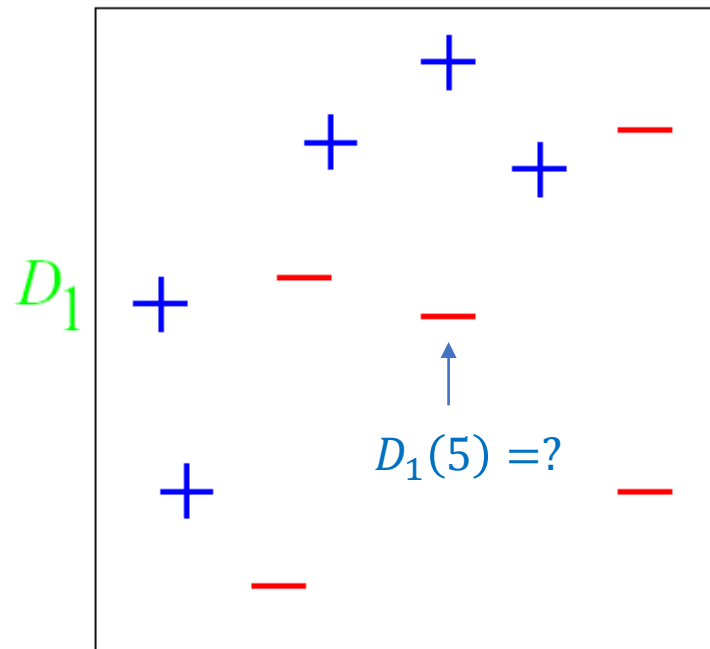
3. Choose $\alpha_l \in \mathbb{R}$, typically $\alpha_l = \frac{1}{2} \ln \frac{1 - \epsilon_l}{\epsilon_l}$, where ϵ_l is the weighted error rate of classifier w_l .

4. Update: $D_{l+1}(i) = \frac{D_l(i) \exp(-\alpha_l y_i w_l(x_i))}{Z_l}$, where Z_l is a normalization factor

$$\exp(-\alpha_l y(i) w_l(x_i)) \begin{cases} < 1, y(i) = w_l(x_i) \\ > 1, y(i) \neq w_l(x_i) \end{cases}$$

Output the final classifier: $s_L(x) = \text{sign}(\sum_{l=1}^L \alpha_l w_l(x))$

AdaBoost (Example)



Original Training set : Equal Weights to all training samples

AdaBoost (Algorithm)

Given: $(x_1, y_1), \dots, (x_N, y_N)$ where $x_i \in \mathbf{X}$, $y_i \in Y = \{-1, +1\}$

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

For $l = 1, \dots, L$

1. Find the classifier $w_l(x_i) \in \{-1, +1\}$ that minimizes the error with respect to the distribution D_l :

$$w_l = \arg \min_{w_j \in W} \epsilon_j = \sum_{i=1}^N D_l(i) [y(i) \neq w_j(x_i)]$$

2. Prerequisite: $\epsilon_l < 0.5$; otherwise, stop.

3. Choose $\alpha_l \in \mathbb{R}$, typically $\alpha_l = \frac{1}{2} \ln \frac{1-\epsilon_l}{\epsilon_l}$, where ϵ_l is the weighted error rate of classifier w_l .

