Decision Tree Learning

Dr. Xiaowei Huang

https://cgi.csc.liv.ac.uk/~xiaowei/

Decision Tree up to now,

- Decision tree representation
- A general top-down algorithm
- How to do splitting on numeric features

- Occam’s razor
- entropy and information gain
- types of decision-tree splits

Topics

- Accuracy of decision trees
- Overfitting
- Stopping criteria of decision trees
- Variants of decision trees
    - Regression trees
    - probability estimation trees
    - m-of-n splits
    - lookahead

Accuracy of Decision Tree

Definition of Accuracy and Error

- Given a set D of samples and a trained model M, the accuracy is the percentage of correctly labeled samples. That is,

$$Accuracy(D, M) = \frac{|\{M(x) = l_x \mid x \in D\}|}{|D|}$$

Where l_x is the true label of sample x and M(x) gives the predicted label of x by M.

- Error is a dual concept of accuracy.

But, what is D?

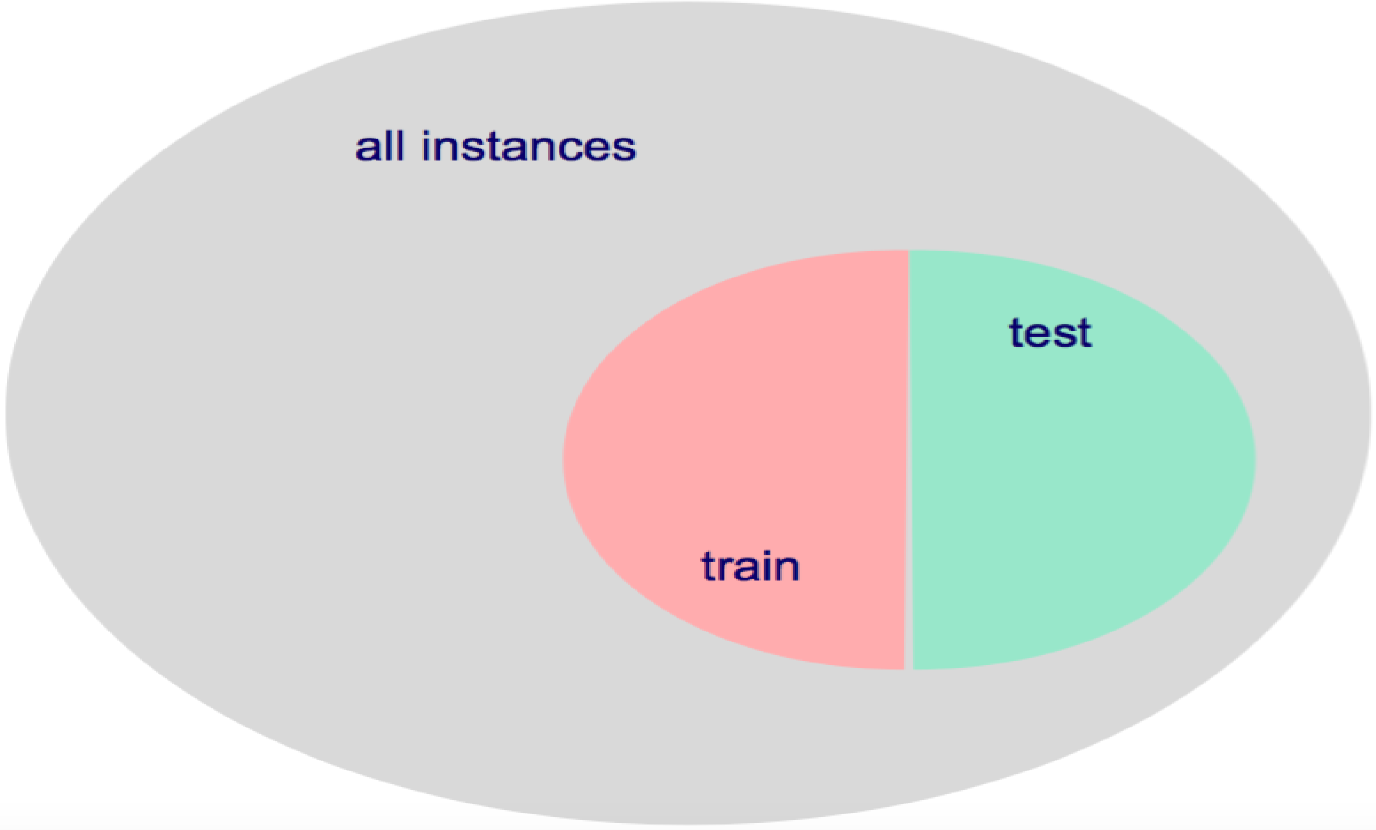
$$Error(D, M) = 1 - Accuracy(D, M)$$

How can we assess the accuracy of a tree?

- Can we just calculate the fraction of training instances that are correctly classified?
- Consider a problem domain in which instances are assigned labels at random with P(Y = t) = 0.5
  - how accurate would a learned decision tree be on previously unseen instances?
  - how accurate would it be on its training set?

How can we assess the accuracy of a tree?

- to get an unbiased estimate of a learned model's accuracy, we must use a set of instances that are held-aside during learning
- this is called a *test set*



Overfitting

Overfitting

- consider error of model M over
  - training data: $Error(D_{training}, M)$
  - entire distribution of data: $Error(D_{true}, M)$
- model $M \in H$ *overfits* the training data if there is an alternative model $M' \in H$ such that

