Decision Trees

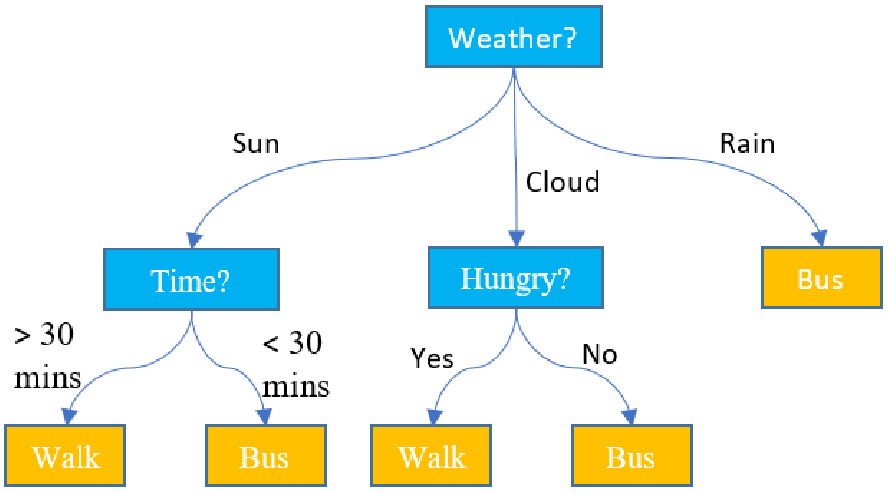
Outline

- Decision Tree
  - One of the primitive machine learning algorithm (Ross Quinlan 1986)
  - Beginning of machine learning era
  - Some theories behind the method
  - Later, evolved into Random Forest

- Contents
  - Top-Down Decision Tree Construction
  - Choosing the Splitting Attribute
  - Use Entropy, Information Gain and Gain Ratio
  - Prunning methods after tree construction

DECISION TREE

- An internal node is a test on an attribute.
- A branch represents an outcome of the test, e.g., Color=red.
- A leaf node represents a class label or class label distribution.
- At each node, one attribute is chosen to split training examples into distinct classes as much as possible
- A new case is classified by following a matching path to a leaf node.



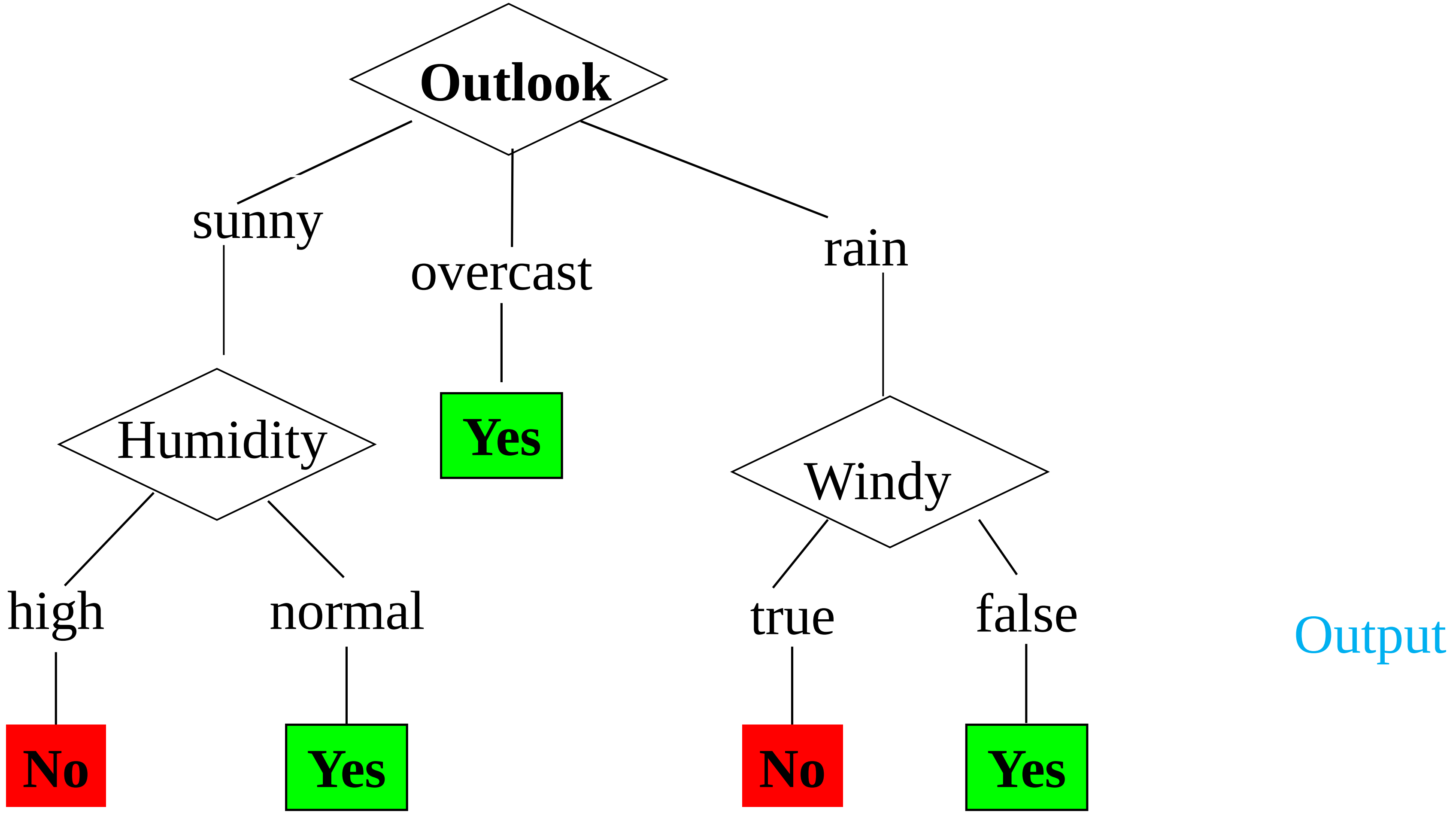
Weather Data: Play or not Play?