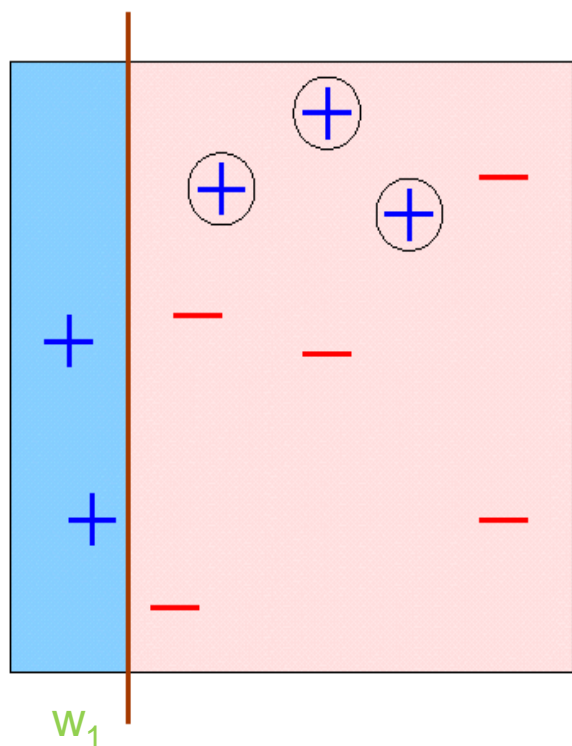
4. Update: $D_{l+1}(i) = \frac{D_l(i) \exp(-\alpha_l y_i w_l(x_i))}{Z_l}$, where Z_l is a normalization factor

$$\exp(-\alpha_l y(i) w_l(x_i)) \begin{cases} < 1, y(i) = w_l(x_i) \\ > 1, y(i) \neq w_l(x_i) \end{cases}$$

Output the final classifier: $s_L(x) = \text{sign}(\sum_{l=1}^L \alpha_l w_l(x))$

AdaBoost (Example)