$$Error(D_{training}, M) < Error(D_{training}, M')$$

Perform better on training dataset

$$Error(D_{true}, M) > Error(D_{true}, M')$$

Perform worse on ground truth data

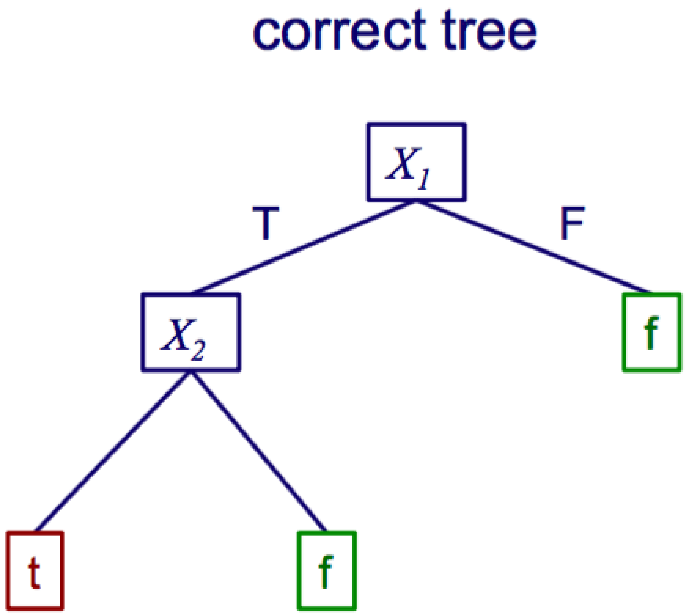
Example 1: overfitting with noisy data

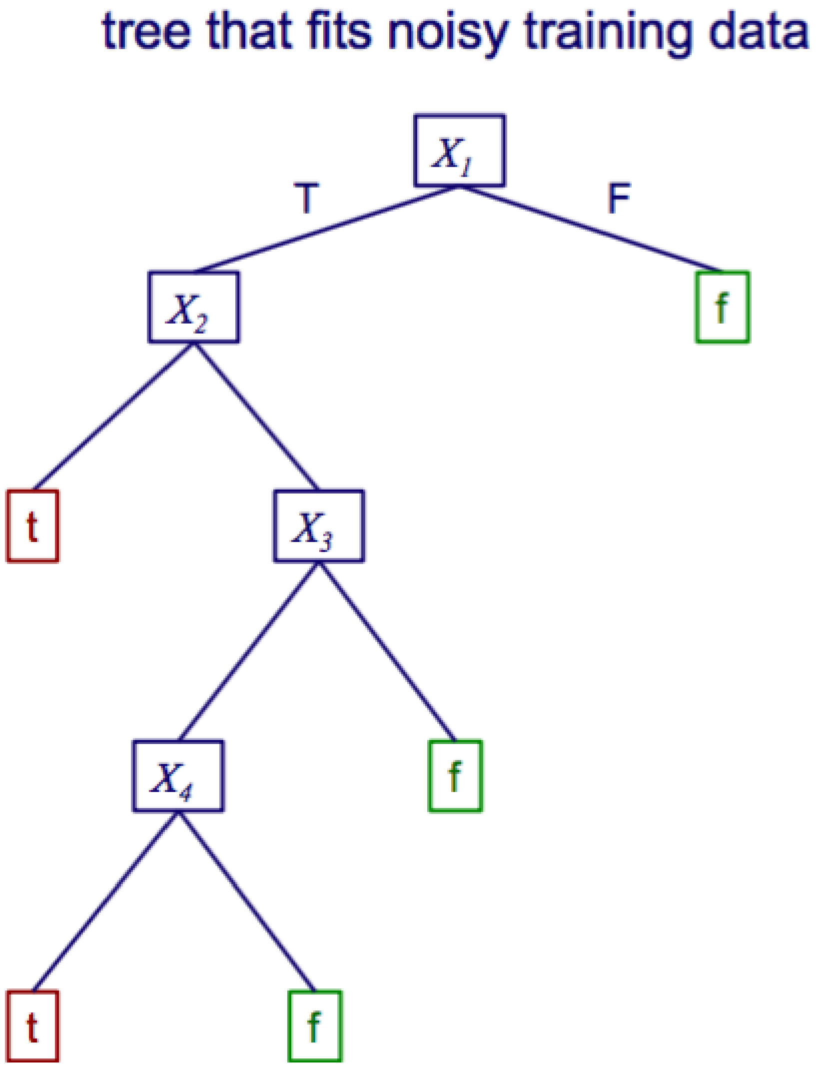
- suppose
  - the target concept is $Y = X_1 \wedge X_2$
  - there is noise in some feature values
  - we're given the following training set

X_1	X_2	X_3	X_4	X_5	…	Y
t	t	t	t	t	…	t
t	t	f	f	t	…	t
t	f	t	t	f	…	t
t	f	f	t	f	…	f
t	f	t	f	f	…	f
f	t	t	f	t	…	f

noisy value

Example 1: overfitting with noisy data





$$Y = X_1 \wedge X_2$$

A noisy data:

X_1 = t

X_2 = f

X_3 = t

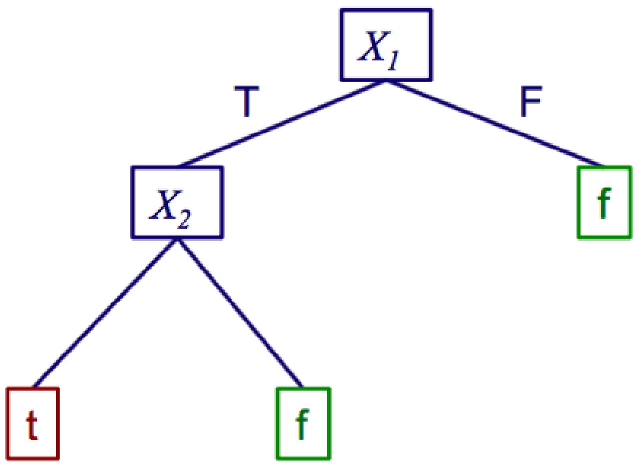
X_4 = t

X_5 = f

Y = t

Example 1: overfitting with noisy data

correct tree



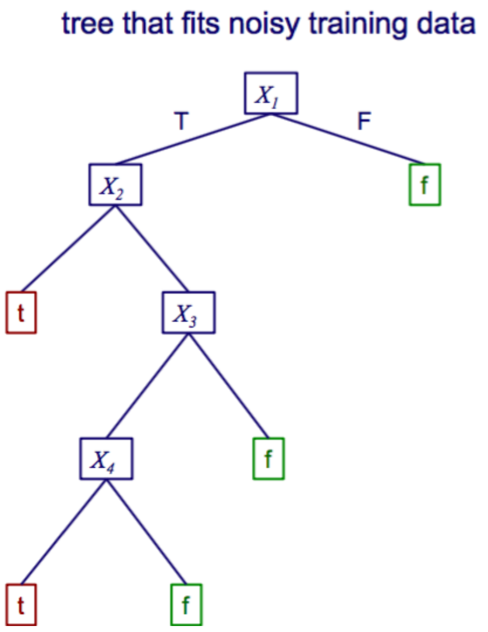
$$Y = X_1 \wedge X_2$$

X_1	X_2	X_3	X_4	X_5	...	Y
t	t	t	t	t	...	t
t	t	f	f	t	...	t
t	f	t	t	f	...	t
t	f	f	t	f	...	f
t	f	t	f	f	...	f
f	t	t	f	t	...	f

noisy value

- What is the accuracy?
  - Accuracy($D_{training}$,M) = 5/6
  - Accuracy(D_{true},M) = 100%

Example 1: overfitting with noisy data



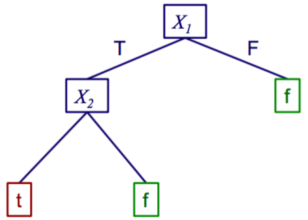
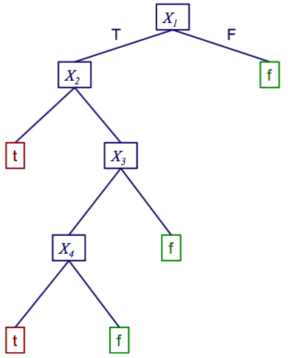
$$Y = X_1 \wedge X_2$$

X_1	X_2	X_3	X_4	X_5	...	Y
t	t	t	t	t	...	t
t	t	f	f	t	...	t
t	f	t	t	f	...	t
t	f	f	t	f	...	f
t	f	t	f	f	...	f
f	t	t	f	t	...	f

noisy value

- What is the accuracy?
  - Accuracy($D_{training}$,M) = 100%
  - Accuracy(D_{true},M) < 100%

Example 1: overfitting with noisy data

		Training set accuracy	True accuracy
M_1	correct tree 	5/6	100%
M_2	tree that fits noisy training data 	100%	< 100 %

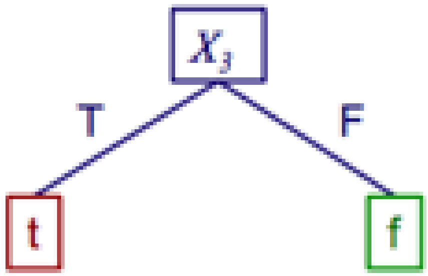
M_2 is overfitting!

Example 2: overfitting with noise-free data

- suppose
  - the target concept is $Y = X_1 \wedge X_2$
  - $P(X_3 = t) = 0.5$ for both classes
  - $P(Y = t) = 0.66$
  - we're given the following training set

X_1	X_2	X_3	X_4	X_5	...	Y
t	t	t	t	t	...	t
t	t	t	f	t	...	t
t	t	t	t	f	...	t
t	f	f	t	f	...	f
f	t	f	f	t	...	f

Example 2: overfitting with noise-free data



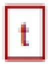
$$Y = X_1 \wedge X_2$$

$P(X_3 = t) = 0.5$
$P(Y{=}t) = 0.66$

- What is the accuracy?
  - Accuracy($D_{training}$,M) = 100%
  - Accuracy(D_{true},M) = 50%

X_1	X_2	X_3	X_4	X_5	...	Y
t	t	t	t	t	...	t
t	t	t	f	t	...	t
t	t	t	t	f	...	t
t	f	f	t	f	...	f
f	t	f	f	t	...	f

Example 2: overfitting with noise-free data



$$Y = X_1 \wedge X_2$$

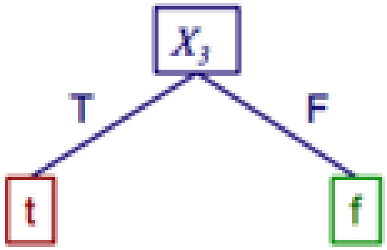
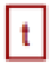
$P(X_3 = t) = 0.5$

$P(Y=t) = 0.66$

- What is the accuracy?
  - Accuracy($D_{training}$,M) = 60%
  - Accuracy(D_{true},M) = 66%

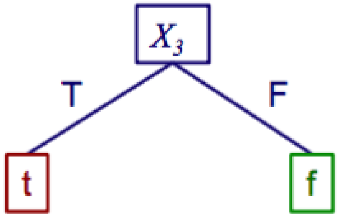
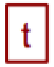
X_1	X_2	X_3	X_4	X_5	...	Y
t	t	t	t	t	...	t
t	t	t	f	t	...	t
t	t	t	t	f	...	t
t	f	f	t	f	...	f
f	t	f	f	t	...	f

