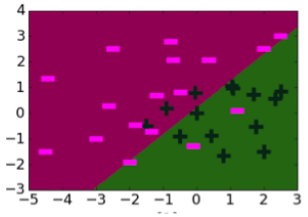
ENCS5341
Machine Learning and Data Science

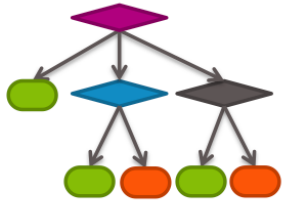
Ensemble Methods

Introduction

Simple (weak) classifiers are good!



Logistic regression w. simple features



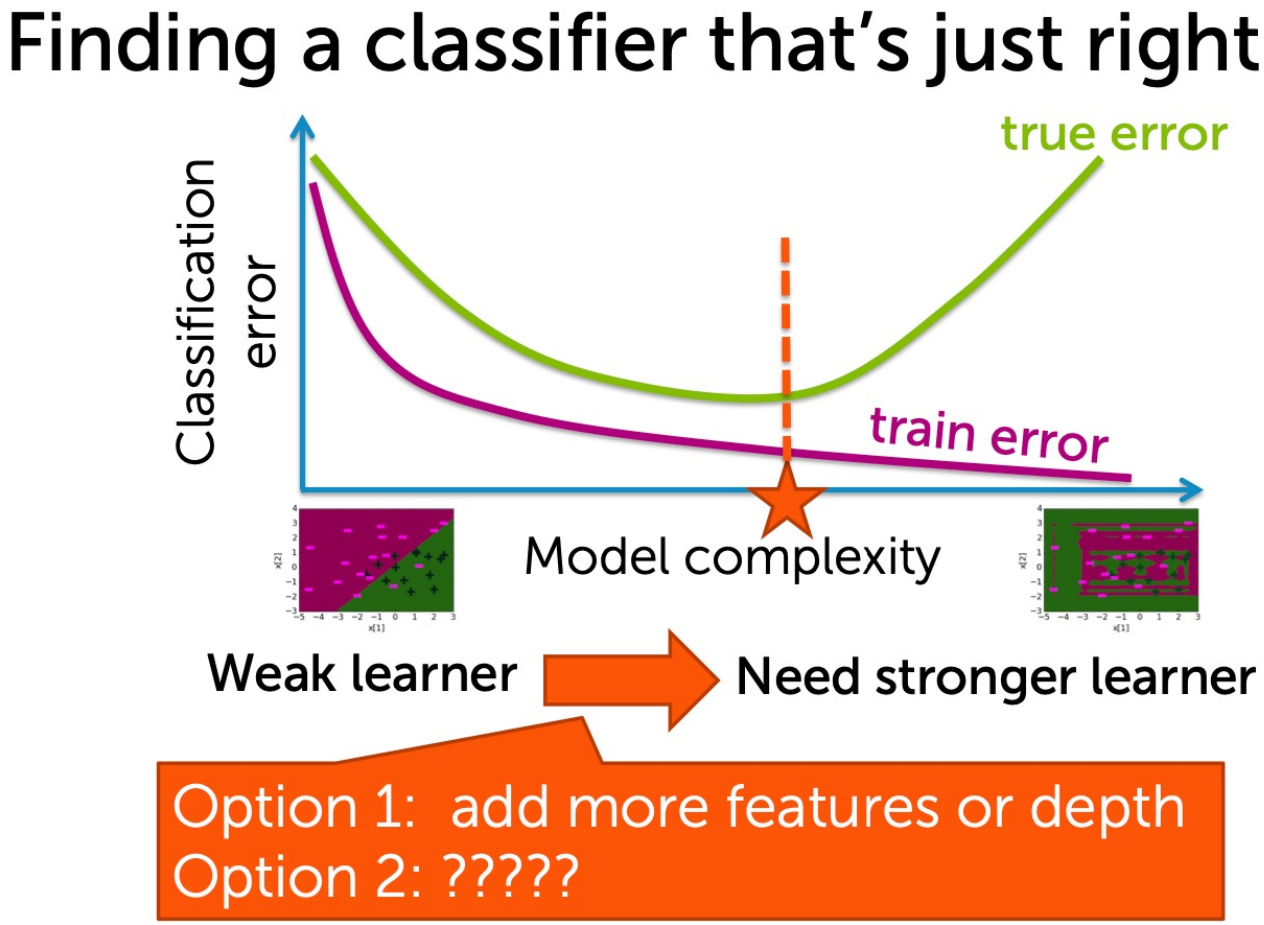
Shallow decision trees

Decision stumps

Low variance. Learning is fast!

But high bias...

Introduction



Ensemble Learning

- Ensemble learning uses multiple weak classifiers and combine their predictions to get a stronger model.
- If different models make different mistakes, can we simply average the predictions?
- Voting classifiers: gives every model a vote on the class labels:
  - Hard vote: majority class wins.
  - Soft vote: average the class probabilities from the different models and select the class with highest average probability.
- Why does this work?
  - Different models might be good at different parts of the data.
  - Individual mistakes can be averaged out.
- Models must be uncorrelated but good enough (otherwise the ensemble is worse).

Ensemble Learning

Which models should we combine?

- If a model underfits (high bias, low variance), combine with other low variance models
  - Need to be different (experts on different parts of the data).
  - Bias reduction can be done with **Boosting.**

- If a model overfits (low bias, high variance) combine with other low bias models
  - Need to be different (each model mistakes must be different).
  - Variance reduction can be done with **Bagging**.

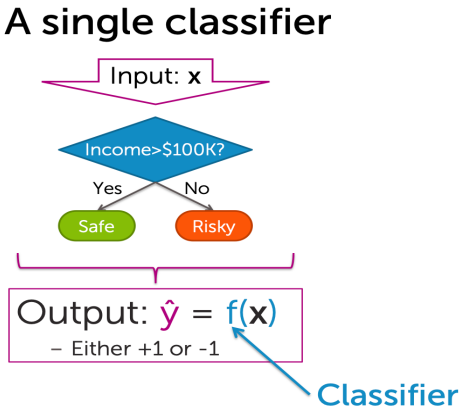
For example, shallow trees have high bias and low variance, whereas deep trees have low bias and high variance.

- We can combine multiple shallow trees via boosting (e.g AdaBoost).
- We can combine multiple deep trees via bagging (e.g. Random Forest).

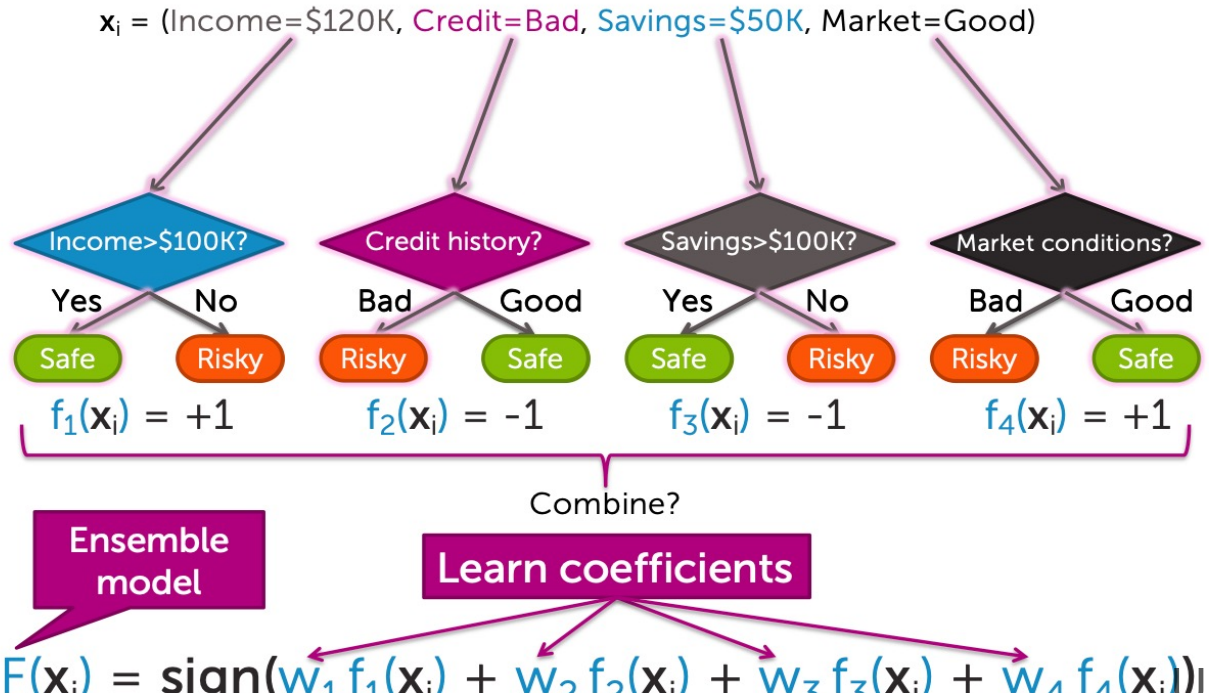
Boosting

Boosting Formulation

- Train a single simple classifier (for example a decision stump)

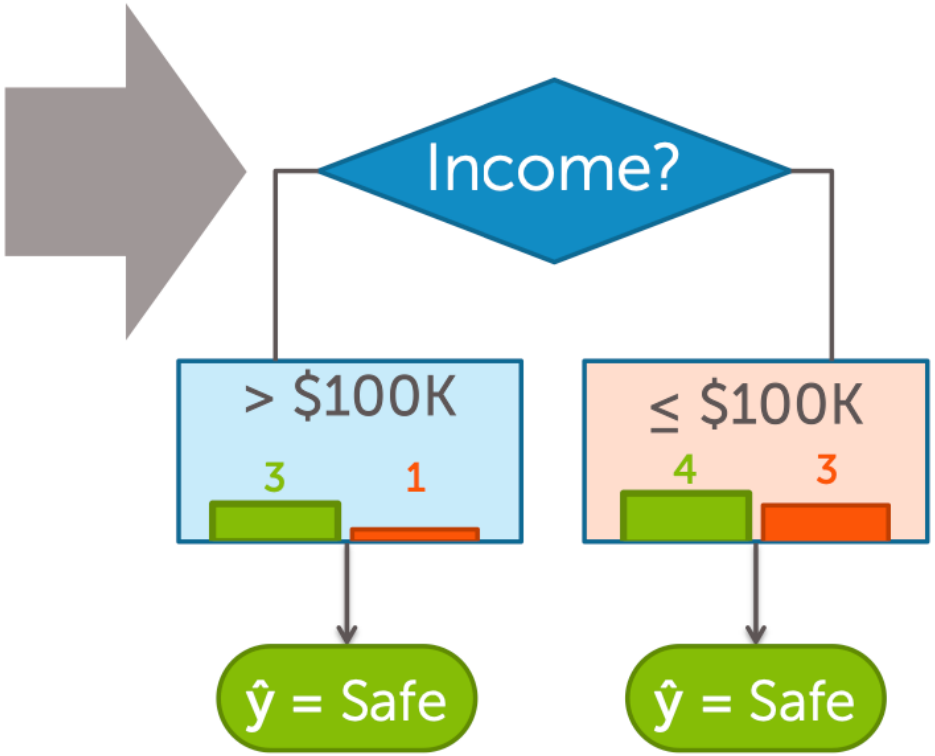


- Combine different classifiers to get a strong a classifier using an additive model



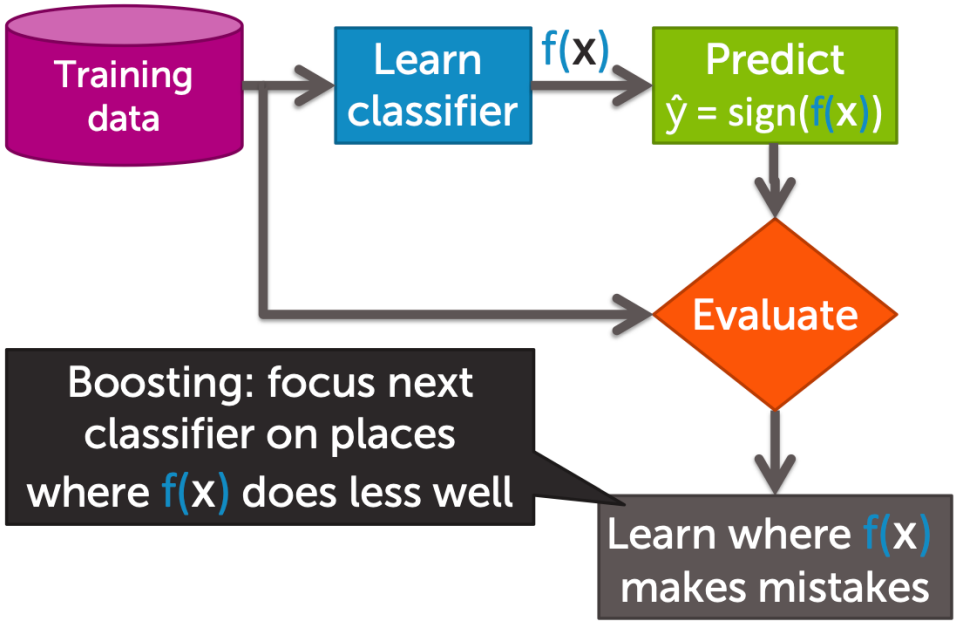
Aside: Learning a decision stump

Credit	Income	y
A	$130K	Safe
B	$80K	Risky
C	$110K	Risky
A	$110K	Safe
A	$90K	Safe
B	$120K	Safe
C	$30K	Risky
C	$60K	Risky
B	$95K	Safe
A	$60K	Safe
A	$98K	Safe