Outlook	Temperature	Humidity	Windy	Play?
sunny	hot	high	false	No
sunny	hot	high	true	No
overcast	hot	high	false	Yes
rain	mild	high	false	Yes
rain	cool	normal	false	Yes
rain	cool	normal	true	No
overcast	cool	normal	true	Yes
sunny	mild	high	false	No
sunny	cool	normal	false	Yes
rain	mild	normal	false	Yes
sunny	mild	normal	true	Yes
overcast	mild	high	true	Yes
overcast	hot	normal	false	Yes
rain	mild	high	true	No

Note:
Outlook is the Forecast,
no relation to Microsoft email program

Input

Example Tree for "Play?"



Building Decision Tree

- Phase 1: Top-down tree construction
  - Starts with a root node/attribute
  - Recursively adding new nodes/attributes to the branches of parental nodes.
- Phase 2: Bottom-up tree pruning
  - Remove subtrees or branches, in a bottom-up (or top-down) manner, to improve the estimated accuracy on new cases.

Choosing the Splitting Attribute

- At each node, available attributes are evaluated on the basis of separating the classes of the training examples. A Goodness function is used for this purpose.

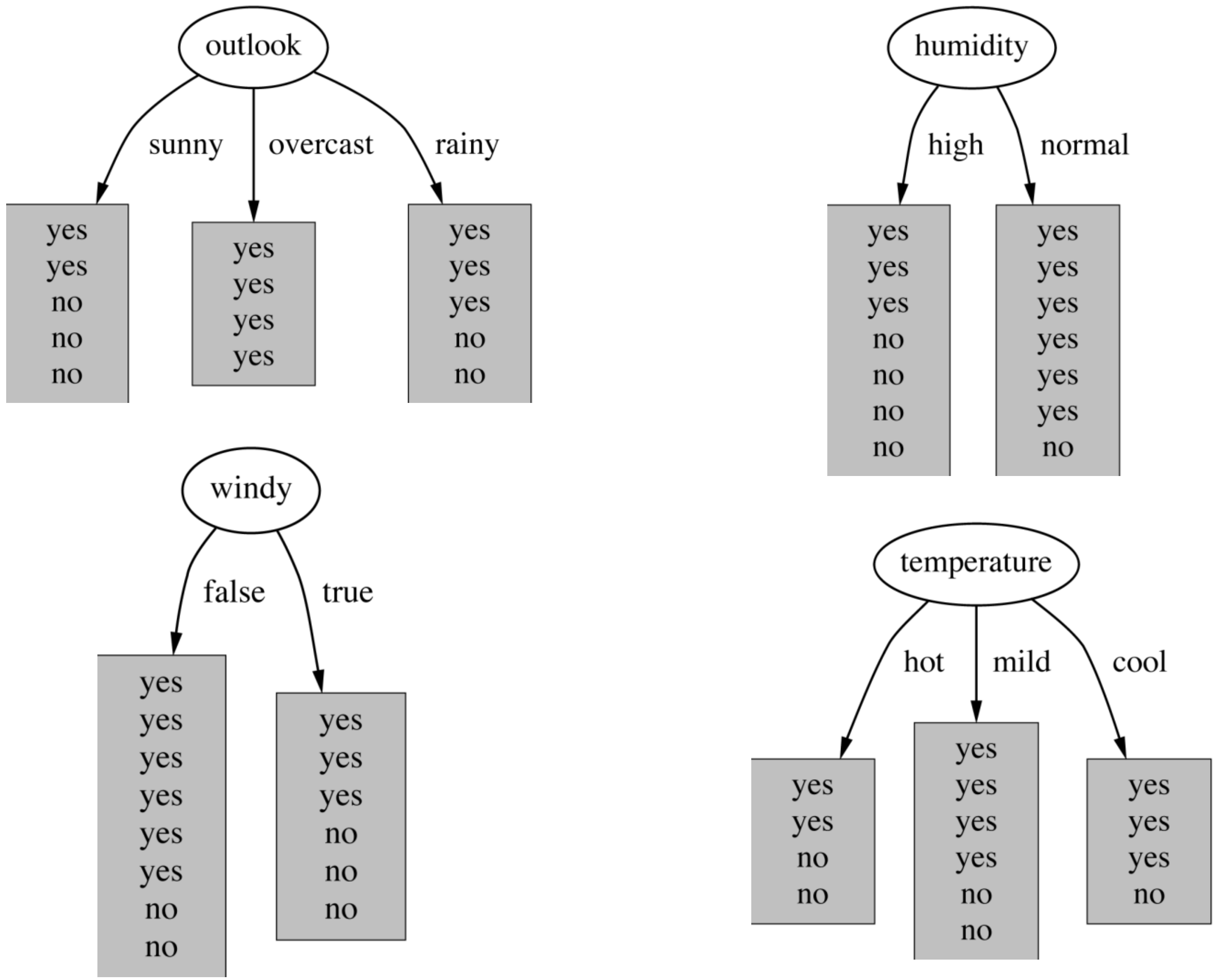
- Typical goodness functions:
  - entropy
  - information gain (ID3/C4.5)
  - information gain ratio
  - gini index

Which attribute to select?

A	B	C	D		Class
0		0	0		Y
0		1	0		Y
0		0	1		Y
0		1	0		Y
1		0	0		N
1		1	1		N
1		0	1		N
1		1	0		N

Attribute A vs attribute C ?

Which attribute to select?