ROUND 1

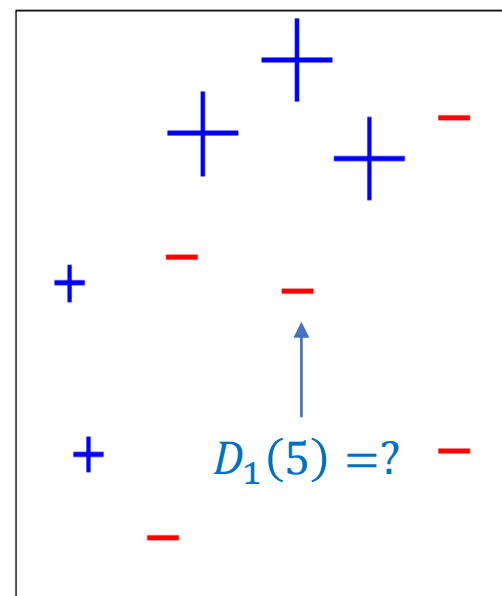


$$\epsilon_1 = 0.30$$

$$\alpha_1 = 0.42$$



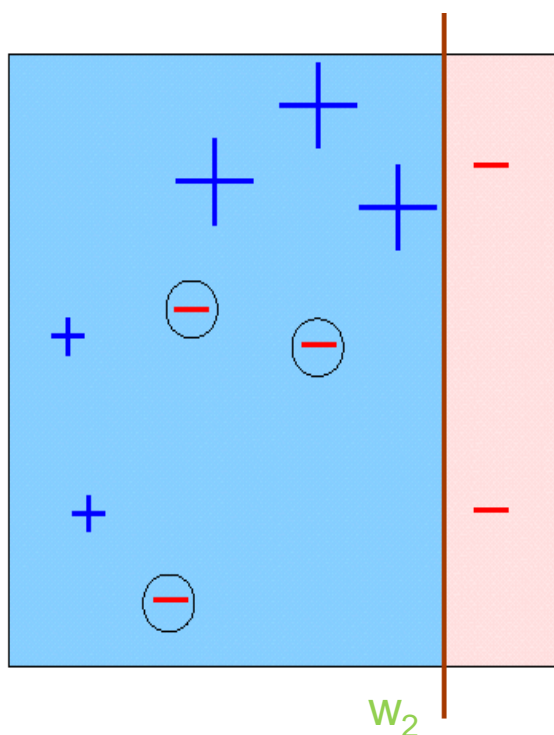
D_2



$$\alpha_l = \frac{1}{2} \ln \frac{1 - \epsilon_l}{\epsilon_l}$$

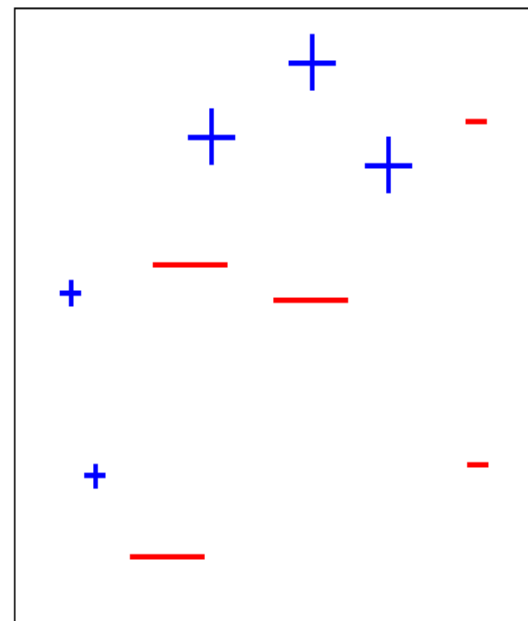
AdaBoost (Example)

ROUND 2



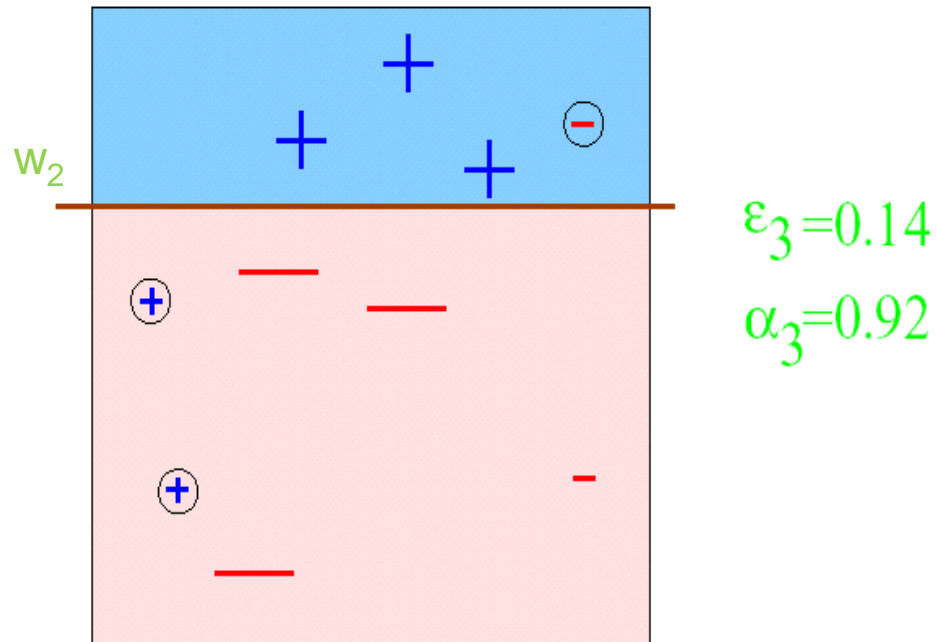
$$\epsilon_2 = 0.21$$
$$\alpha_2 = 0.65$$

D_3



AdaBoost (Example)

ROUND 3



AdaBoost (Algorithm)

Given: $(x_1, y_1), \dots, (x_N, y_N)$ where $x_i \in \mathbf{X}$, $y_i \in Y = \{-1, +1\}$

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

For $l = 1, \dots, L$

1. Find the classifier $w_l(x_i) \in \{-1, +1\}$ that minimizes the error with respect to the distribution D_l :

$$w_l = \arg \min_{w_j \in W} \epsilon_j = \sum_{i=1}^N D_l(i) [y(i) \neq w_j(x_i)]$$

2. Prerequisite: $\epsilon_l < 0.5$; otherwise, stop.

3. Choose $\alpha_l \in \mathbb{R}$, typically $\alpha_l = \frac{1}{2} \ln \frac{1 - \epsilon_l}{\epsilon_l}$, where ϵ_l is the weighted error rate of classifier w_l .