Boosting

- Boosting = Focus learning on “hard” points

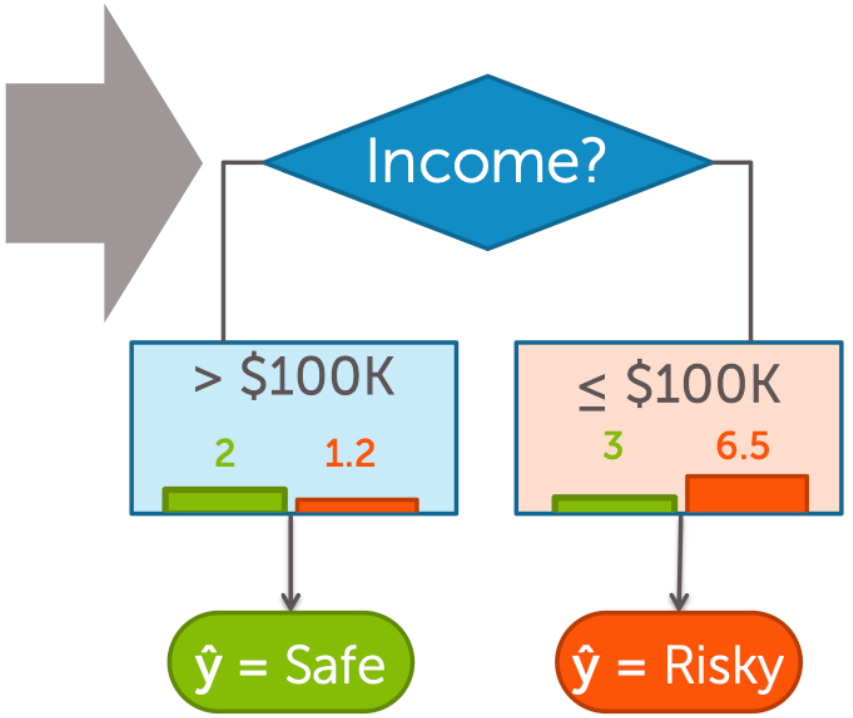


- We focus on hard examples by learning on weighted data
  - More weights on hard or more important data.
  - Each data point x_i is weighted by α_i (more important point = higher weight).
  - During learning, each data point x_i is counted as α_i points.

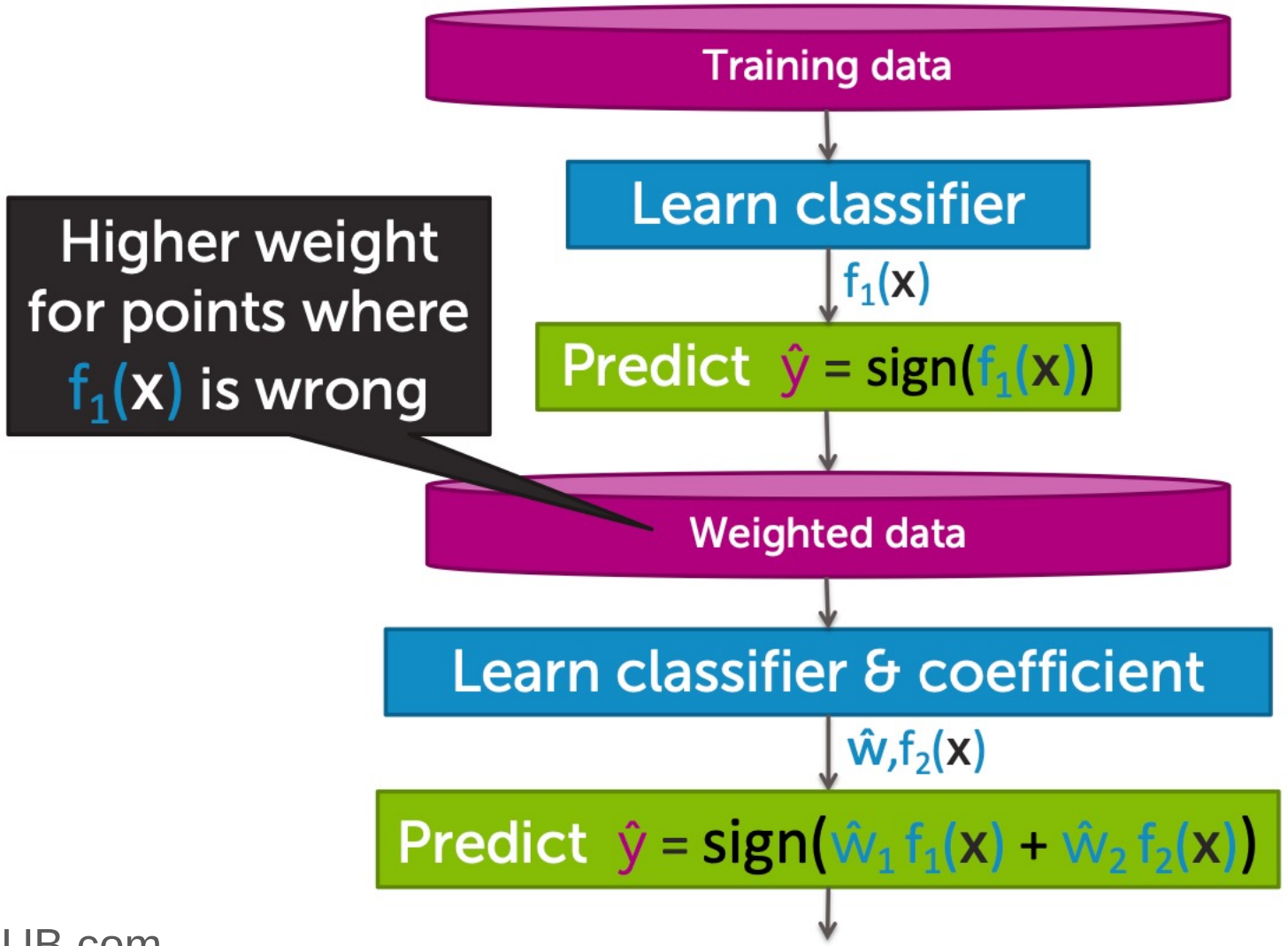
Aside: Learning a decision stump on weighted data

Increase weight α of harder/misclassified points

Credit	Income	y	Weight α
A	$130K	Safe	0.5
B	$80K	Risky	1.5
C	$110K	Risky	1.2
A	$110K	Safe	0.8
A	$90K	Safe	0.6
B	$120K	Safe	0.7
C	$30K	Risky	3
C	$60K	Risky	2
B	$95K	Safe	0.8
A	$60K	Safe	0.7
A	$98K	Safe	0.9

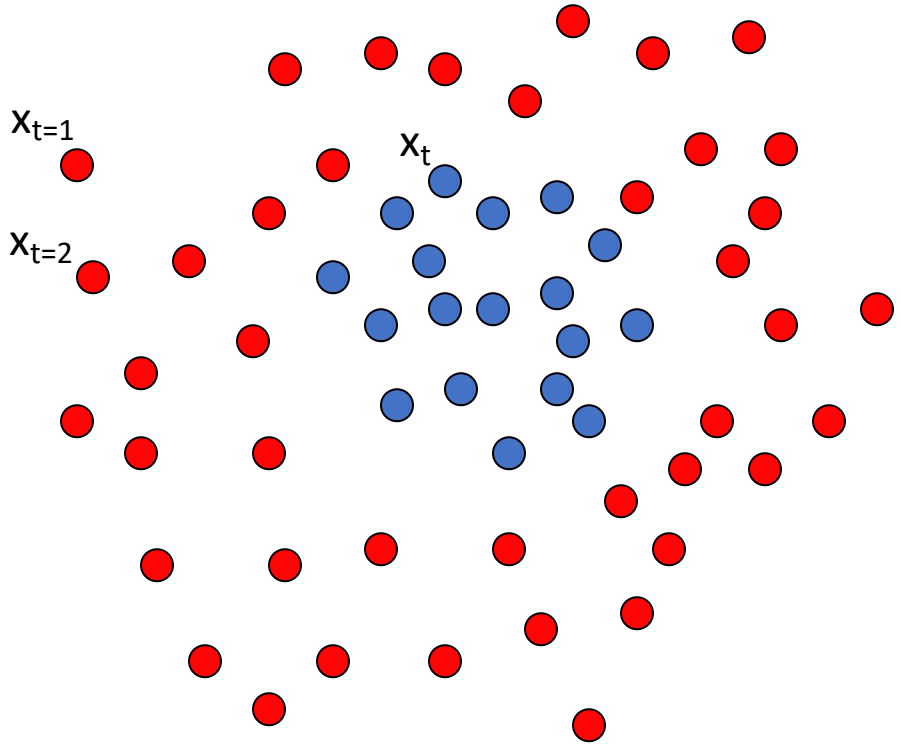


Boosting = gready learning ensembles from data



Boosting: Toy Example

- It is a sequential procedure:



Each data point has

a class label:

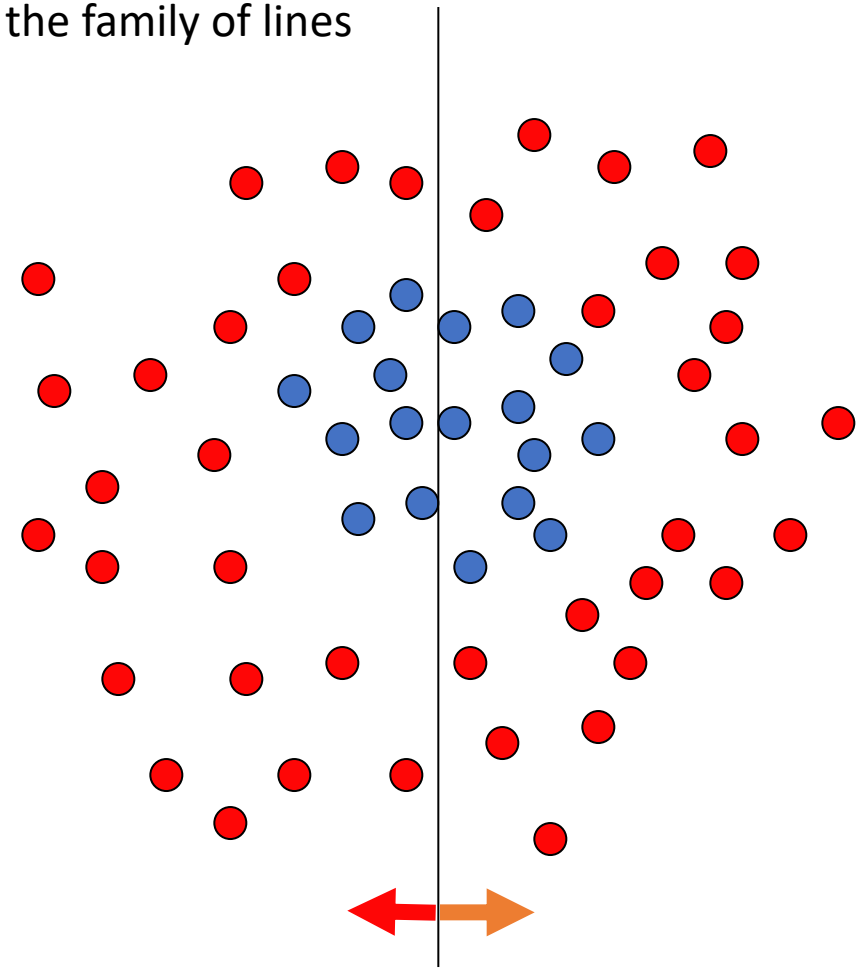
$$y_t = \begin{cases} +1 & \\ -1 & \end{cases}$$

and a weight:

$w_t = 1$

Boosting: Toy Example

Weak learners from the family of lines



Each data point has

a class label:

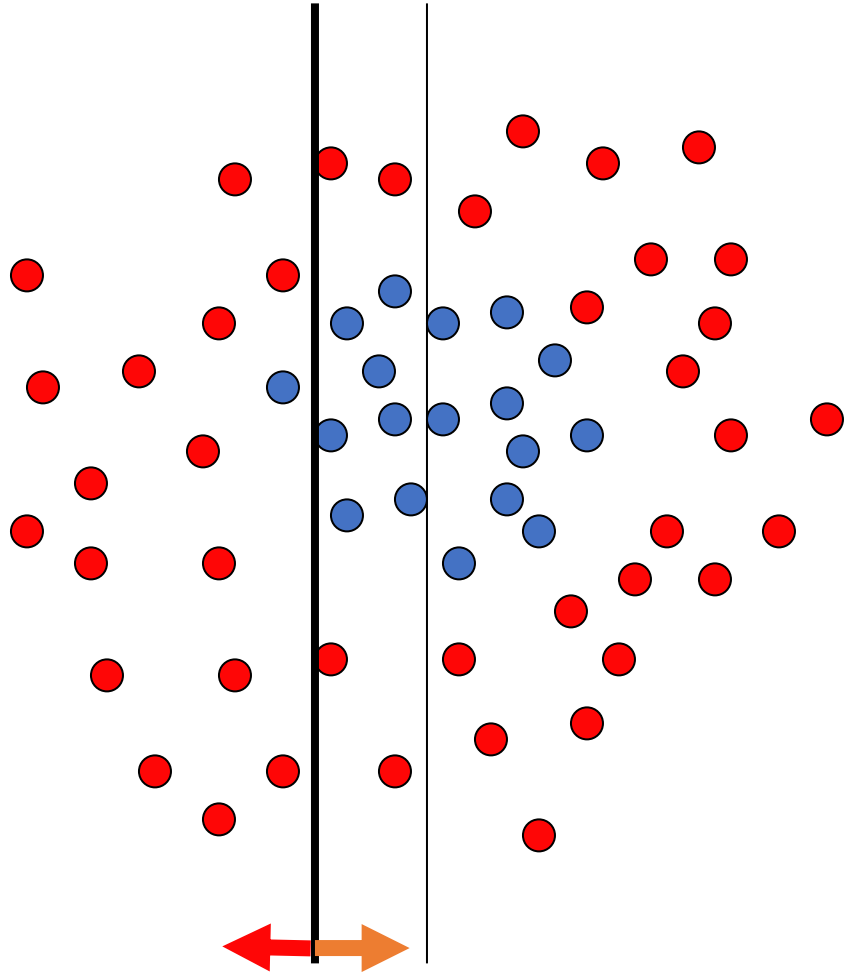
$$y_t = \begin{cases} +1 & (\bullet) \\ -1 & (\bullet) \end{cases}$$

and a weight:

$w_t = 1$

h => p(error) = 0.5 it is at chance

Boosting: Toy Example



Each data point has

a class label:

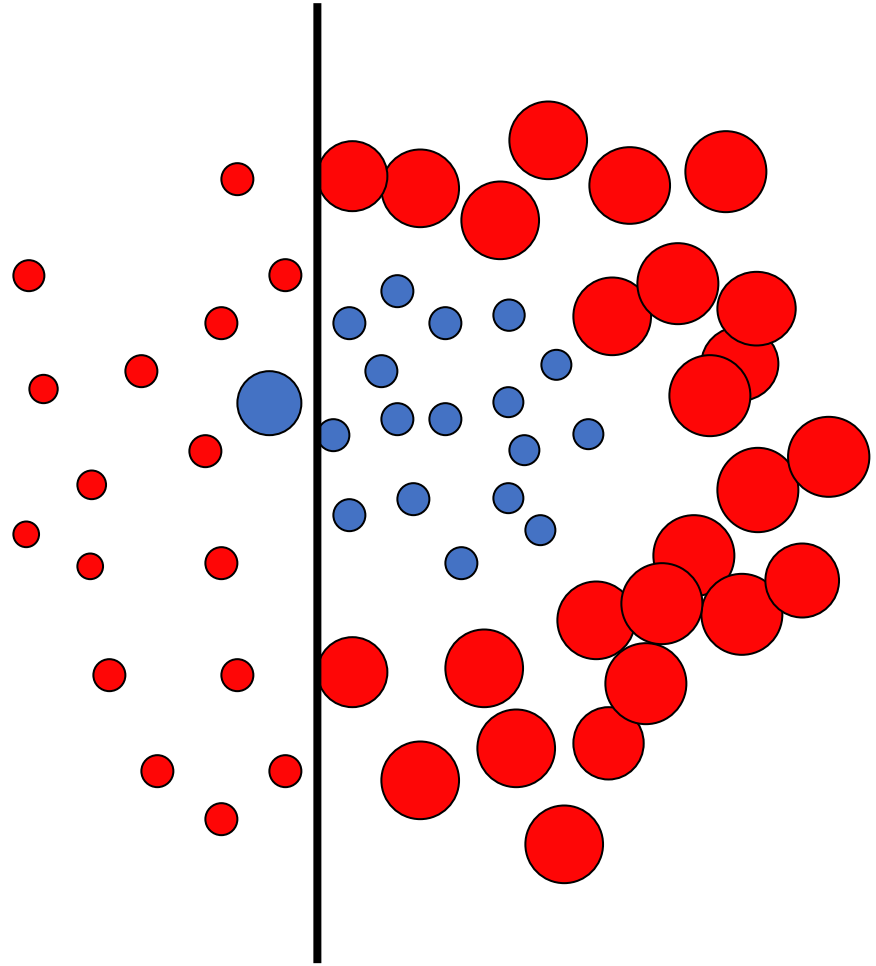
$y_t = \begin{cases} +1 & (\bullet) \\ -1 & (\bullet) \end{cases}$

and a weight:

$w_t = 1$

This is a '**weak classifier**': It performs slightly better than chance.

Boosting: Toy Example



Each data point has

a class label:

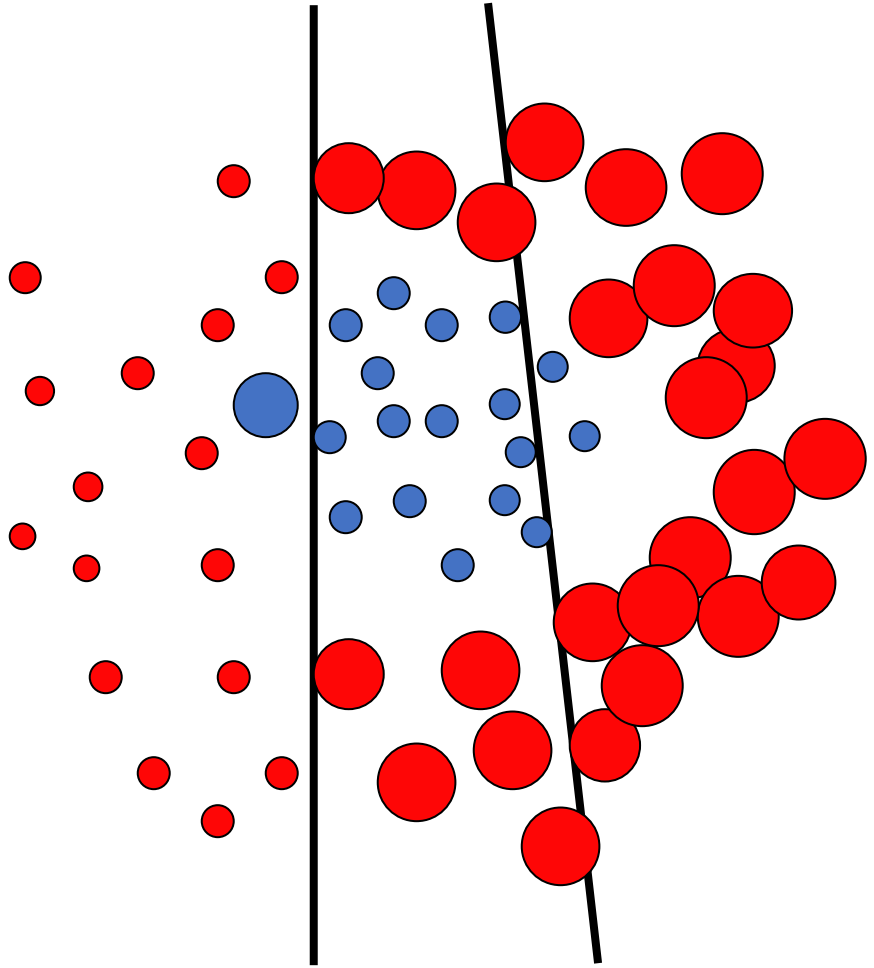
$$y_t = \begin{cases} +1 & (\bigcirc) \\ -1 & (\bigcirc) \end{cases}$$

We update the weights:

$$w_t \leftarrow w_t \exp\{-y_t F_t\}$$

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

Boosting: Toy Example



Each data point has

a class label:

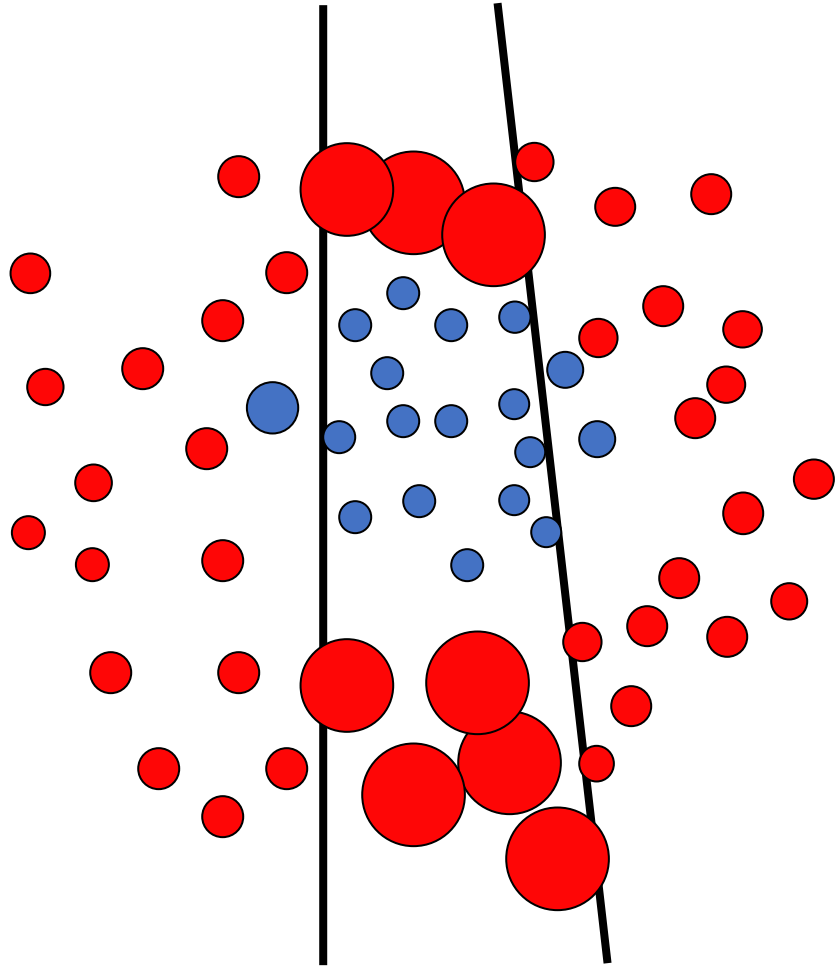
$$y_t = \begin{cases} +1 & (\bigcirc) \\ -1 & (\bigcirc) \end{cases}$$

We update the weights:

$$w_t \leftarrow w_t \exp\{-y_t F_t\}$$

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

Boosting: Toy Example



Each data point has

a class label:

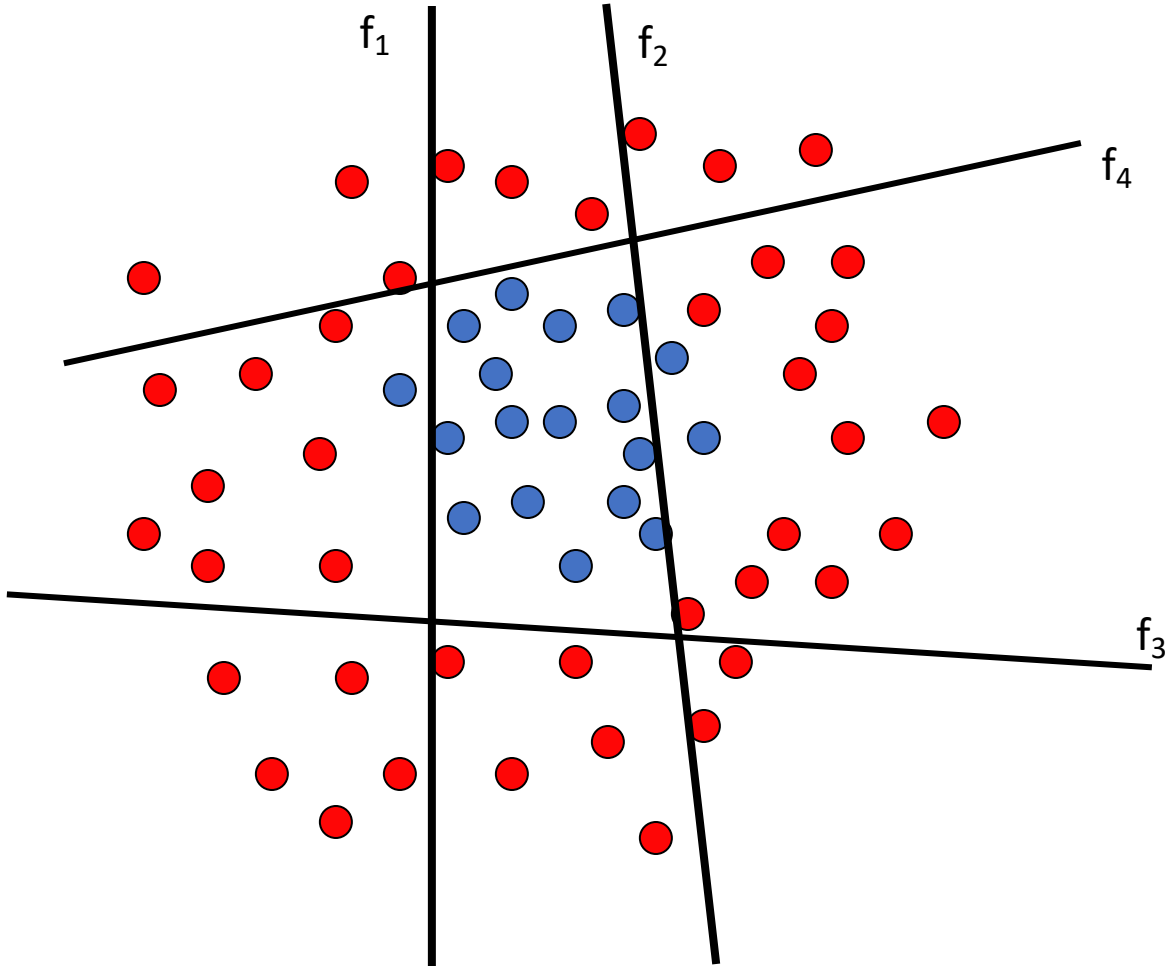
$$y_t = \begin{cases} +1 & (\bullet) \\ -1 & (\bullet) \end{cases}$$

We update the weights:

$$w_t \leftarrow w_t \exp\{-y_t F_t\}$$

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

Boosting: Toy Example



The strong (non- linear) classifier is built as the combination of all the weak (linear) classifiers.