Example 2: overfitting with noise-free data

		Training set accuracy	True accuracy	
M_1		100%	50%	M_1 is overfitting!
M_2		60%	66%	

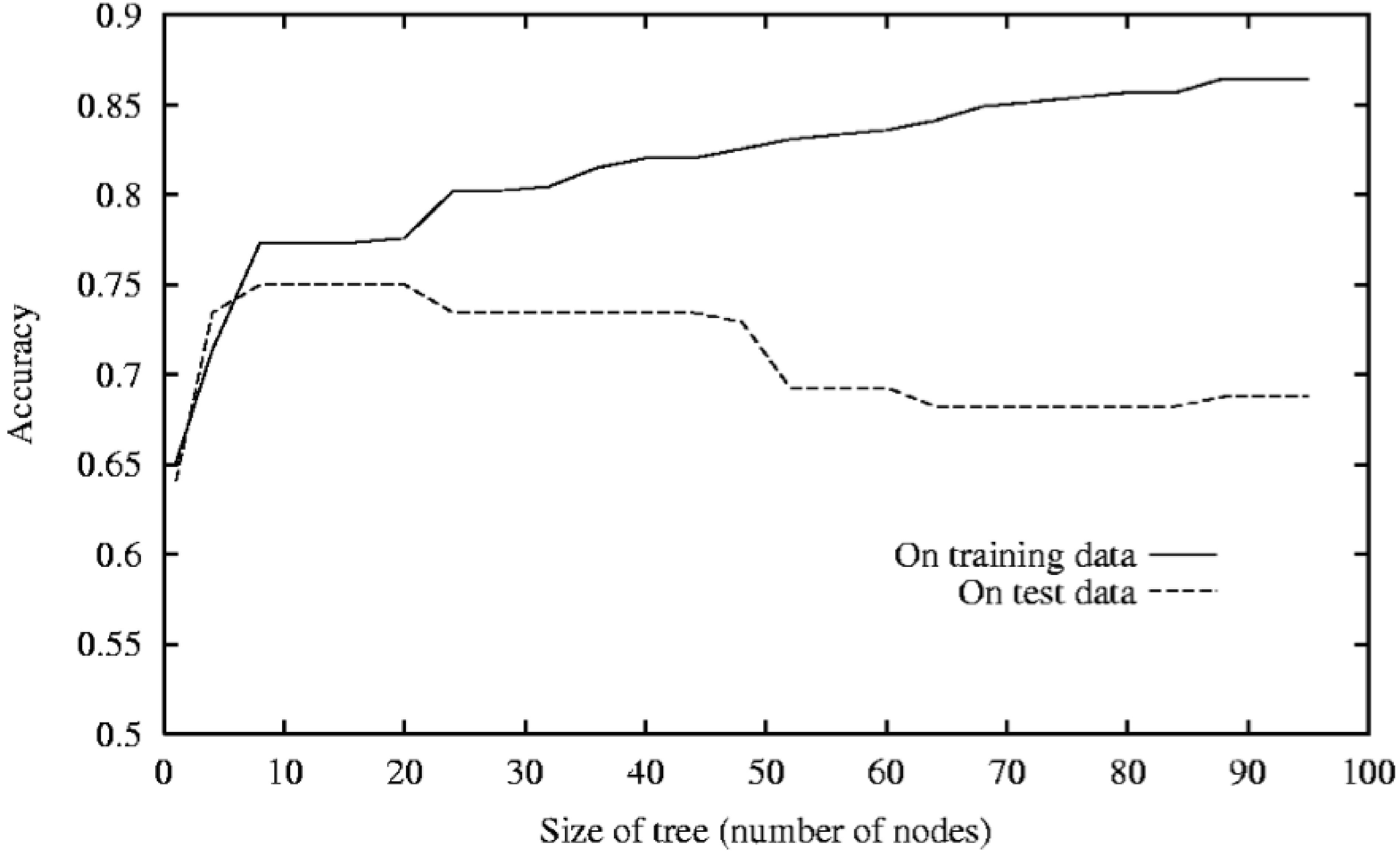
- because the training set is a limited sample, there might be (combinations of) features that are correlated with the target concept by chance

Example 2: overfitting with noise-free data

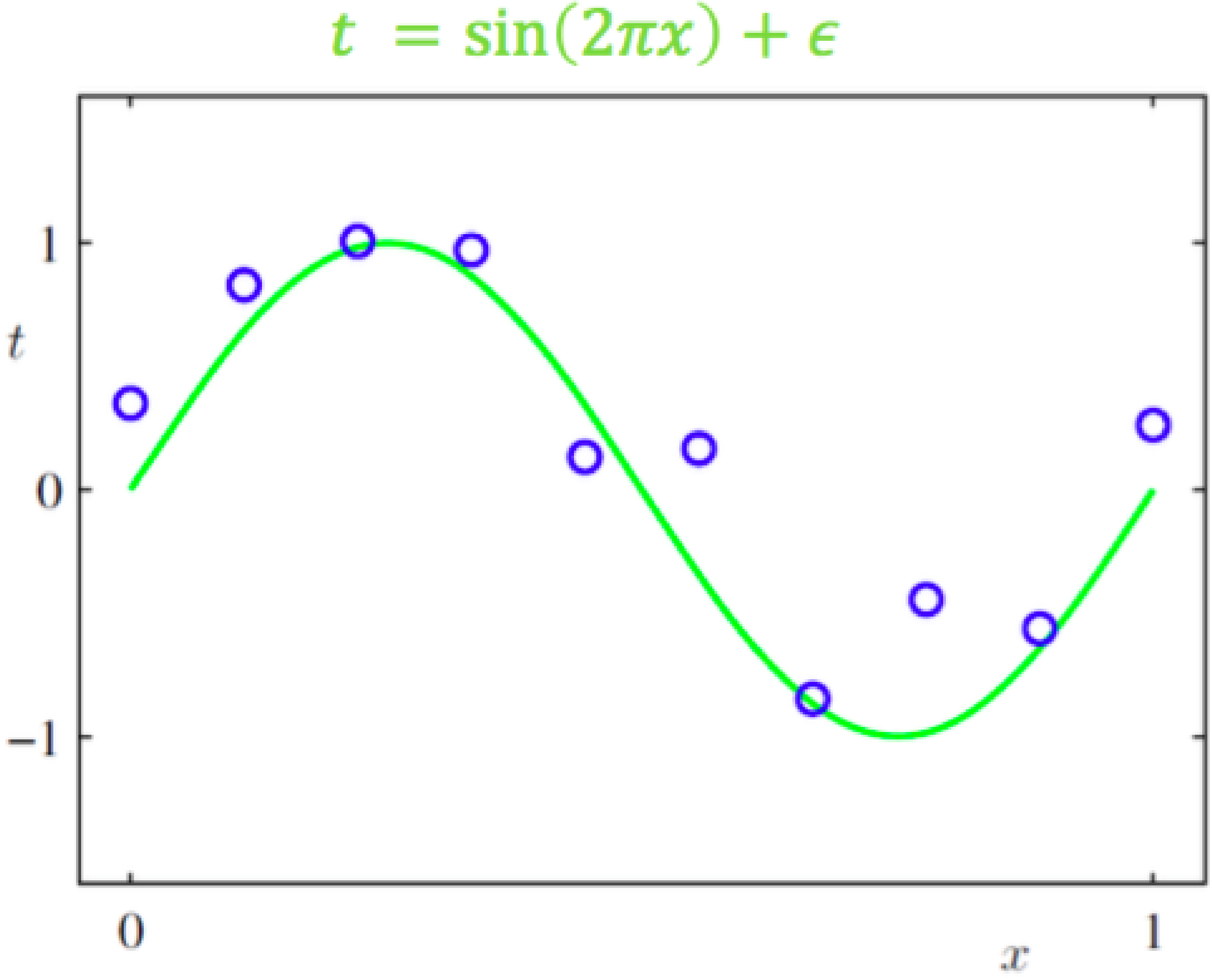
	training set accuracy	test set accuracy
	100%	50%
	66%	66%

- because the training set is a limited sample, there might be (combinations of) features that are correlated with the target concept by chance