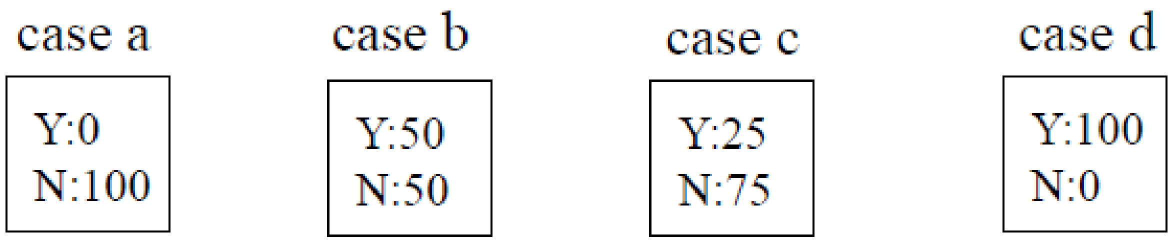


A criterion for attribute selection

- Which is the best attribute?
  - Heuristic: choose the attribute that produces the "purest" nodes
  - The attribute with "pure" class distribution
- How do we measure the degree of purity/certainty/clearness ?
  - or degree of impurity/uncertainty/noisiness
- Popular criterion: entropy, information gain (ratio), gini index

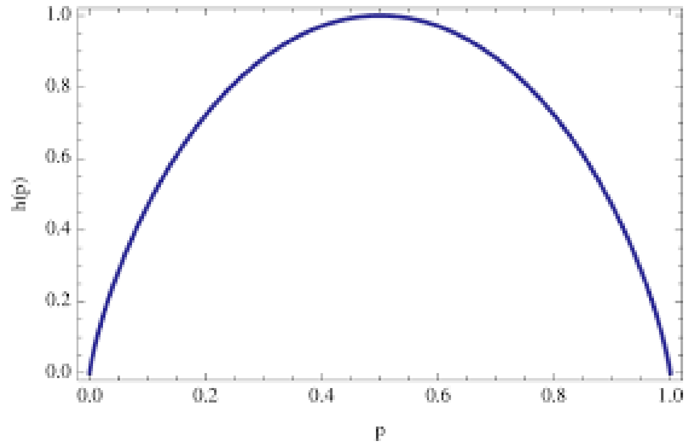
A	Class
0	Y
0	Y
0	Y
0	Y
1	N
1	N
1	N
1	N

A criterion for attribute selection



- Which one is more impure/uncertain?
- Question now is how do we represent the degree of impurity/uncertainty?
- We need a function or formula for that.

Shannon Entropy



- How do we measure the degree of impurity?
- Entropy function(Shannon entropy):

  1) represents the degree of impurity

  in prob. dist.

  2) also represents the amount of information prob. distribution contains

- Information is measured in *bits*
  - Given a probability distribution, the info required to predict an event is the distribution's *entropy*
  - Entropy gives the information required in bits (this can involve fractions of bits!)
- Formula for computing the entropy entropy*(log=log2)*

$$\text{entropy}(p_1, p_2, \ldots, p_n) = -p_1 \log p_1 - p_2 \log p_2 \ldots - p_n \log p_n$$

Entropy

- Entropy as the measure of uncertainty(impurity)

bag a

B:0 W:100

P_b=0, P_w=1

entropy = 0

bag b

B:50 W:50

P_b=0.5, P_w=0.5

entropy = 1

bag c

B:25 W:75

P_b=0.25, P_w=0.75

entropy = -0.25 log0.25
-0.75 log0.75

Entropy: the amount of info.

- Entropy represents
  1) weighted average of self-information
  2) the amount of information needed to remove uncertainty from the system

- With a variable X with events $x_1, x_2, \dots, x_n$
  - entropy $= -\sum_i P(x_i) \log P(x_i)$

- For an event x_i,
  - probability of event x_i : $P(x_i)$
  - self-information of event x_i : $\mathrm{I}(\mathrm{x_i}) = -\log P(x_i)$
  - average self-information of X : $\mathrm{E}[\mathrm{I}(\mathrm{x_i})] = -\sum_i P(x_i) \log P(x_i)$
  - entropy of X = average self-information of a variable

Self-information

bag a	bag b	bag c
B:0 W:100	B:50 W:50	B:25 W:75

- Suppose you pick one ball randomly out of these bags.
- You goal is to know the color of the ball (Black or White)

- You pick one ball out of bag a:
  - How much information do you get ?

amount of info(W) = - log(p(W))
= - log(1)=0

- You pick one ball out of bag b:
  - How much information do you get

amount of info(W) = - log(p(W))
= - log(0.5)=1

- You pick one ball out of bag c:
  - How much information do you get

amount of info(W) = - log(p(W))
= - log(0.75)=0.415

Example: entropy of attribute Outlook

- "Outlook" = "Sunny":

$$\text{info}([2,3]) = \text{entropy}(2/5,3/5) = -2/5\log(2/5) - 3/5\log(3/5) = 0.971\,\text{bits}$$

- "Outlook" = "Overcast":

$$\text{info}([4,0]) = \text{entropy}(1,0) = -1\log(1) - 0\log(0) = 0\,\text{bits}$$

Note: log(0) is not defined, but we evaluate 0*log(0) as zero

- "Outlook" = "Rainy":

$$\text{info}([3,2]) = \text{entropy}(3/5,2/5) = -3/5\log(3/5) - 2/5\log(2/5) = 0.971\,\text{bits}$$

- Expected information for attribute: (weighted average)

$$\begin{aligned}\text{info}([3,2],[4,0],[3,2]) &= (5/14)\times 0.971 + (4/14)\times 0 + (5/14)\times 0.971 \\ &= 0.693\,\text{bits}\end{aligned}$$

Computing the information gain

- Decision tree uses information gain

- Popular impurity criterion: information gain
- Information gain increases with the average purity of the subsets that an attribute produces

- Starategy: choose attribute that results in greatest information gain

- Information gain considers a priori probability distribution of class(C) values
  - Information gain(attribute)
    = entropy(class) – entropy(attribute)