Flavors of boosting

- AdaBoost (Freund and Shapire, 1995)
- Real AdaBoost (Friedman et al, 1998)
- LogitBoost (Friedman et al, 1998)
- Gentle AdaBoost (Friedman et al, 1998)
- BrownBoosting (Freund, 2000)
- FloatBoost (Li et al, 2002)
- …

AdaBoost Algorithm

- **Given** $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \epsilon X$, $y_i \epsilon \{-1, +1\}$
- **Initialise** weights $D_1(i) = 1/m$
- **Iterate** *t=1,…,T:*
  - Train weak learner using distribution ***Dt***
  - Get weak classifier: $h_t : X \rightarrow \mathrm{R}$
  - 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 (chosen so that D_{t+1} will be a distribution), and α_t:
$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) > 0$$
- **Output** – the final classifier
$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Bagging

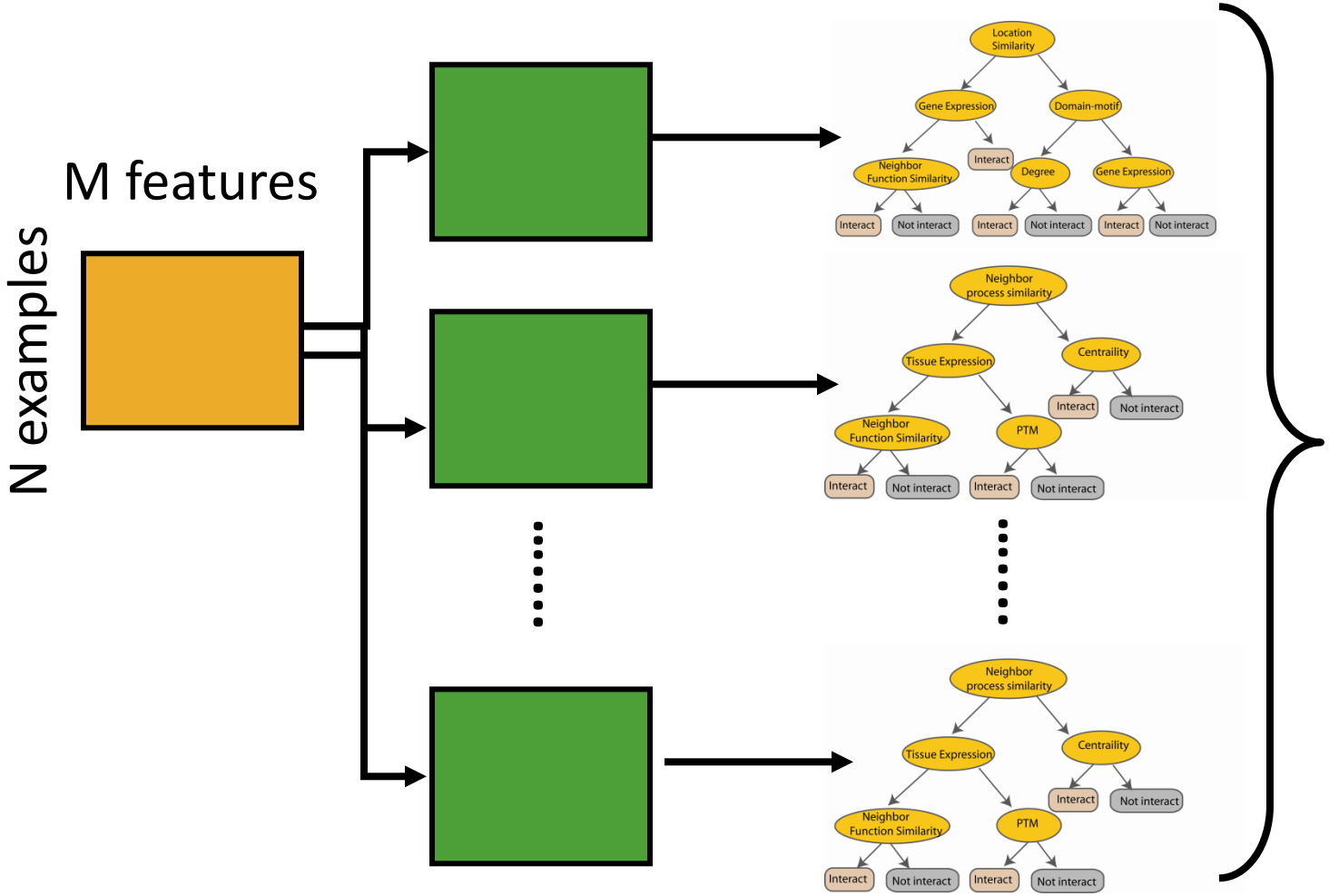
Bagging

- Obtain different models by training the same model on different training sample.
  - Reduce overfitting by averaging out individual predictions.
- In practice: take M bootstraps samples of your data, train a model on each bootstrap.
- Final prediction is obtained by averaging predictions from base models.
  - Soft voting (or majority) for classification.
  - Mean value for regression.
- Can produce uncertainty estimate as well by combining class probabilities of individual models.

Bagging - Aggregate Bootstrapping

- Given a standard training set D of size n
- For i = 1 .. M
  - Draw a sample of size $n_i \leq n$ from D uniformly and with replacement
  - Learn classifier C_i on the n_i samples
- Final classifier is a vote of C_1 .. C_M

Bagging Tree Example

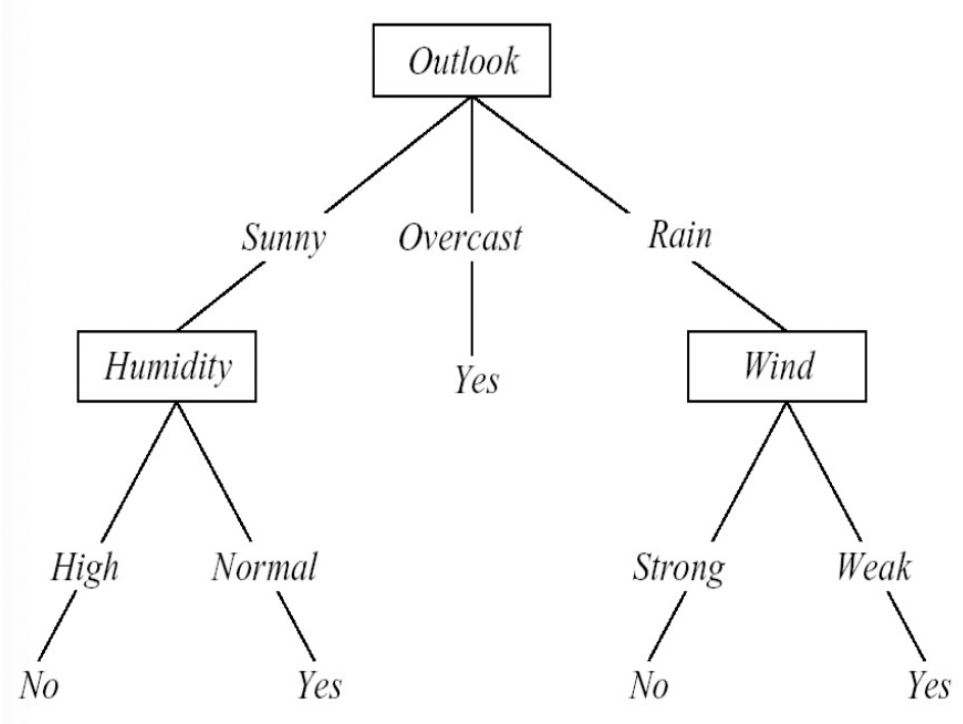


Decision Trees Review

Introduction

- *Decision tree learning is one of the most widely used techniques for classification.*

- *The classification model is a tree, and at each node a decision has to be made, hence the name Decision Tree.*



The PlayTennis Example

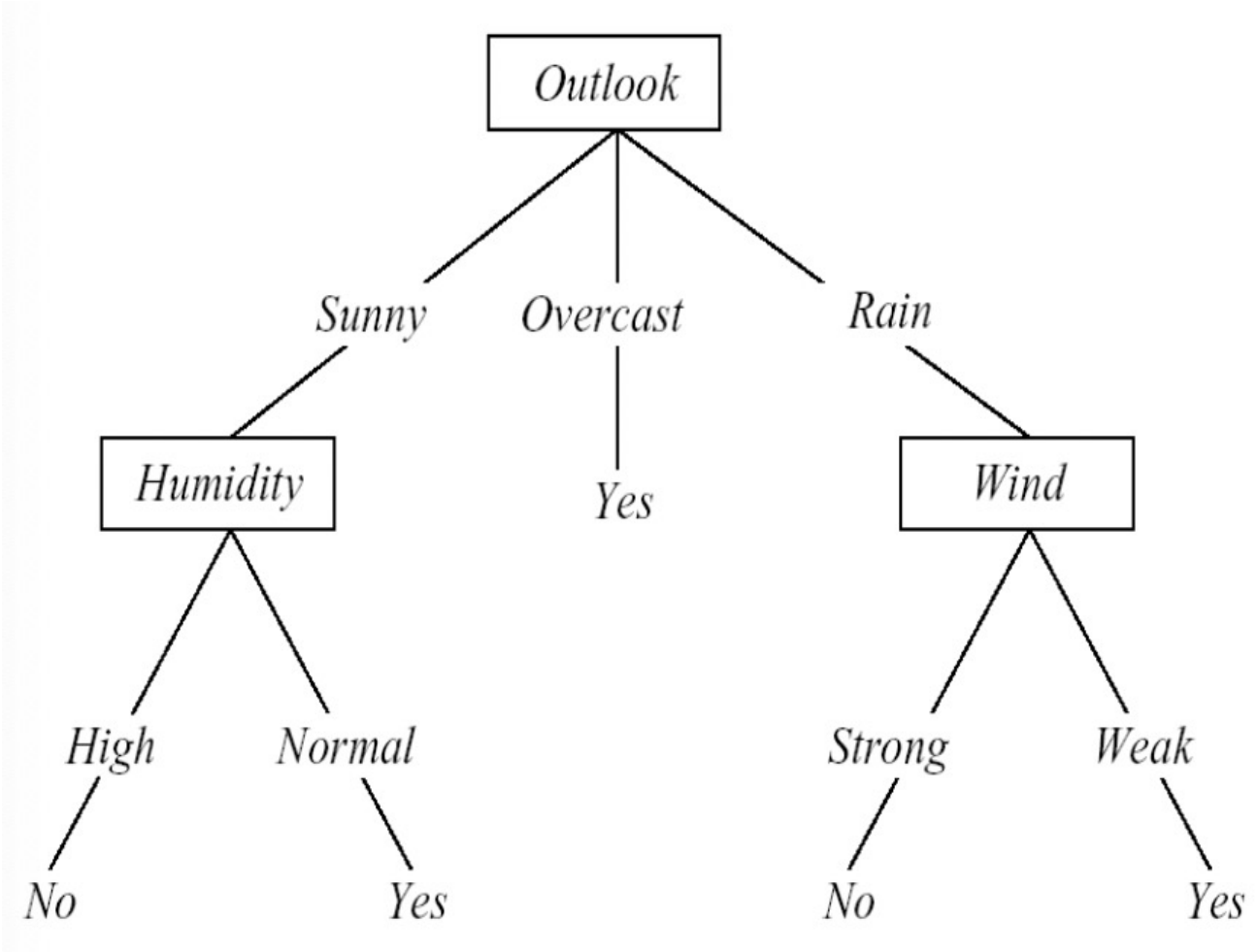
We want to predict if a person is going to play tennis on a specific day given his behavior in the past.