Overfitting in decision trees

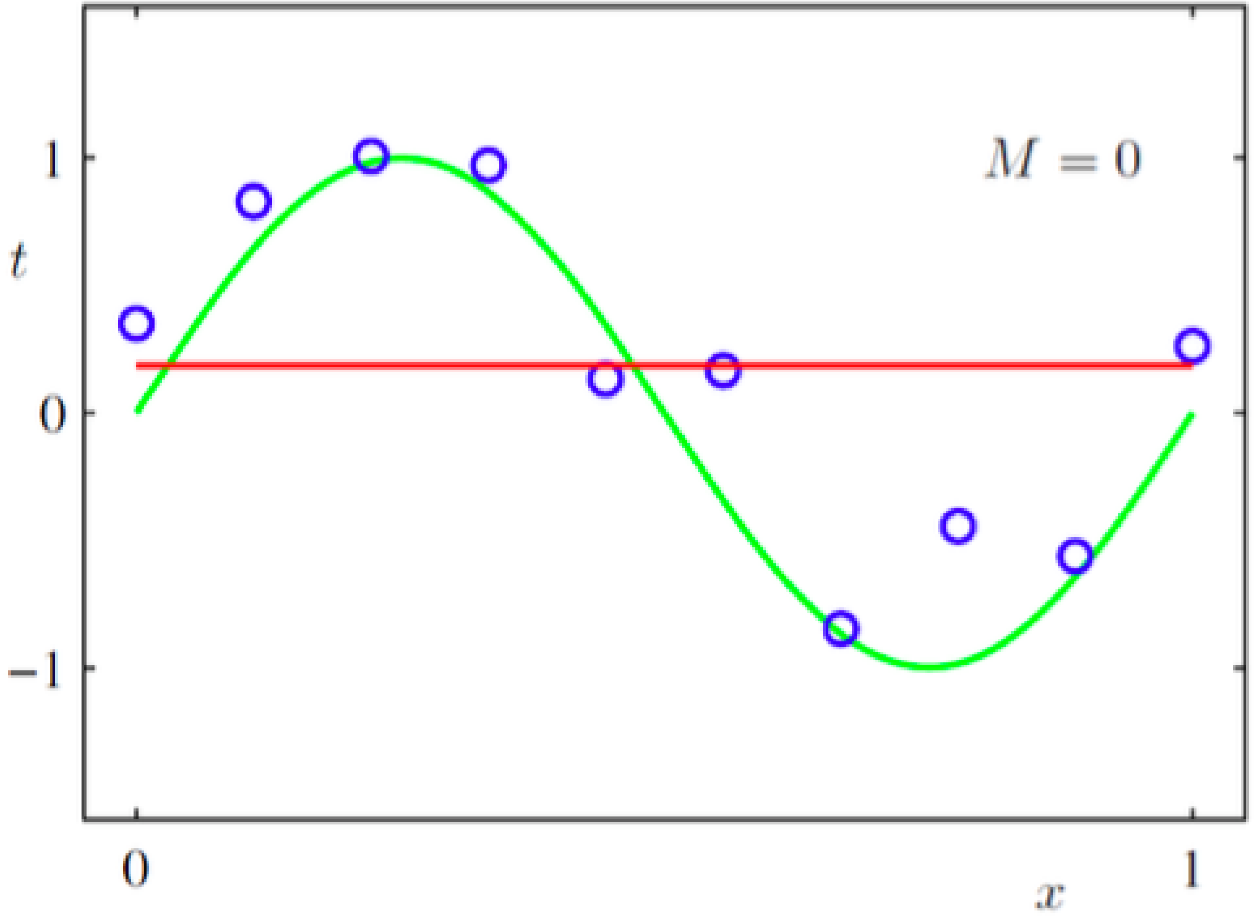


Example 3: regression using polynomial



Example 3: regression using polynomial

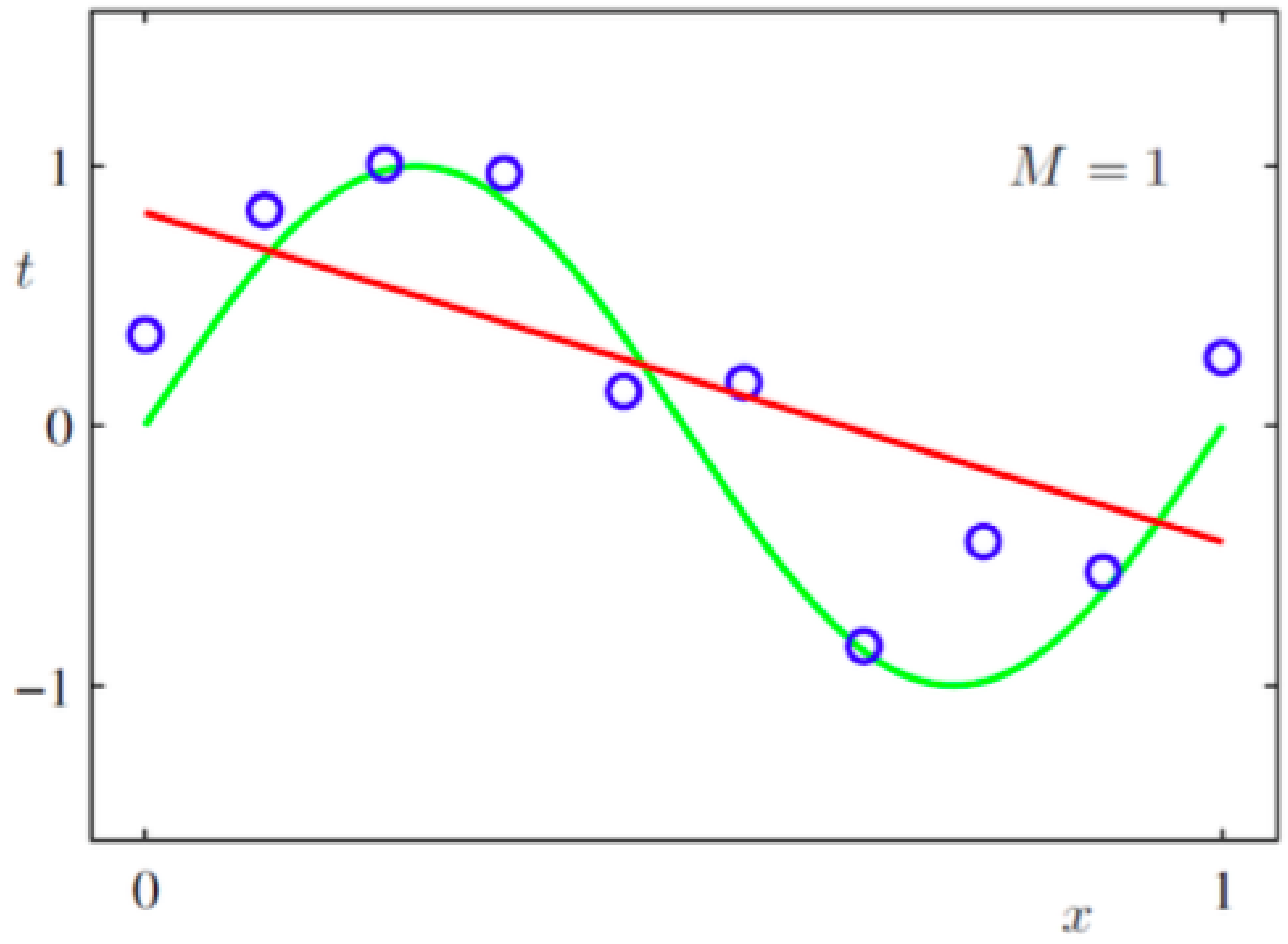
$$t = \sin(2\pi x) + \epsilon$$



Regression using polynomial of degree M

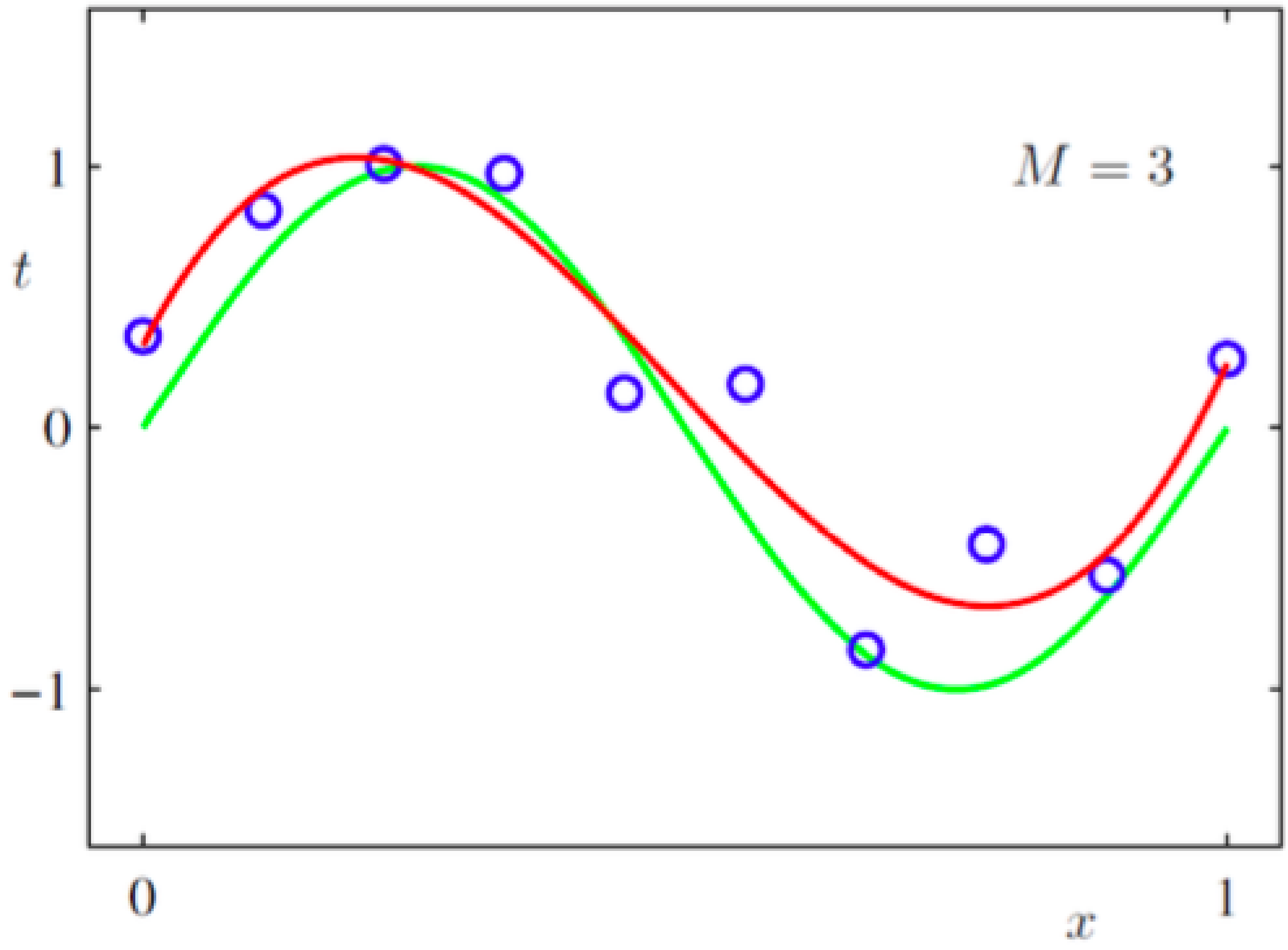
Example 3: regression using polynomial

$$t = \sin(2\pi x) + \epsilon$$



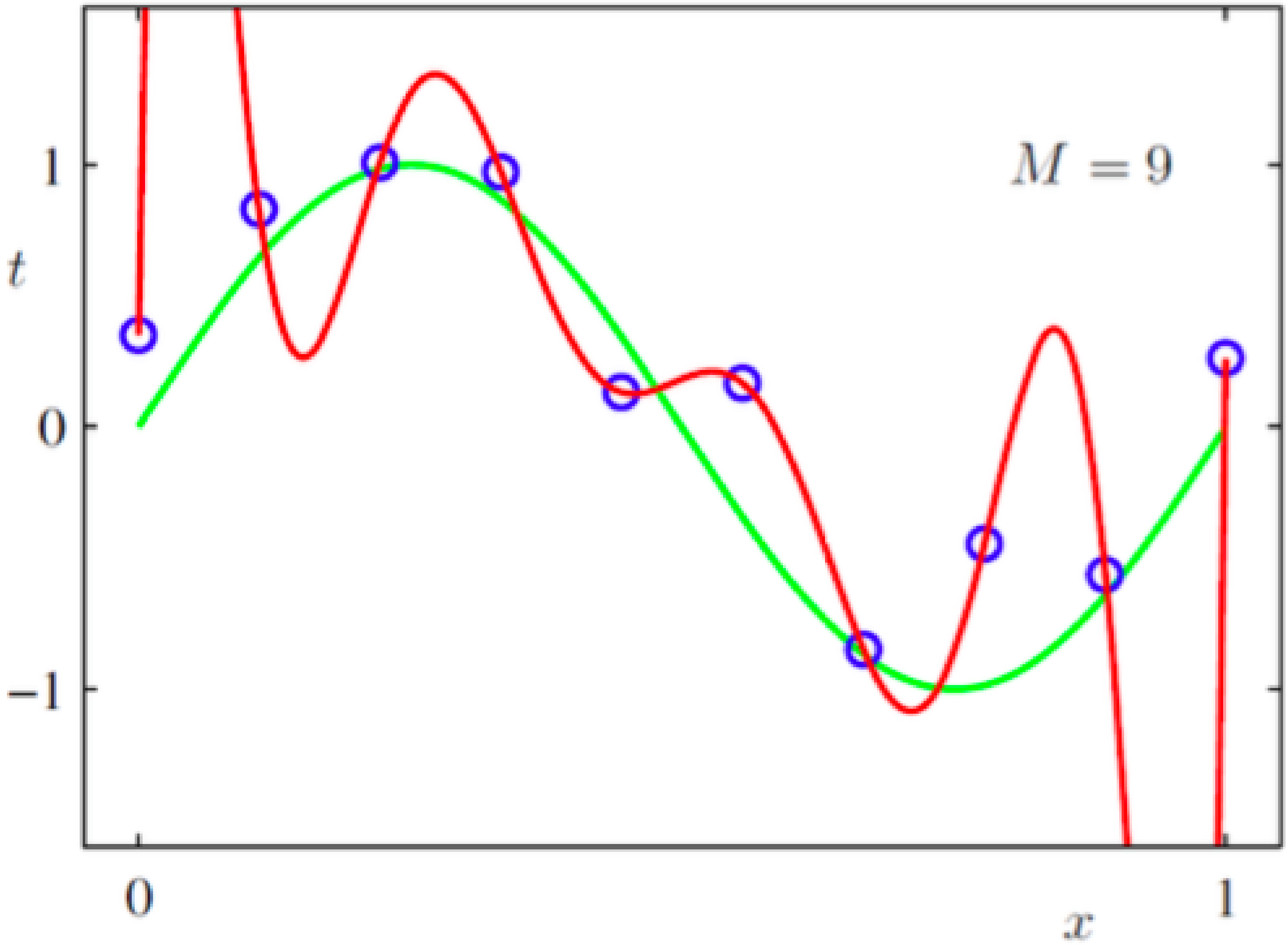
Example 3: regression using polynomial

$$t = \sin(2\pi x) + \epsilon$$

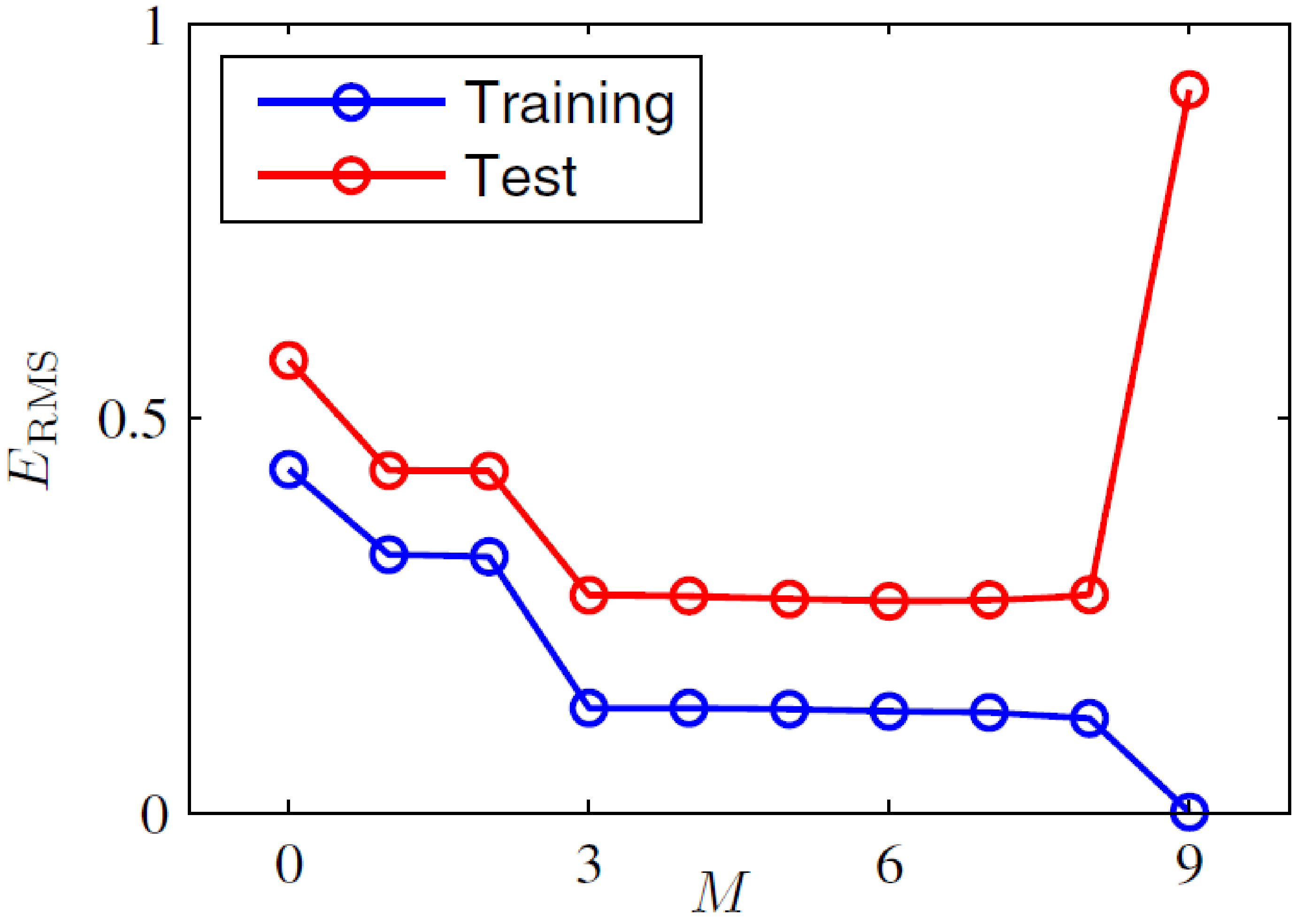


Example 3: regression using polynomial

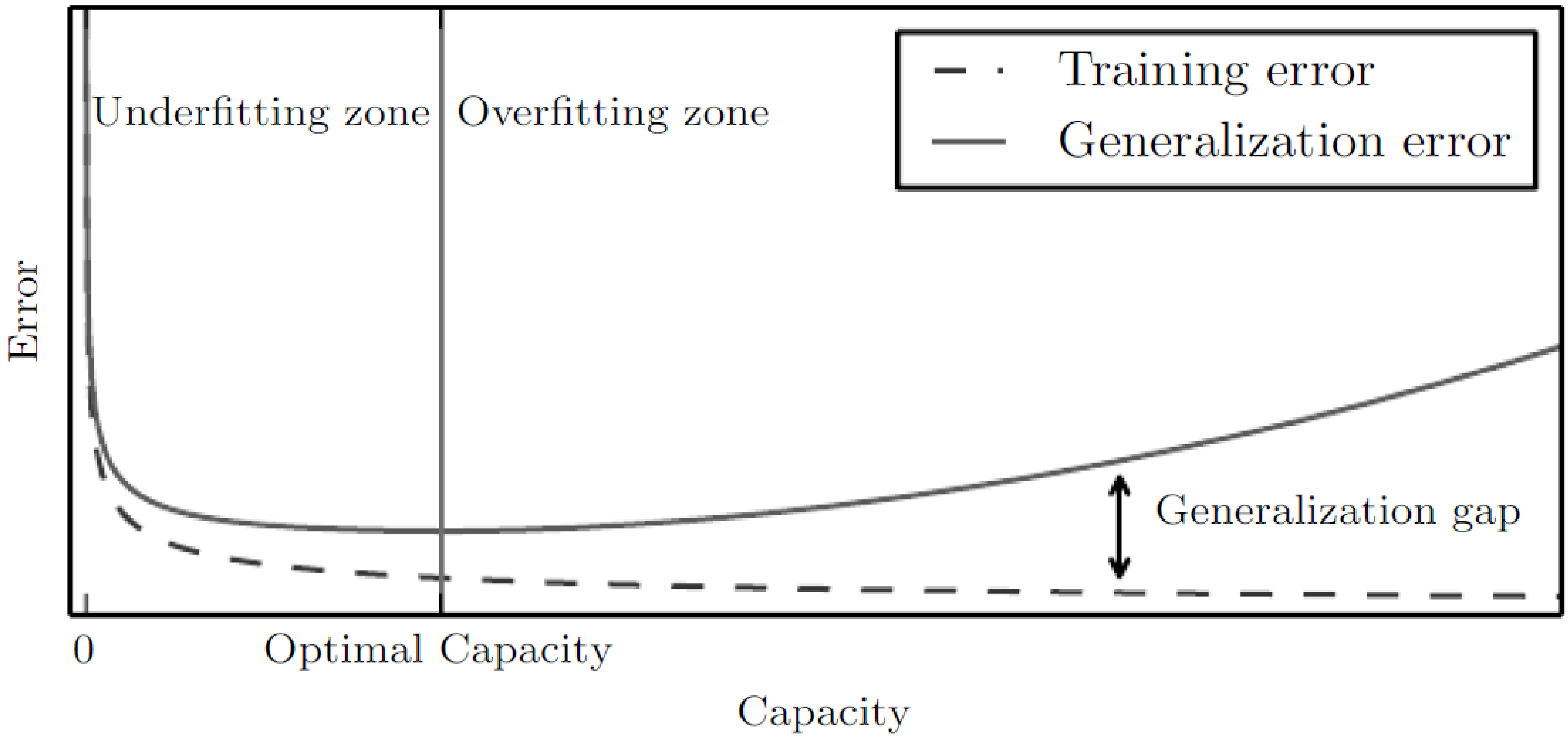