- Information gain is a better measure in skewed data
  - when class dist. is (~1,~0), entropy(0.5,0.5) gives a lot of information

Computing the information gain

- Information gain:

(entropy before split) – (entropy after split) =

(entropy of class values) – (entropy of attribute)

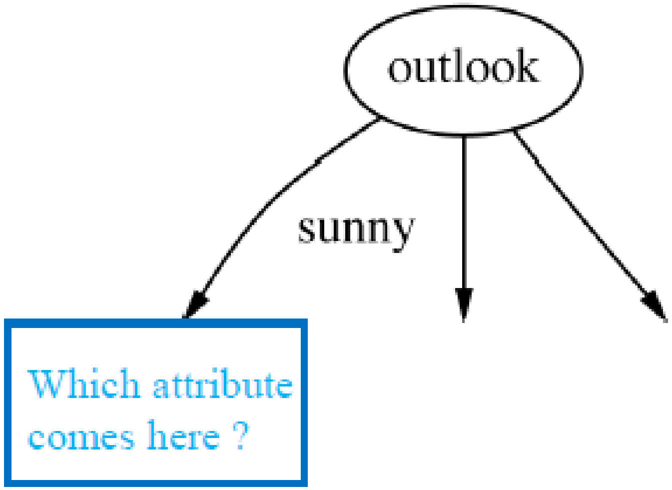
$$\text{gain}(\text{"Outlook"}) = \text{info}([9,5]) - \text{info}([2,3],[4,0],[3,2]) = 0.940 - 0.693$$
$$= 0.247 \text{ bits}$$

- Information gain for attributes from weather data:

$$\text{gain}(\text{"Outlook"}) = 0.247 \text{ bits}$$
$$\text{gain}(\text{"Temperature"}) = 0.029 \text{ bits}$$
$$\text{gain}(\text{"Humidity"}) = 0.152 \text{ bits}$$
$$\text{gain}(\text{"Windy"}) = 0.048 \text{ bits}$$