Attributes/Features

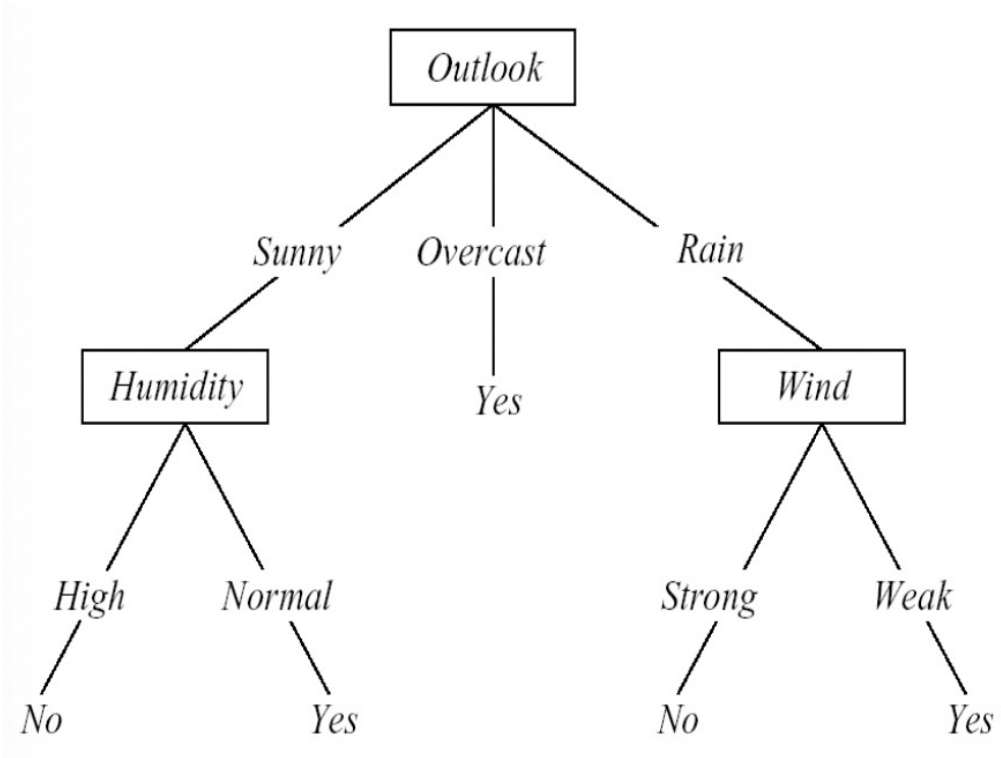
Target

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Decision Tree for PlayTennis



Decision Tree for PlayTennis



Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No
D15	Rain	Hot	High	Weak	? Yes

Learning Decision Trees

- Problem: find a decision tree that agrees with the training set.

- Trivial solution: construct a tree with one branch for each sample of the training set
  - works perfectly for the samples in the training set
  - may not work well for new samples (generalization)
  - results in relatively large tree

- Better solution: find a concise tree that still agrees with all samples
  - corresponds to the simplest hypothesis that is consistent with the training set

Constructing Decision Trees

- The most likely hypothesis is the simplest one that is consistent with all observations
  - general principle for inductive learning.
  - a simple hypothesis that is consistent with all observations is more likely to be correct than a complex one.

- In general, constructing the smallest possible decision tree is an intractable problem.

- Algorithms exist for constructing reasonably small trees.

- basic idea: test the most important attribute first

Algorithm for decision tree learning

Recursive Formulation:

- select the best attribute to split positive and negative examples.
- if only positive or only negative examples are left, we are done.
- if we have positive and negative examples left, but no attributes to split them we are in trouble
  - samples have the same description, but different classifications
  - may be caused by incorrect data (noise), or by a lack of information, or by a truly non-deterministic domain

Top-Down Induction of Decision Trees

- **Let T := Node := a decision tree consisting of an empty root node**
- **RETURN TDIDT(E,Atts,T,Node)**

E: examples

Atts: attributes

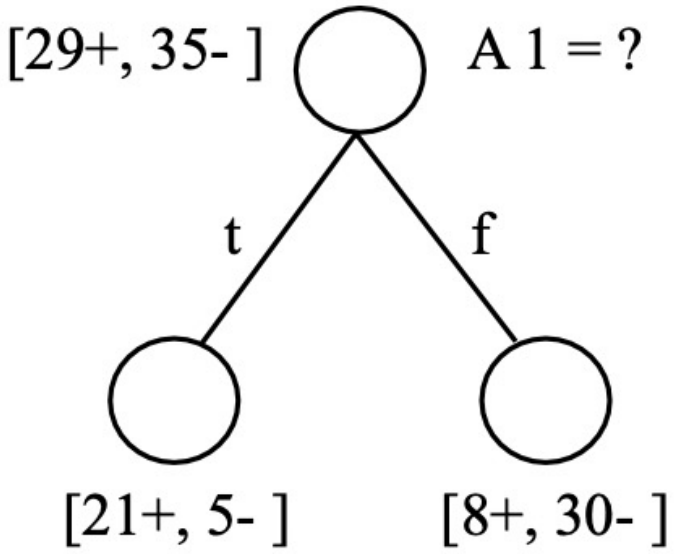
TDIDT(E,Atts,T,Node):

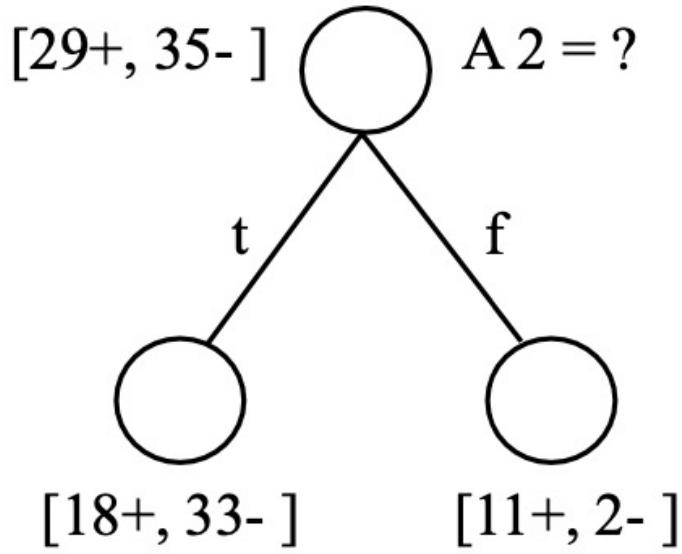
- **IF D "perfectly classified" (contains only examples of class c)**
  - **THEN make Node a leaf node in T, set class(Node) to c, RETURN T**
- **IF no test splits the data**
  - **THEN make Node a leaf node in T, set class(Node) to the majority class of E, RETURN T**
- **Select Test, the "best" test for Node based on attributes in Attr**
- **Assign Test as test for Node in T**
- **Let Descendants = {For each value of Test, create new descendant of Node in T}**
- **FOR D in Descendants**

  LET E(D) the examples that match D's test value

  TDIDT(E(D),Atts,T,D).
- **RETURN T**

Which attribute is best?





Which attribute is best?

- The key to building a decision tree - which attribute to choose in order to branch.
- The objective is to reduce impurity or uncertainty in data as much as possible.
  - A subset of data is pure if all instances belong to the same class.
- A commonly used heuristic is to choose the attribute with the maximum Information Gain or Gain Ratio based on information theory.

Entropy

- Entropy (S) = expected number of bits needed to encode class (+ or -) of randomly drawn member of D (under the optimal, shortest-length code).
- Information theory: optimal length code assigns $-\log_2 p$ bits to message having probability p.
- So, expected number of bits to encode + or - of random member of S:

$$p_+ (-\log_2 p_+) + p_- (-\log_2 p_-)$$

$$\text{Entropy (S)} = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

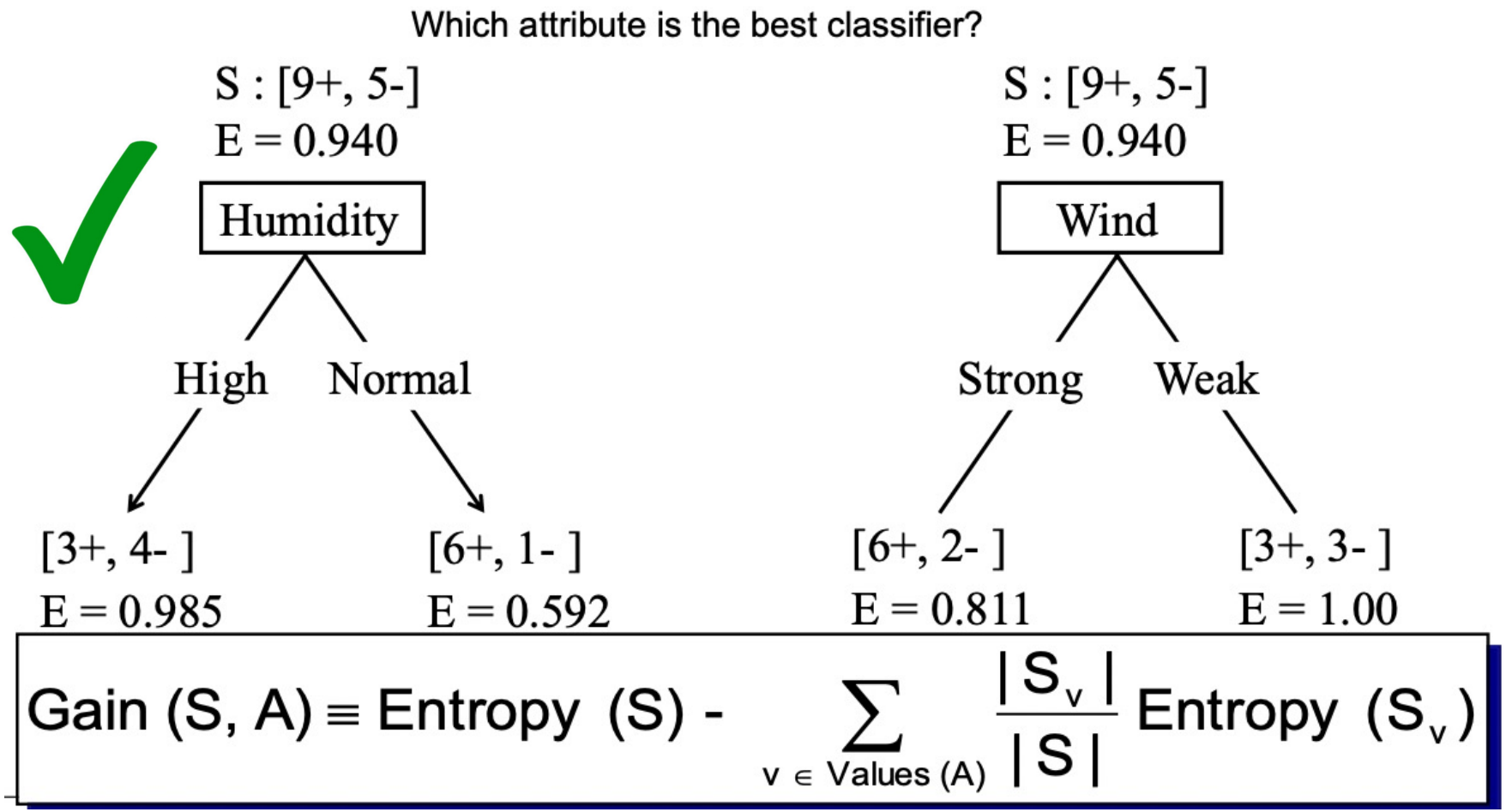
For more than two classes: $\text{Entropy (S)} = -\sum_{i=1}^{|C|} p(c_i) \log_2 p(c_i)$

Information Gain

- Gain (S, A) = expected reduction in entropy due to sorting on A

$$\text{Gain}(S, A) \equiv \text{Entropy}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

Selecting the Next Attribute



Example

$$entropy(D) = -\frac{6}{15} \times \log_2 \frac{6}{15} - \frac{9}{15} \times \log_2 \frac{9}{15} = 0.971$$

$$entropy_{Own\_house}(D) = -\frac{6}{15} \times entropy(D_1) - \frac{9}{15} \times entropy(D_2)$$
$$= \frac{6}{15} \times 0 + \frac{9}{15} \times 0.918$$
$$= 0.551$$

$$entropy_{Age}(D) = -\frac{5}{15} \times entropy(D_1) - \frac{5}{15} \times entropy(D_2) - \frac{5}{15} \times entropy(D_3)$$
$$= \frac{5}{15} \times 0.971 + \frac{5}{15} \times 0.971 + \frac{5}{15} \times 0.722$$
$$= 0.888$$

ID	Age	Has_Job	Own_House	Class
1	young	false	false	No
2	young	false	false	No
3	young	true	false	Yes
4	young	true	true	Yes
5	young	false	false	No
6	middle	false	false	No
7	middle	false	false	No
8	middle	true	true	Yes
9	middle	false	true	Yes
10	middle	false	true	Yes
11	old	false	true	Yes
12	old	false	true	Yes
13	old	true	false	Yes
14	old	true	false	Yes
15	old	false	false	No

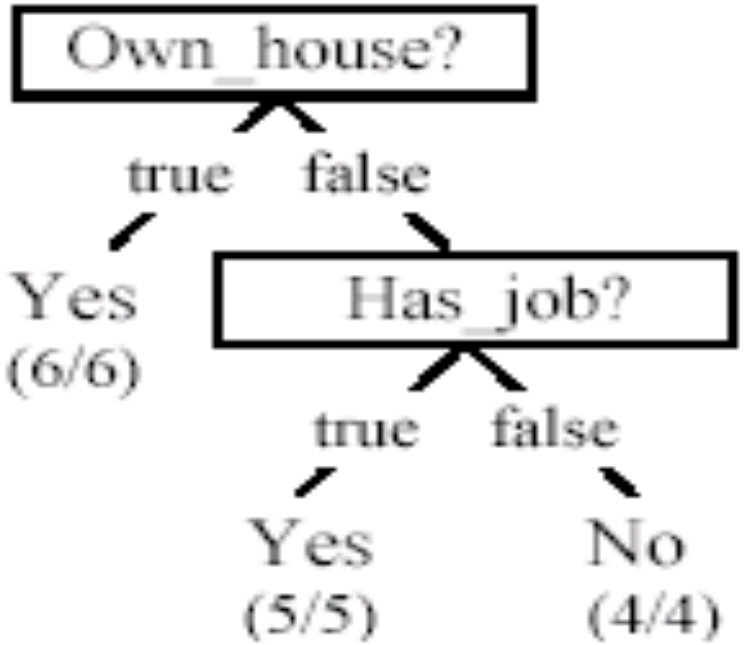
Age	Yes	No	entropy(Di)
young	2	3	0.971
middle	3	2	0.971
old	4	1	0.722

$gain(D, \text{Age}) = 0.971 - 0.888 = 0.083$
$gain(D, \text{Own\_house}) = 0.971 - 0.551 = 0.420$
$gain(D, \text{Has\_Job}) = 0.971 - 0.647 = 0.324$

Own_house is the best choice for the root.