$$t = \sin(2\pi x) + \epsilon$$



Example 3: regression using polynomial



General phenomenon



Prevent overfitting

- cause: training error and expected error are different
  - there may be noise in the training data
  - training data is of limited size, resulting in difference from the true distribution
  - larger the hypothesis class, easier to find a hypothesis that fits the difference between the training data and the true distribution
- prevent overfitting:
  - cleaner training data help!
  - more training data help!
  - throwing away unnecessary hypotheses helps! (Occam's Razor)

Avoiding overfitting in DT learning

- two general strategies to avoid overfitting
  - *1. early stopping*: stop if further splitting not justified by a statistical test
    - Quinlan's original approach in ID3
  - *2. post-pruning*: grow a large tree, then prune back some nodes
    - more robust to myopia of greedy tree learning

Stopping criteria

Stopping criteria

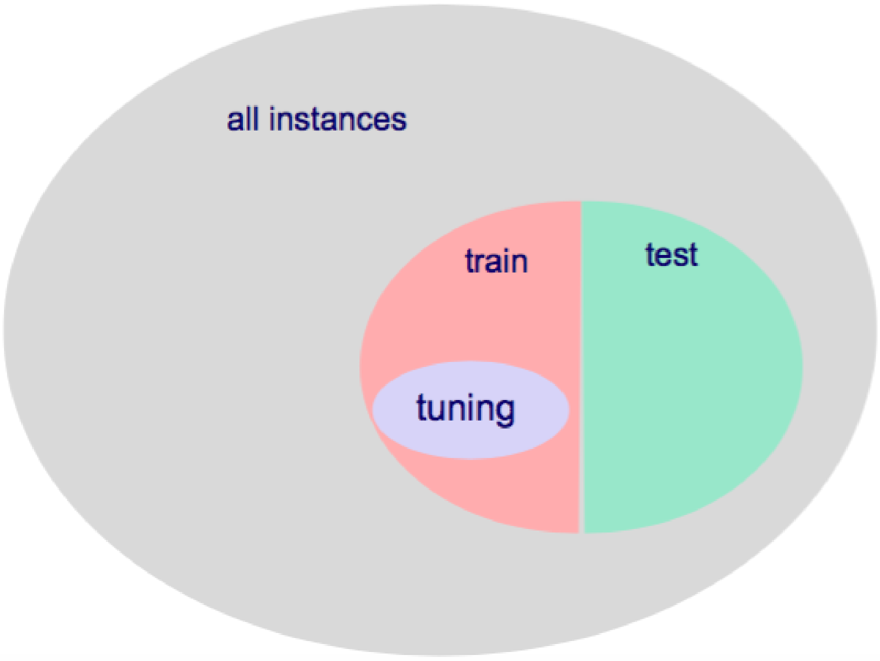
- We should form a leaf when
  - all of the given subset of instances are of the same class
  - we’ve exhausted all of the candidate splits
- Is there a reason to stop earlier, or to prune back the tree?