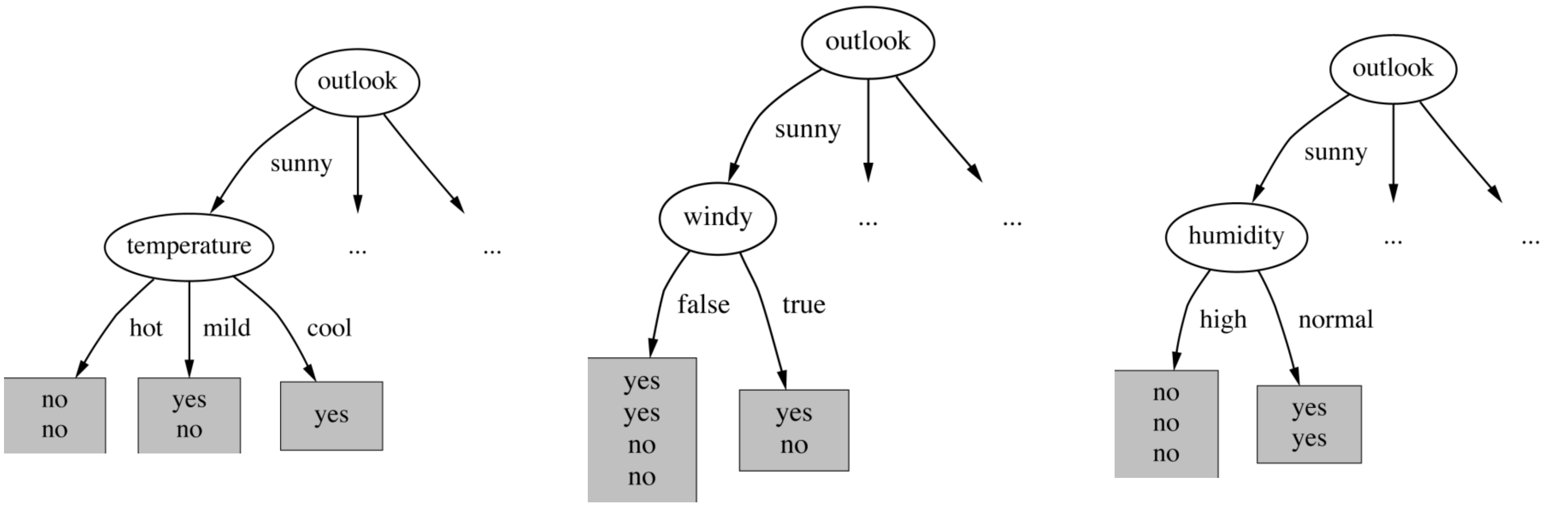
Continuing to split

So Outlook becomes the root of the decision tree



Remember: Now we consider "outlook=sunny" data only

Continuing to split

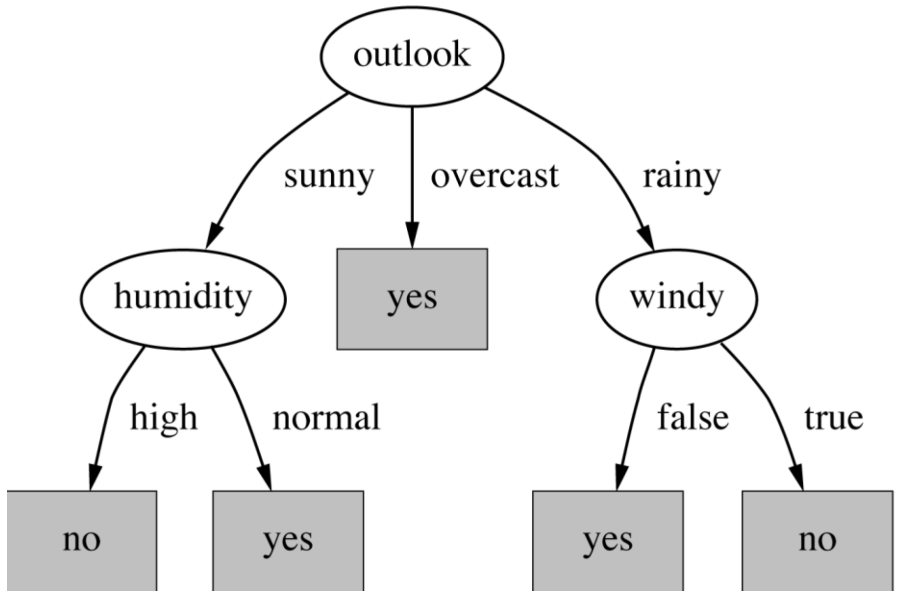


$$\text{gain}(\text{"Humidity"}) = 0.971\ \text{bits}$$

$$\text{gain}(\text{"Temperature"}) = 0.571\ \text{bits}$$

$$\text{gain}(\text{"Windy"}) = 0.020\ \text{bits}$$

The final decision tree



- Note: not all leaves need to be pure; sometimes identical instances have different classes

  ⇒ Splitting stops when data can’t be split any further

CART Splitting Criteria: Gini Index

- If a data set T contains examples from n classes, gini index, gini(T) is defined as

$$gini(T) = 1 - \sum_{j=1}^{n} p_j^2$$

  where p_j is the relative frequency of class j in T

- gini(T) is minimized if the classes in T are skewed.
- Difference between entropy:

  - *entropy before split* is not considered

  - no a priori distribution

  - make difference in skewed distributions

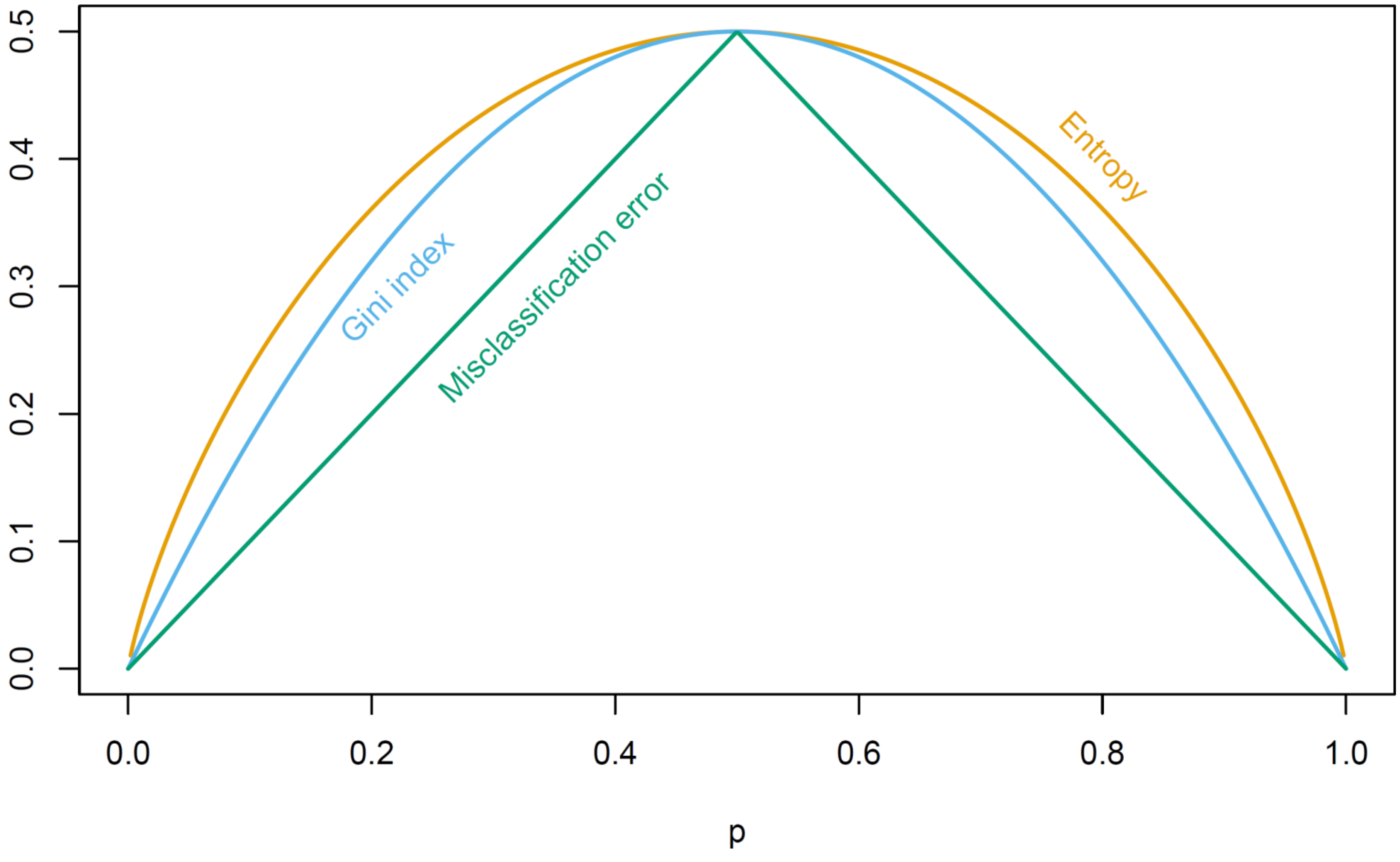
Gini Index

- After splitting T into two subsets T1 and T2 with sizes N1 and N2, the gini index of the split data is defined as

$$gini_{split}(T) = \frac{N_1}{N} gini(T_1) + \frac{N_2}{N} gini(T_2)$$

- The attribute providing smallest $gini_{split}(T)$ is chosen to split the node.