Example (cont.)

- We build the final tree



Continuous Valued Attributes

Handling numeric data by:

- expert preprocessing
  - Use whenever there are natural boundary points in the value range of the continuous attribute, e.g. human body temperature 37 degrees C
- automatic preclustering
  - Use a clustering method to create groups of values for each attribute
  - Use whenever you would like the splits to be identical everywhere in the tree
- search at split time

Search at split time

- Sort attribute values
- Create m-1 binary splits from continuous attribute with m values
  - Common to place split values at or halfway between values
- Compare and select best split using entropy/information gain

Example:

Temperature:	40	48	60	72	80	90
PlayTennis:	No	No	Yes	Yes	Yes	No

- E.g. (Temperature =< 44), (Temperature =< 54), (Temperature =< 66), (Temperature =< 76), (Temperature =< 85)
- Or (Temperature =< 40), (Temperature =< 48), (Temperature =< 60), (Temperature =< 72), (Temperature =< 80)

Attributes with Many Values

Problem:

- If attribute has many values, Gain will select it
- Imagine using Data = Jun_3_1996 as attribute

One approach: use GainRatio instead!

Binary Decision Trees, Subsetting

- Most DT systems use only binary decision trees
- Need further split operator: subsetting
  - Subsetting: use value subsets for categorical attributes
  - For a categorical attribute with k possible value there are 2^k possible subsets. However some tests are redundant
- Test of type "Value of attribute A in set S"
  - Example: for the Outlook attribute, use the following tests

    Outlook ∈ {Sunny}

    Outlook ∈ {Overcast}

    Outlook ∈ {Rain}
  - What about Outlook ∈ {Sunny, Overcast}?

    Outlook ∈ {Sunny, Rain}? Outlook ∈ {Overcast, Rain}?

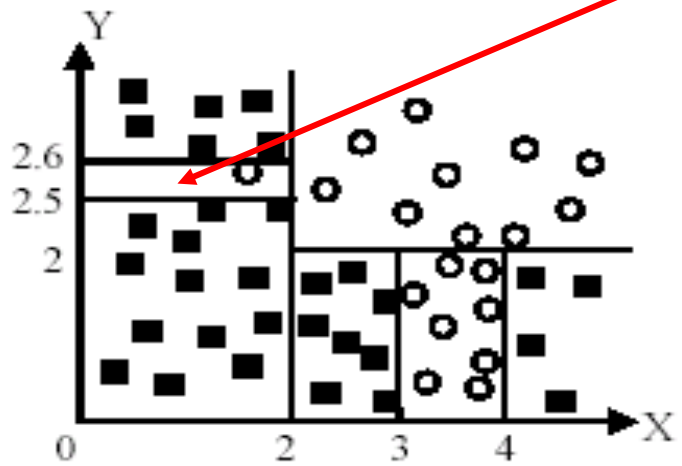
Outlook	Wind	Play Tennis
Sunny	Strong	No
Overcast	Weak	Yes
Sunny	Weak	Yes
Rain	Strong	Yes

Overfitting in Decision Trees

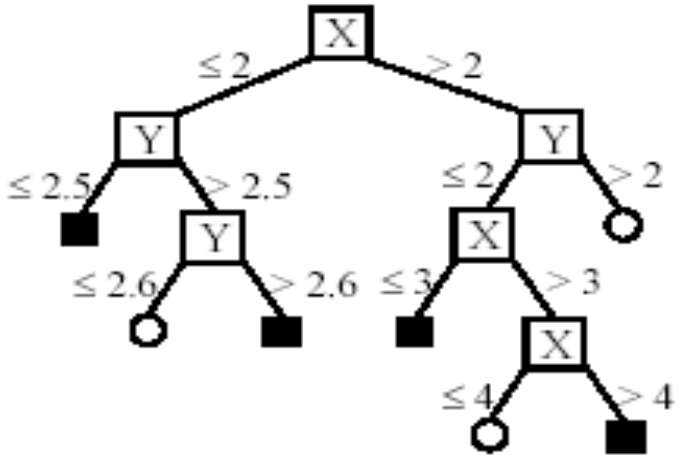
- Ideal goal of classification: Find the simplest decision tree that fits the data and generalizes to unseen data
- Overfitting: A tree may overfit the training data
- Good accuracy on training data but poor on test data
  - Symptoms: tree too deep and too many branches, some may reflect anomalies due to noise or outliers
  - Overfitting results in decision trees that are more complex than necessary
  - Trade-off full consistency for compactness
    - Larger decision trees can be more consistent
    - Smaller decision trees generalize better
- Causes of overfiting
  - Overfitting Due to Presence of Noise.