Pruning in C4.5

- split given data into training and *validation* (*tuning*) sets
- Grow a complete tree
- do until further pruning is harmful
  - evaluate impact on tuning-set accuracy of pruning each node
  - greedily remove the one that most improves tuning-set accuracy

Validation sets

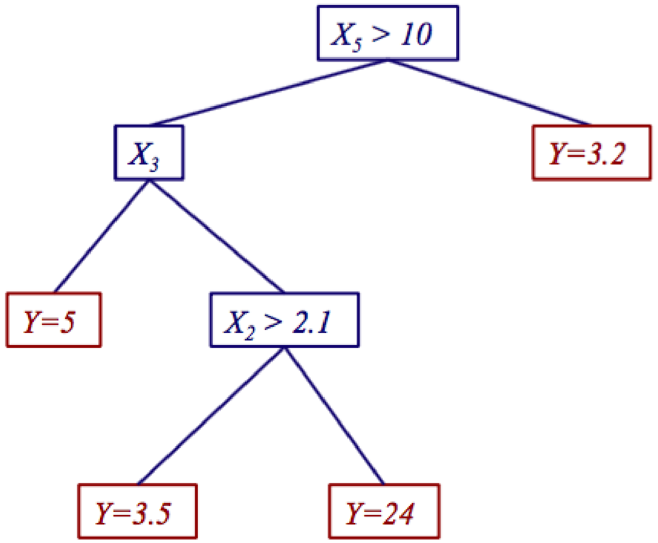
- a *validation set* (a.k.a. *tuning set*) is a subset of the training set that is held aside
  - not used for primary training process (e.g. tree growing)
  - but used to select among models (e.g. trees pruned to varying degrees)

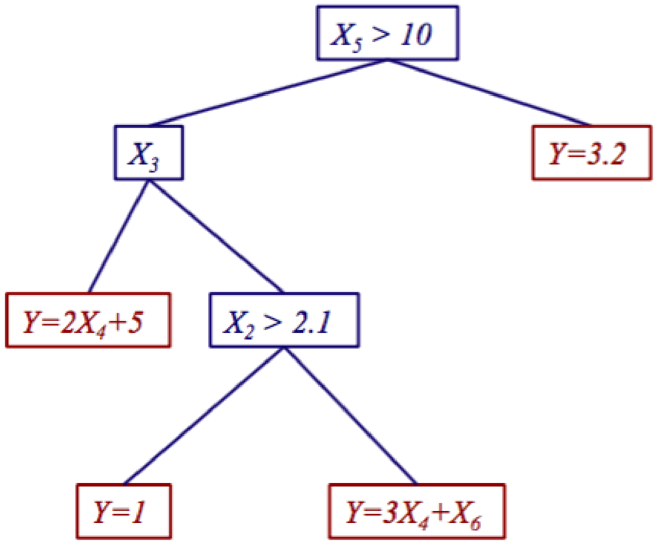


Variant: Regression Trees

Regression trees

- in a regression tree, leaves have functions that predict numeric values instead of class labels
- the form of these functions depends on the method
  - CART uses constants
  - some methods use linear functions





Regression trees in CART

- CART does *least squares regression* which tries to minimize

$$\sum_{i=1}^{|D|} (y^{(i)} - \hat{y}^{(i)})^2$$

target value for i^{th} training instance

value predicted by tree for i^{th} training instance (average value of y for training instances reaching the leaf)

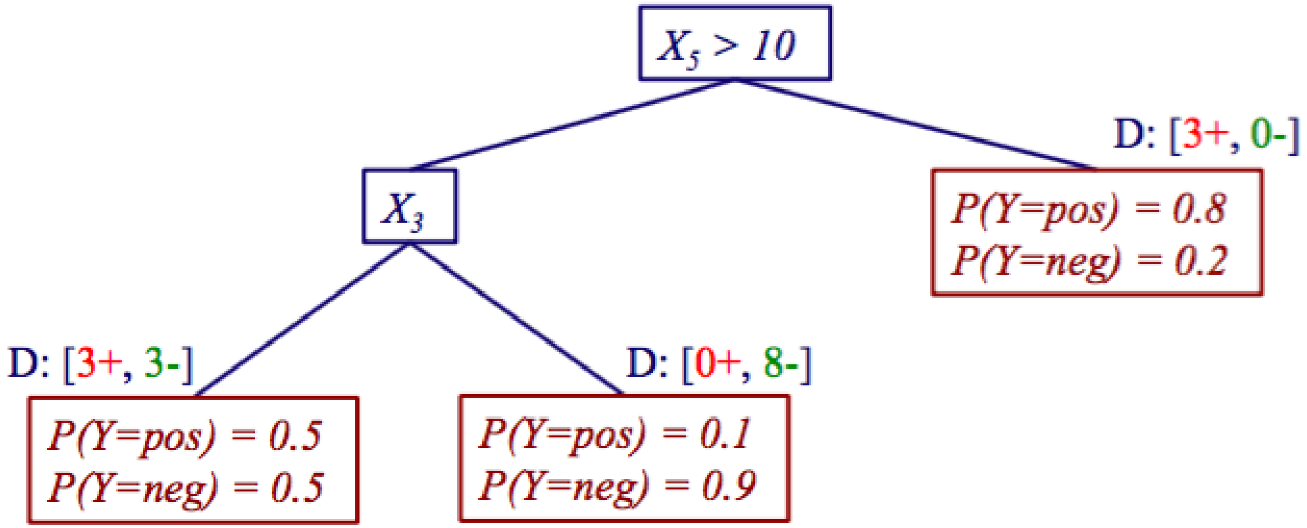
$$= \sum_{L \in \text{leaves}} \sum_{i \in L} (y^{(i)} - \hat{y}^{(i)})^2$$

- at each internal node, CART chooses the split that most reduces this quantity

Variant: Probability estimation trees

Probability estimation trees

- in a PE tree, leaves estimate the probability of each class
- could simply use training instances at a leaf to estimate probabilities, but ...
- *smoothing* is used to make estimates less extreme (we'll revisit this topic when we cover Bayes nets)