Node (im)purity measures for K=2



Hastie *et al* (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd Ed.

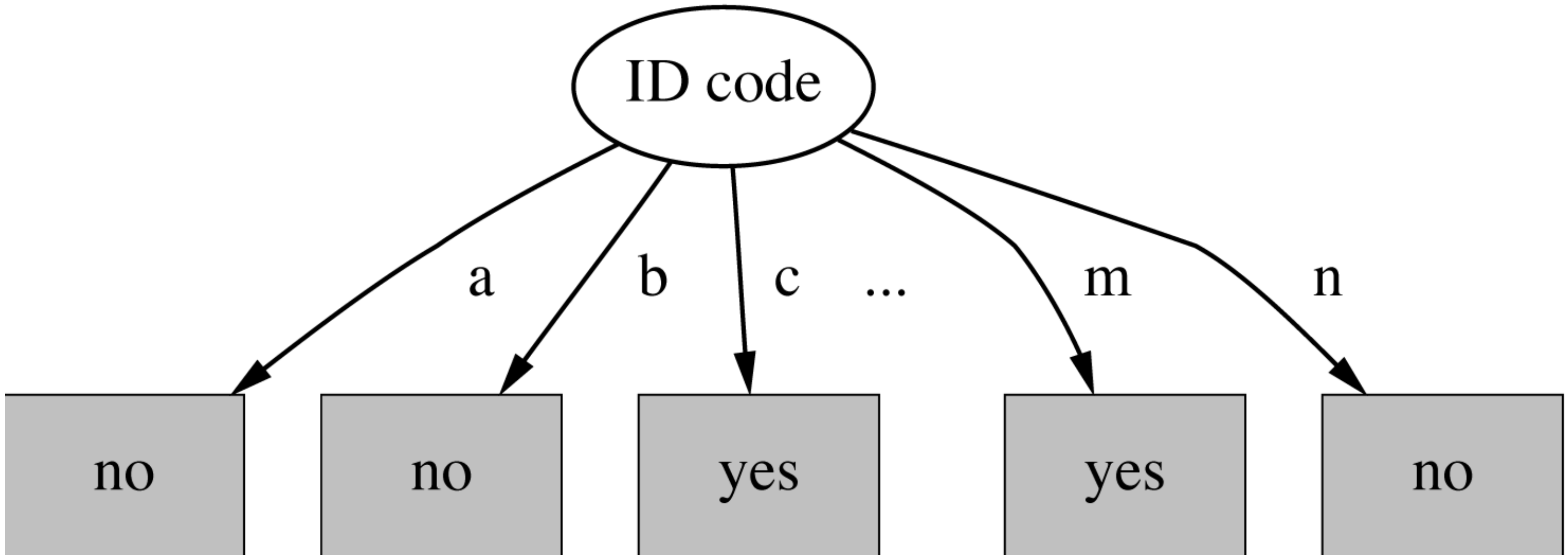
Problem of highly-branching attributes

- Problematic: attributes with a large number of values (extreme case: ID code)
- Subsets are more likely to be pure if there is a large number of values
  - $\Rightarrow$ Information gain is biased towards choosing attributes with a large number of values
  - $\Rightarrow$ This may result in *overfitting* (selection of an attribute that is non-optimal for prediction)

Weather Data with ID code

ID	Outlook	Temperature	Humidity	Windy	Play?
A	sunny	hot	high	false	No
B	sunny	hot	high	true	No
C	overcast	hot	high	false	Yes
D	rain	mild	high	false	Yes
E	rain	cool	normal	false	Yes
F	rain	cool	normal	true	No
G	overcast	cool	normal	true	Yes
H	sunny	mild	high	false	No
I	sunny	cool	normal	false	Yes
J	rain	mild	normal	false	Yes
K	sunny	mild	normal	true	Yes
L	overcast	mild	high	true	Yes
M	overcast	hot	normal	false	Yes
N	rain	mild	high	true	No

Split for ID Code Attribute



- Entropy of split = 0 (since each leaf node is "pure", having only one case.
- Information gain is maximal for ID code

Gain ratio

- *Gain ratio*: a modification of the information gain that reduces its bias on high-branch attributes
- Gain ratio should be
  - Large when data is evenly spread
  - Small when all data belong to one branch
- Gain ratio takes number and size of branches into account when choosing an attribute
  - It corrects the information gain by taking the *intrinsic information* of a split into account (i.e. how much info do we need to tell which branch an instance belongs to)

Gain Ratio and Intrinsic Info.

- *Intrinsic information*: entropy of distribution of instances into branches

$$IntrinsicInfo(S,A) \equiv -\sum \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}.$$

- *Gain ratio* (Quinlan'86) normalizes info gain by:

$$GainRatio(S,A) = \frac{Gain(S,A)}{IntrinsicInfo(S,A)}.$$

Computing the gain ratio

- Example: intrinsic information for ID code

$$\text{info}([1,1,\ldots,1]) = 14 \times (-1/14 \times \log 1/14) = 3.807 \text{ bits}$$

- Importance of attribute decreases as intrinsic information gets larger
- Example of gain ratio:

$$\text{gain\_ratio}(\text{"Attribute"}) = \frac{\text{gain}(\text{"Attribute"})}{\text{intrinsic\_info}(\text{"Attribute"})}$$

- Example: $\text{gain\_ratio}(\text{"ID\_code"}) = \dfrac{0.940 \text{ bits}}{3.807 \text{ bits}} = 0.246$

Gain ratios for weather data

Outlook		**Temperature**	
Info:	0.693	Info:	0.911
Gain: 0.940-0.693	0.247	Gain: 0.940-0.911	0.029
Split info: info([5,4,5])	1.577	Split info: info([4,6,4])	1.362
Gain ratio: 0.247/1.577	0.156	Gain ratio: 0.029/1.362	0.021