Example

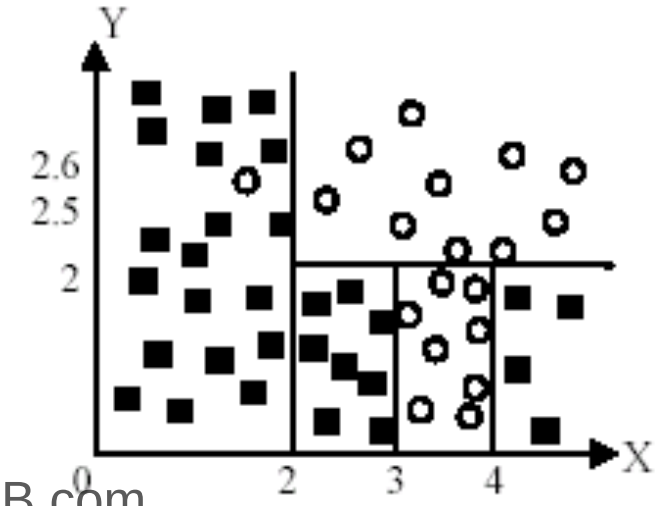
Likely to overfit the data

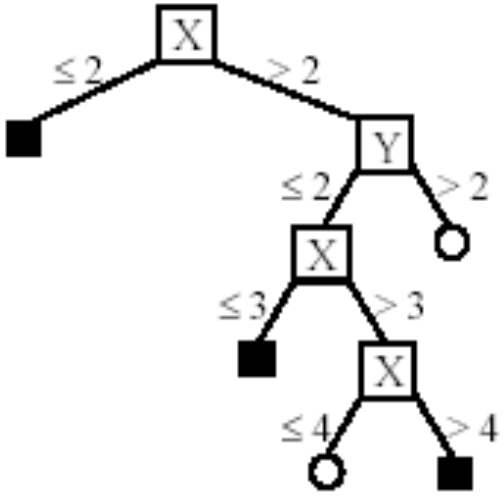


(A) A partition of the data space

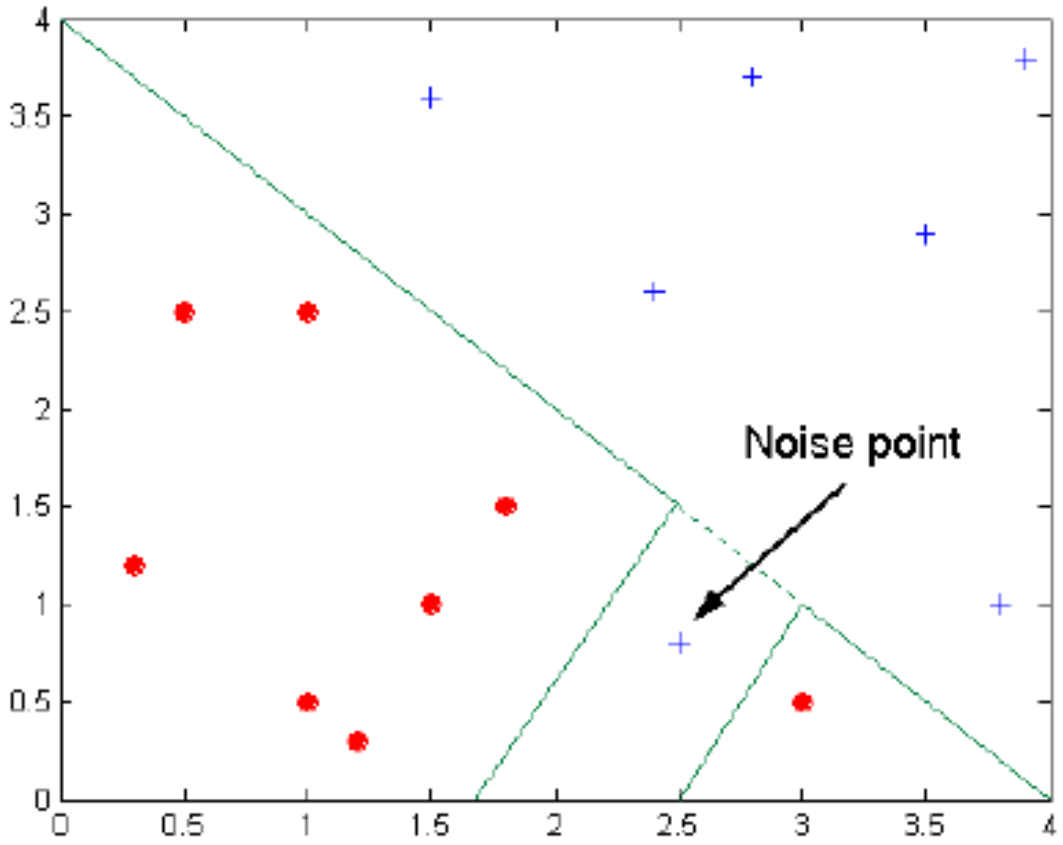


(B). The decision tree





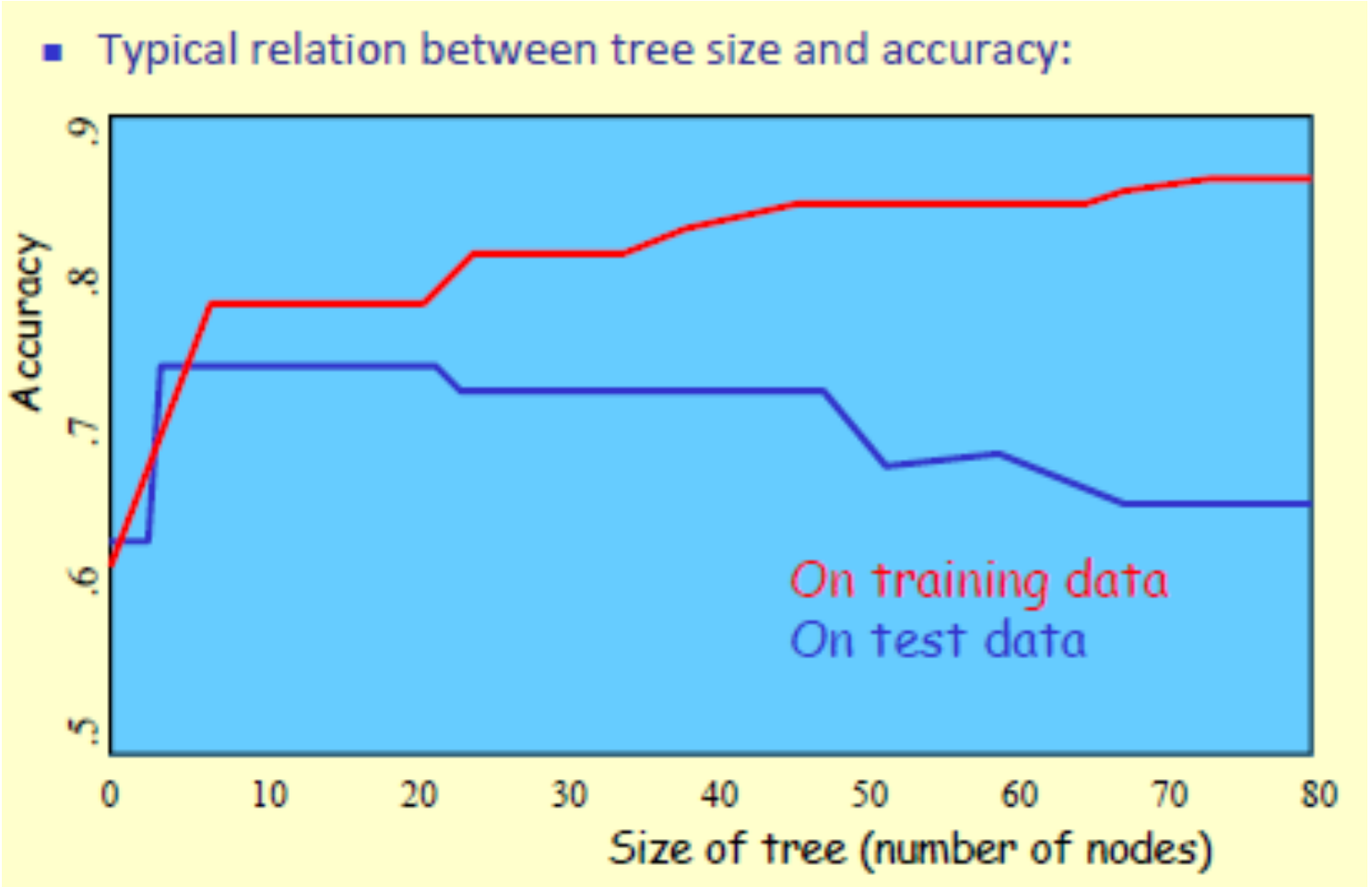
Overfitting due to Noise



Decision boundary is distorted by noise point

Overfitting and accuracy

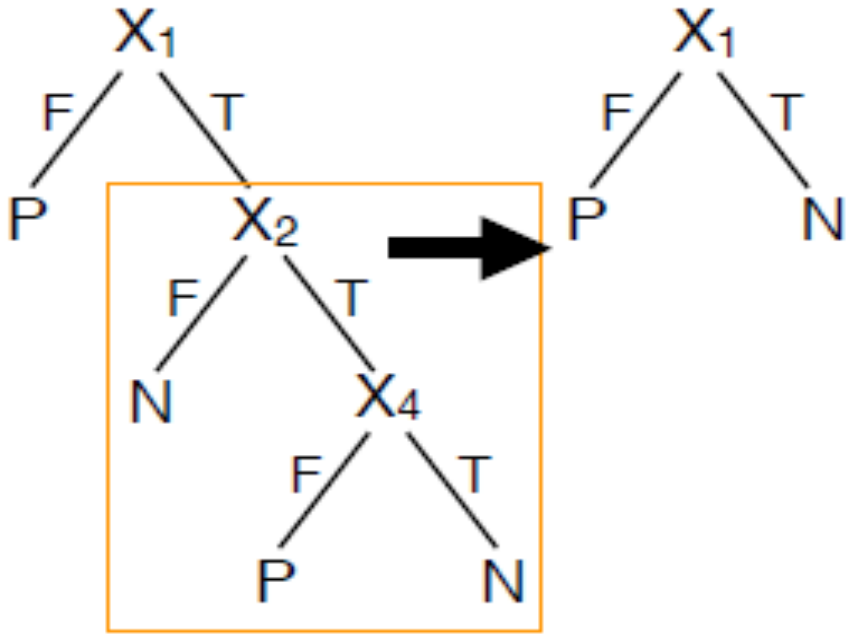
- Typical relation between tree size and accuracy:



Pruning to avoid overfitting

- Remove a subtree
- Replace with a leaf
- Class of leaf is most common class of data in subtree
- Example

X_1	X_2	X_3	X_4	C
F	F	F	F	P
F	F	T	T	P
F	T	F	T	P
T	T	T	F	P
T	F	F	F	N
T	T	T	T	N
T	T	T	F	N



Decision Tree Pruning Methodologies

- Pre-pruning (top-down)
  - Stopping criteria while growing the tree
  - Ex:
    - Stop if the tree depth exceeds a predefined maximum depth value
    - Stop if number of instances is less than some user-specified threshold
    - Stop if expanding the current node does not improve impurity measures

- Post-pruning (bottom-up)
  - Grow the tree, then prune
  - If generalization error improves after trimming, replace sub-tree by a leaf node
  - Class label of leaf node is determined from majority class of instances in the sub-tree

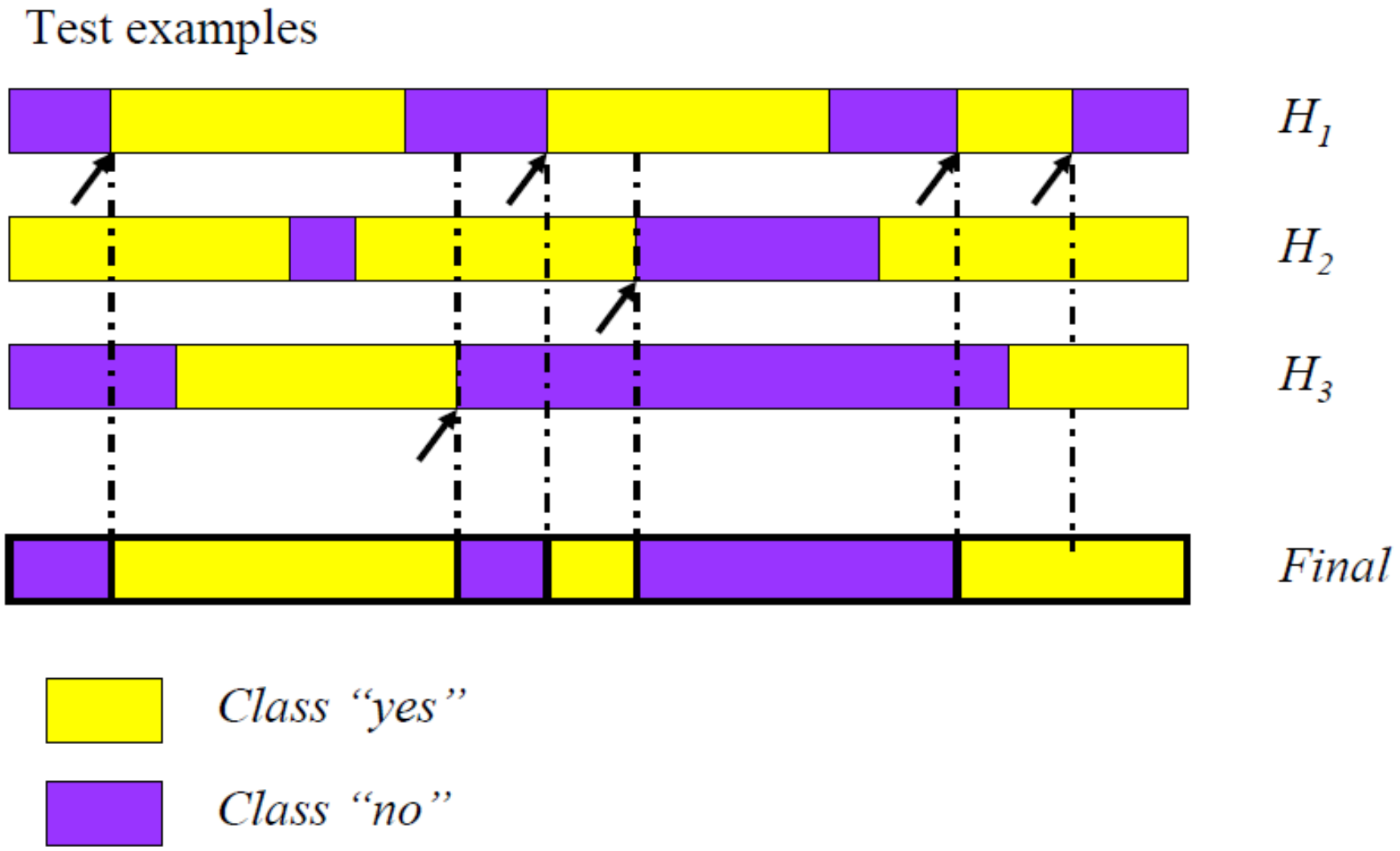
Advantages/Disadvantages of Decision Trees

- Advantages:
  - Easy to understand and implement
  - Interpretability
  - Easy to generate rules

- Disadvantages:
  - May suffer from overfitting.
  - Classifies by rectangular partitioning (so does not handle correlated features very well).
  - Can be quite large – pruning is necessary.
  - Does not handle streaming data easily

Back to Ensembles

Simple Majority Voting



Random Forests

- Main idea: build a larger number of un-pruned decision trees and combine their predictions.
- The trees has to be different from each others. This is achieved by using two sources of randomness:
  - Bagging: randomizing the training set.
  - Randomized node optimization (RNO): randomizing the set of features to select from in each node.
- Final prediction is obtained by either majority voting, or averaging the predictions of all trees.

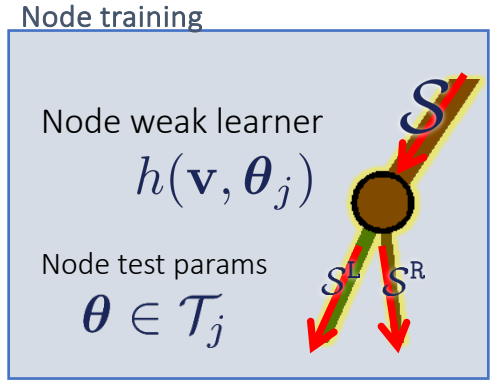
Sources of randomness in decision forest

- Bagging

$\mathcal{S}_0$	The full training set
$\mathcal{S}_0^t \subset \mathcal{S}_0$	The randomly sampled subset of training data made available for the tree *t*

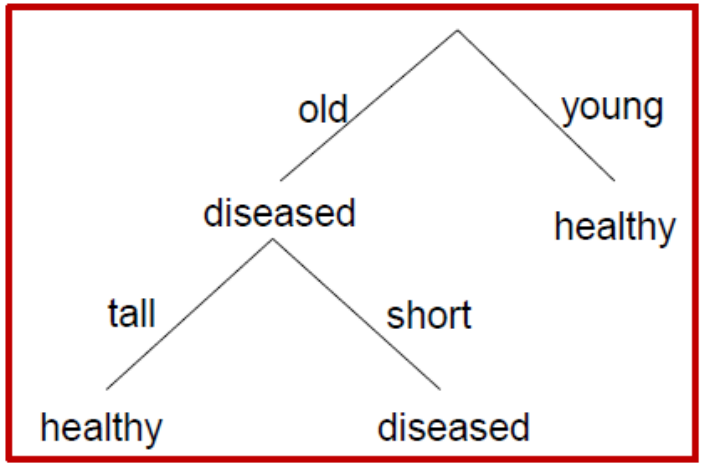
- Randomized node optimization (RNO)

$\mathcal{T}$	The full set of all possible node test parameters
$\mathcal{T}_j \subset \mathcal{T}$	For each node the set of randomly sampled features
$\rho = \lvert\mathcal{T}_j\rvert$	Randomness control parameter. For $\rho = \lvert\mathcal{T}\rvert$ no randomness and maximum tree correlation. For $\rho = 1$ max randomness and minimum tree correlation.

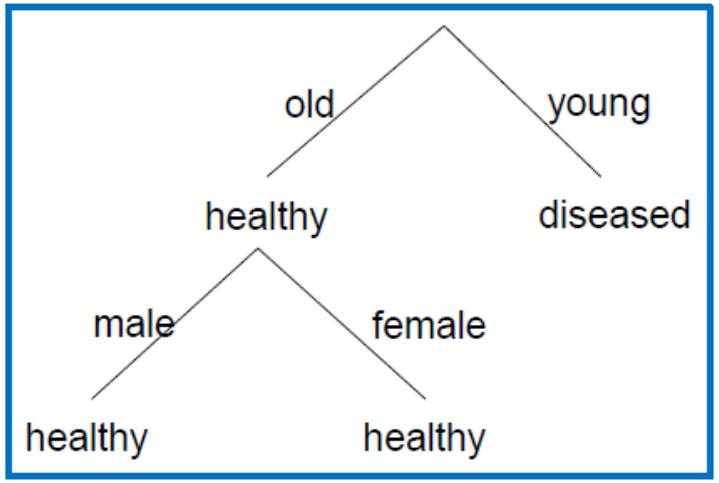


Random Forests - Example

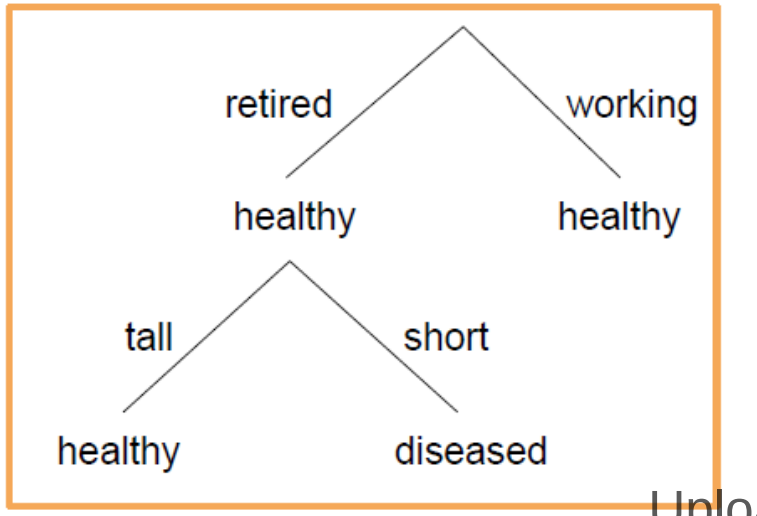
Tree 1



Tree 2



Tree 3



New sample:

old, retired, male, short

Tree predictions:

diseased, healthy, diseased

Majority rule:

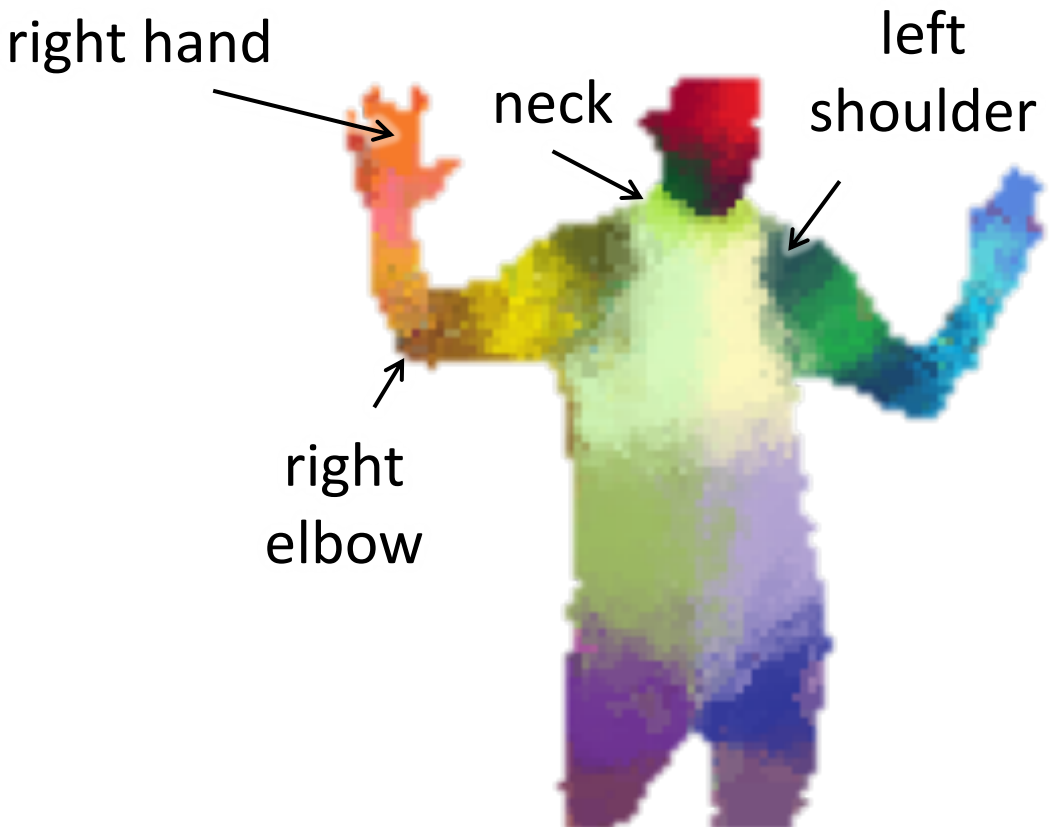
diseased

Random forest algorithm

- Each tree is constructed using the following algorithm:
  - Let $\mathcal{S}_0$ be the set of all training examples with size N, and $\mathcal{T}$ the set of all features/variables in the dataset.
  - Choose a training set for this tree by choosing a bootstrap sample $\mathcal{S}_0^t \subset \mathcal{S}_0$ of size n from all N available training cases.
  - For each node of the tree, randomly choose a subset of features $\mathcal{T}_j \subset \mathcal{T}$ of size ρ on which to base the decision at that node. Calculate the best split based on these ρ features in the training set.
  - Each tree is fully grown and not pruned.
- For prediction, a new sample is pushed down the tree. It is assigned the label of the leaf node it ends up in. This procedure is iterated over all trees in the ensemble, and the average vote of all trees is reported as random forest prediction.

Random Forest Application

- Body part classification

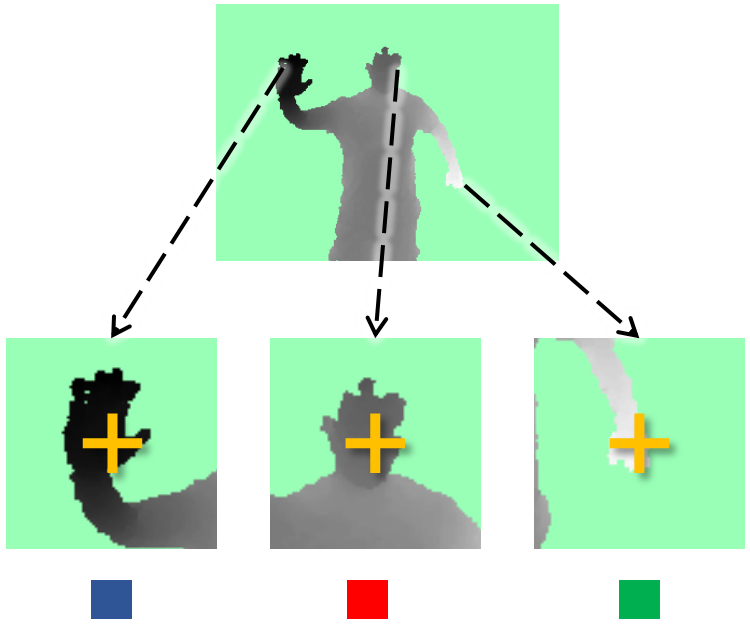


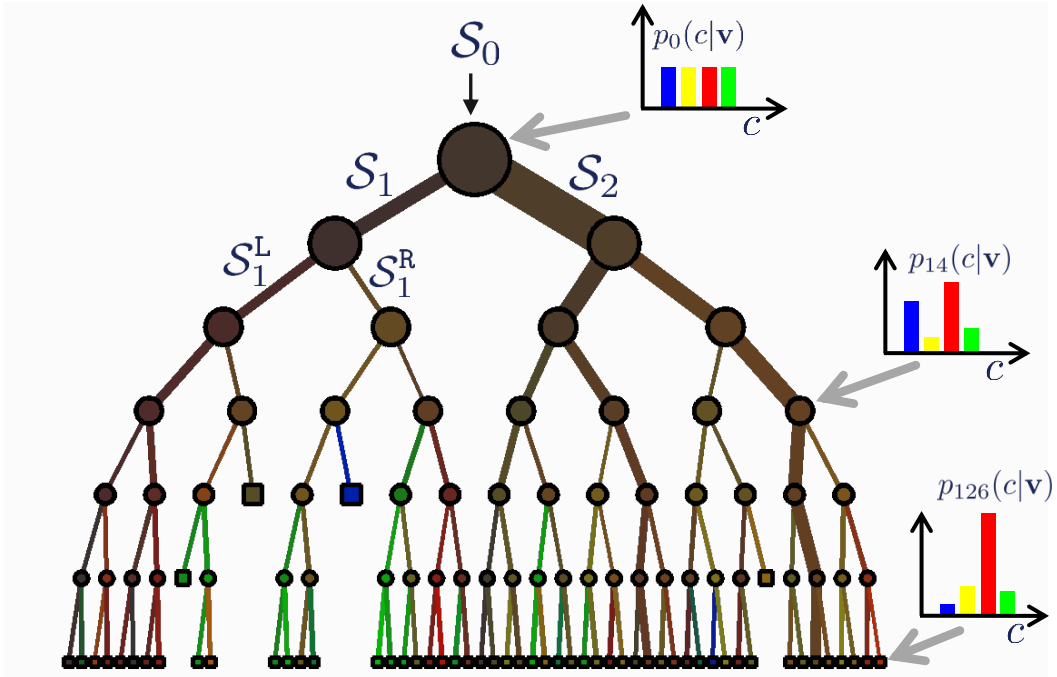
[J. Shotton et al. **Real-Time Human Pose Recognition in Parts from a Single Depth Image**. CVPR 2011]

Body part classification

- Input: Labeled training data (depth images of human body)

- Tree: Separate data based on class label





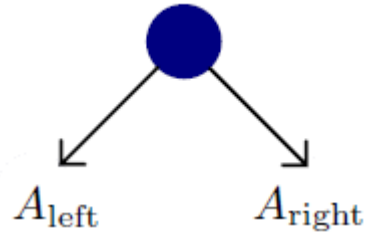
Training Data

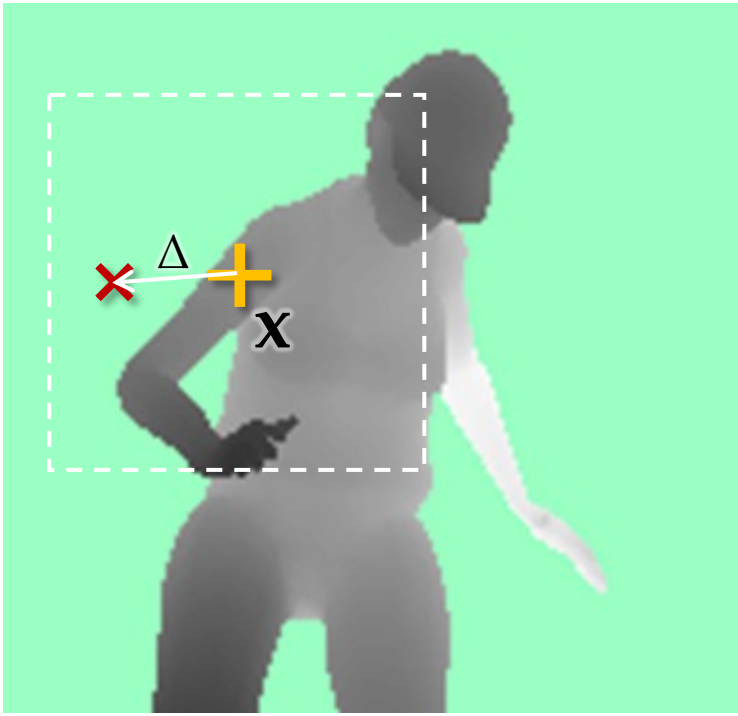
synthetic

real

Training

- At each node, perform a binary test





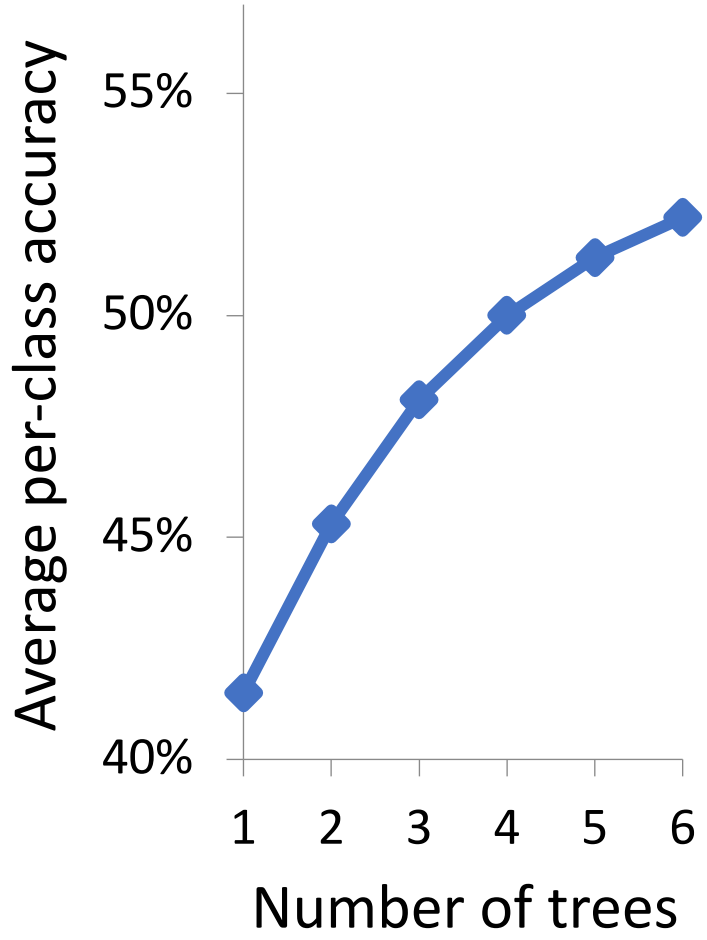
- A_{left} : all points with a difference of depth value between that point and another point with offset Δ is less than a threshold τ

- A_{right} : all points with a difference of depth value between that point and another point with offset Δ is greater than a threshold τ

$$f_\phi(\mathcal{P}(\boldsymbol{x})) = \begin{cases} 0 & \text{if } d(\boldsymbol{x}) - d\left(\boldsymbol{x} + \frac{\boldsymbol{v}}{d(\boldsymbol{x})}\right) < \tau \\ 1 & \text{otherwise.} \end{cases}$$

$$\phi = (\boldsymbol{v}, \tau)$$

Testing



ground truth

1 tree

3 trees

6 trees