4. Update: $D_{l+1}(i) = \frac{D_l(i) \exp(-\alpha_l y_i w_l(x_i))}{Z_l}$, where Z_l is a normalization factor

$$\exp(-\alpha_l y(i) w_l(x_i)) \begin{cases} < 1, y(i) = w_l(x_i) \\ > 1, y(i) \neq w_l(x_i) \end{cases}$$

Output the final classifier: $s_L(x) = \text{sign}(\sum_{l=1}^L \alpha_l w_l(x))$

AdaBoost (Example)

$$s_L(x) = \text{sign} \left(0.42 \begin{array}{|c|} \hline \text{blue} \\ \hline \end{array} + 0.65 \begin{array}{|c|} \hline \text{blue} \\ \hline \end{array} + 0.92 \begin{array}{|c|} \hline \text{blue} \\ \hline \end{array} \right)$$

The diagram shows three weak classifiers, each represented by a square divided by a vertical line. The first classifier has a weight of 0.42, the second 0.65, and the third 0.92. The regions are colored blue and red. The final decision is the sign of the weighted sum of these classifiers.

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

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

→

AdaBoost

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

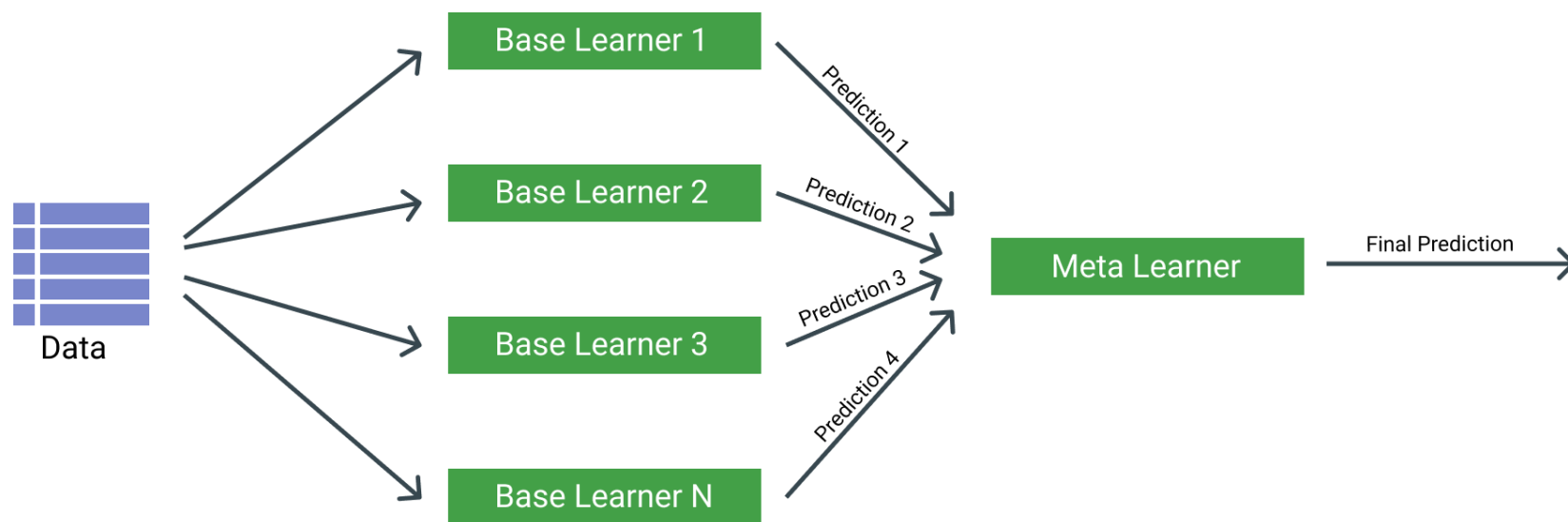
↓

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

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

Stacking



What have we learnt

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

Questions?



model trained
for 1000 epochs



model trained
for 100 epochs



model trained
for 1 epoch

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