Humidity		**Windy**	
Info:	0.788	Info:	0.892
Gain: 0.940-0.788	0.152	Gain: 0.940-0.892	0.048
Split info: info([7,7])	1.000	Split info: info([8,6])	0.985
Gain ratio: 0.152/1	0.152	Gain ratio: 0.048/0.985	0.049

More on the gain ratio

- "Outlook" still comes out top
- However: "ID code" has greater gain ratio
  - Standard fix: *ad hoc* test to prevent splitting on that type of attribute
- Problem with gain ratio: it may overcompensate
  - May choose an attribute just because its intrinsic information is very low
  - Standard fix:
    - First, only consider attributes with greater than average information gain
    - Then, compare them on gain ratio

Decision Tree in sklearn

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
pima = pd.read_csv("diabetes.csv", header=None, names=col_names)

#split dataset in features and target variable
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
X = pima[feature_cols] # Features
y = pima.label # Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)
```

Decision Tree in sklearn

```
#Predict the response for test dataset
y_pred = clf.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

# Visualization of decision  tree
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus

dot_data = StringIO()
export_graphviz(clf, out_file=dot_data, filled=True, rounded=True,
                special_characters=True, feature_names = feature_cols,
                class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```

Decision Tree Pruning

Overfitting

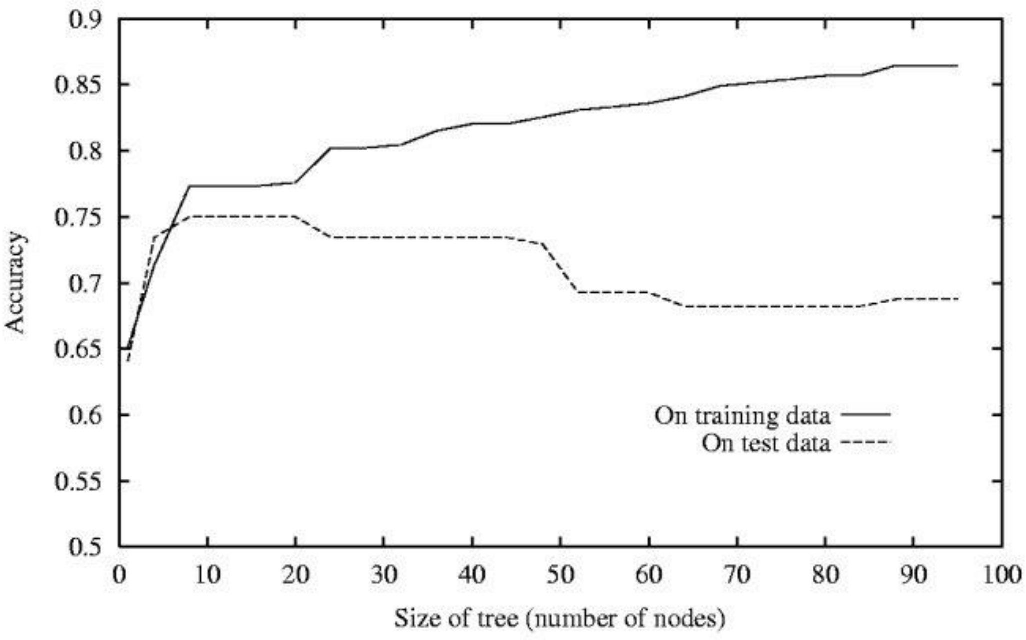
- Why do we prune decision trees?
  - avoid overfitting problem by making trees smaller

- Classifier h overfits iff ∃ classifier h' with
  - $error_{train}(h) < error_{train}(h')$ &
  - $error_{test}(h) > error_{test}(h')$

- Classifier h **overfits** the data if there exists h' with greater error than h over training data but less error than h over test data

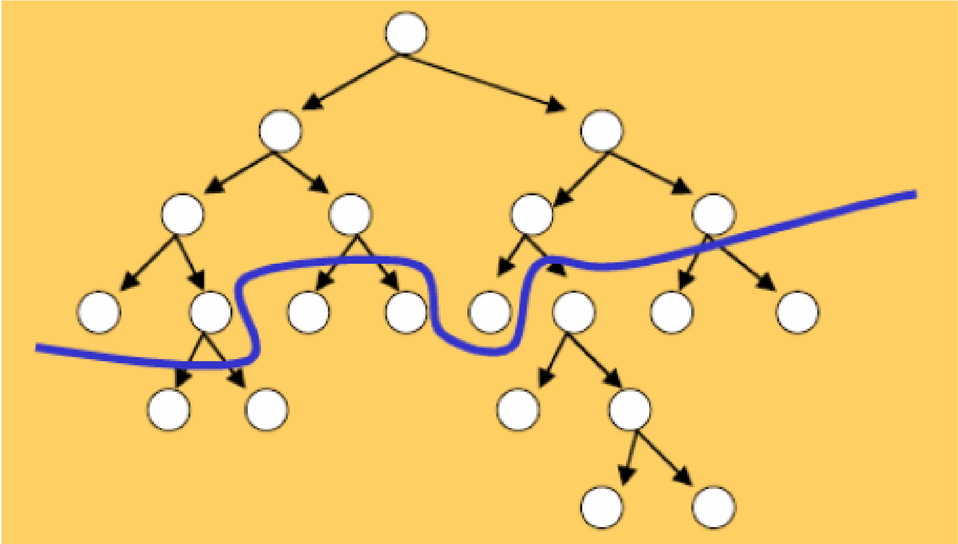
Overfitting

- Serious problem for most machine learning methods
- Complex trees actually degrade the performance (simpler one is better)
- Trees may grow to include irrelevant attributes (e.g., Date, Color, etc.)
- Noisy examples may add spurious nodes to tree



Tree Pruning

- Avoid ovefitting the data by tree pruning
- After pruning the classification accuracy on unseen data may increase!