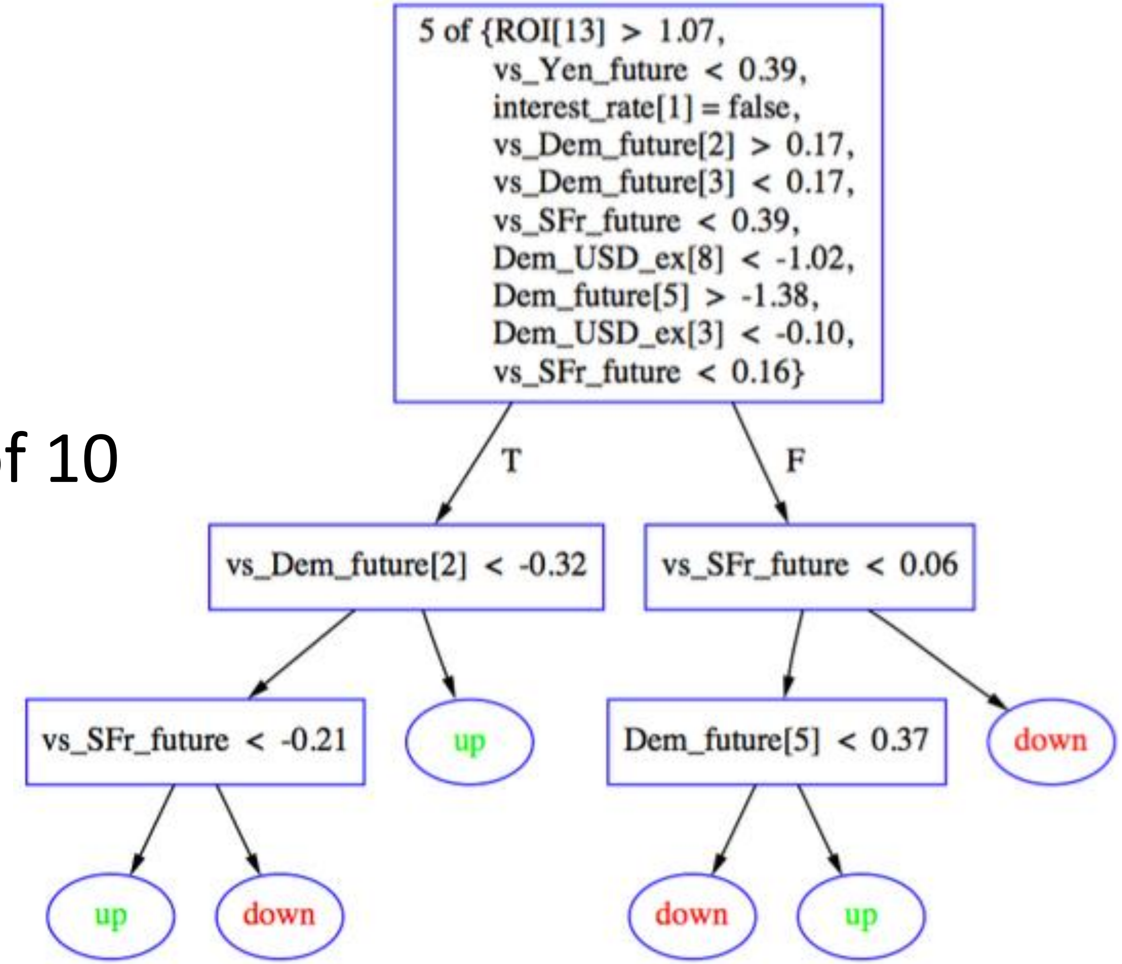


Variant: m-of-n splits

m-of-n splits

- a few DT algorithms have used m-of-n splits [Murphy & Pazzani '91]
- each split is constructed using a heuristic search process
- this can result in smaller, easier to comprehend trees

test is satisfied if 5 of 10 conditions are true



tree for exchange rate prediction
[Craven & Shavlik, 1997]

Searching for m-of-n splits

- m-of-n splits are found via a hill-climbing search
- initial state: best 1-of-1 (ordinary) binary split
- evaluation function: information gain
- operators:
  - m-of-n => m-of-(n+1)
  - 1 of { X1=t, X3=f } => 1 of { X1=t, X3=f, X7=t }
  - m-of-n => (m+1)-of-(n+1)
  - 1 of { X1=t, X3=f } => 2 of { X1=t, X3=f, X7=t }

Variant: Lookahead

Lookahead

- most DT learning methods use a hill-climbing search
- a limitation of this approach is myopia: an important feature may not appear to be informative until used in conjunction with other features
- can potentially alleviate this limitation by using a *lookahead* search [Norton ‘89; Murphy & Salzberg ‘95]
- empirically, often doesn’t improve accuracy or tree size

Choosing best split in ordinary DT learning

- OrdinaryFindBestSplit (set of training instances D, set of candidate splits C)

$maxgain = -\infty$

for each split S in C

  $gain$ = InfoGain(D, S)

  if $gain > maxgain$

    $maxgain = gain$

    $S_{best} = S$

return S_{best}

Choosing best split with lookahead (part 1)

- LookaheadFindBestSplit (set of training instances D, set of candidate splits C)

$maxgain$ = -∞

for each split S in C

  $gain$ = EvaluateSplit(D, C, S)

  if $gain > maxgain$

    $maxgain = gain$

    $S_{best} = S$

return S_{best}

Choosing best split with lookahead (part 2)

EvaluateSplit(D, C, S)

if a split on S separates instances by class (i.e. $H_D(Y \mid S) = 0$)

// no need to split further

return $H_D(Y) - H_D(Y \mid S)$

else

for each outcome k of S

// see what the splits at the next level would be

D_k = subset of instances that have outcome k

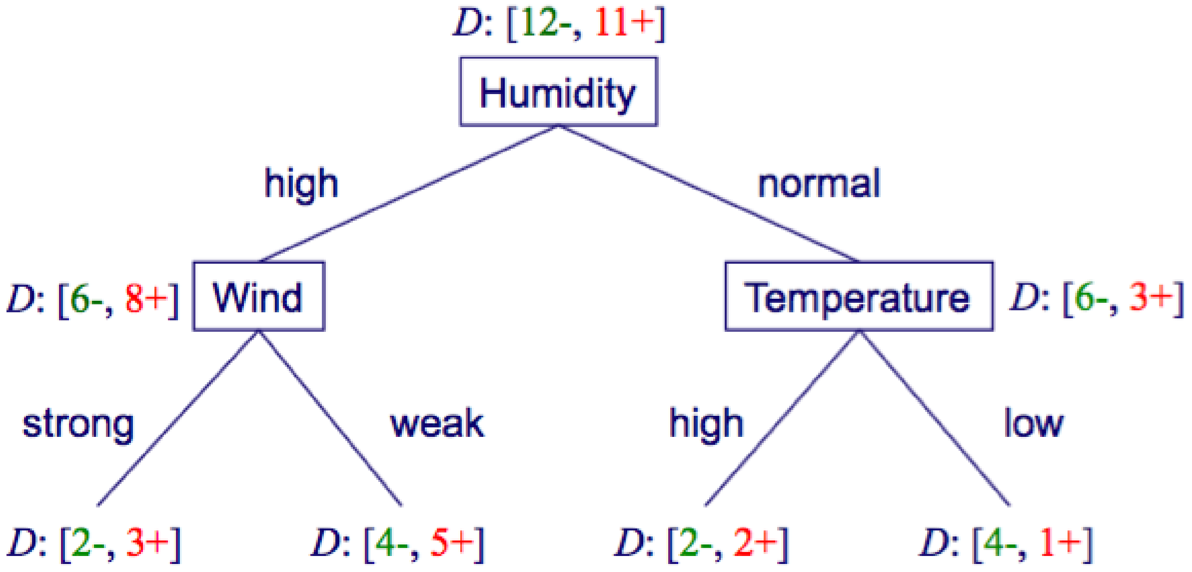
S_k = OrdinaryFindBestSplit(D_k, $C - S$)

// return information gain that would result from this 2-level subtree

return $$H_D(Y) - \left(\sum_k \frac{|D_k|}{|D|} H_{D_k}(Y \mid S = k, S_k \right)$$

Calculating information gain with lookahead

- Suppose that when considering Humidity as a split, we find that Wind and Temperature are the best features to split on at the next level



- We can assess value of choosing Humidity as our split by

$$H_D(Y) - \left(\frac{14}{23} H_D(Y \mid \text{Humidity} = \text{high}, \text{Wind}) + \frac{9}{23} H_D(Y \mid \text{Humidity} = \text{low}, \text{Temperature}) \right)$$

Calculating information gain with lookahead

- Using the tree from the previous slide:

$$\frac{14}{23} H_D(Y \mid \text{Humidity} = \text{high}, \text{Wind}) + \frac{9}{23} H_D(Y \mid \text{Humidity} = \text{low}, \text{Temperature})$$

$$= \frac{5}{23} H_D(Y \mid \text{Humidity} = \text{high}, \text{Wind} = \text{strong}) +$$

$$\frac{9}{23} H_D(Y \mid \text{Humidity} = \text{high}, \text{Wind} = \text{weak}) +$$

$$\frac{4}{23} H_D(Y \mid \text{Humidity} = \text{low}, \text{Temperature} = \text{high}) +$$

$$\frac{5}{23} H_D(Y \mid \text{Humidity} = \text{low}, \text{Temperature} = \text{low})$$

$$H_D(Y \mid \text{Humidity} = \text{high}, \text{Wind} = \text{strong}) = -\frac{2}{5} \log\left(\frac{2}{5}\right) - \frac{3}{5} \log\left(\frac{3}{5}\right)$$

Comments on decision tree learning

- widely used approach
- many variations
- provides humanly comprehensible models when trees not too big
- insensitive to monotone transformations of numeric features
- standard methods learn axis-parallel hypotheses*
- standard methods not suited to on-line setting*
- usually not among most accurate learning methods

* although variants exist that are exceptions to this