Tree Pruning

- Goal: Prevent overfitting to noise in the data

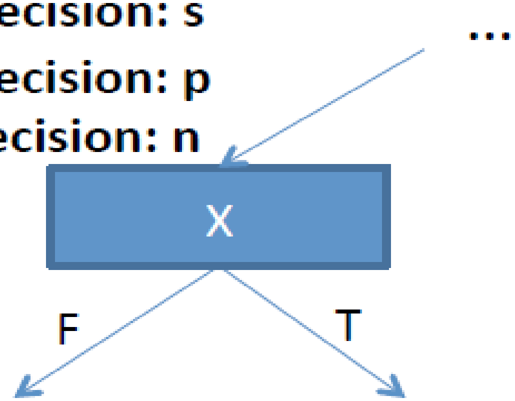
- Pre pruning: stop growing the tree earlier
  - Use Chi-square test
  - Stops growing if the number of samples in leaf nodes is too small to make a reliable decision(min # of objects pruning)
  - Stops if a proportion of a single class is larger than a given threshold

- Post pruning: allow overfit and then post-prune the tree
  - Reduced Error Pruning
  - Subtree Replacement
    - Estimation of errors to decide which subtrees should be pruned

- Postpruning preferred in practice—prepruning can “stop too early”

Chi-Square Test

of instances entering this decision: s
of + instances entering this decision: p
of - instances entering this decision: n



instances here: s_f
of + instances here: p_f
of - instances here: n_f

instances here: s_t
of - instances here: p_t
of - instances here: n_t

Class value

X

	p	n	Total
T	p_t	n_t	p_t+n_t
F	p_f	n_f	p_f+n_f
Total	p_t+p_f	n_t+n_f	$p_t+n_t+p_f+n_f$

Calculate the Chi-square (χ^2) value between X (T/F) and label (p/n)

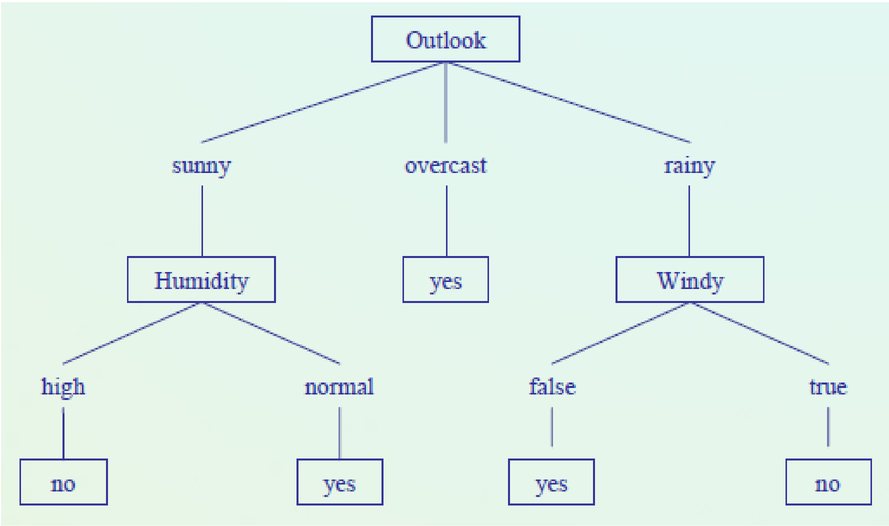
Post-pruning

- First, build full tree
- Then, prune it
  - Fully-grown tree shows all attribute interactions
- Problem: some subtrees might be due to chance effects

- Two pruning operations:
  1. *Reduced Error Pruning*
  2. *Subtree replacement*

Reduced Error Pruning(REP)

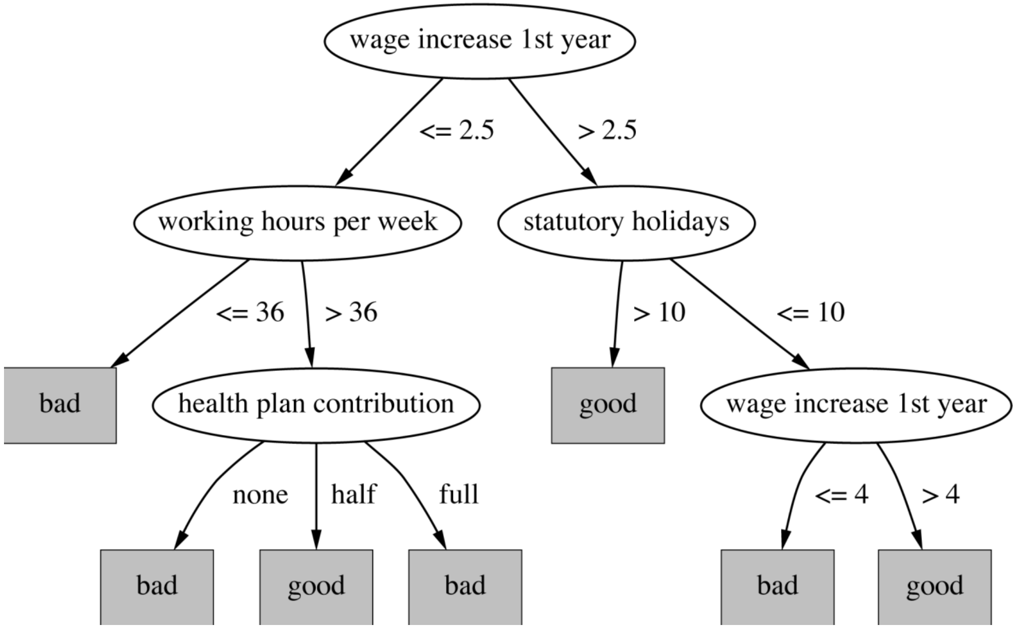
- Split training data into training data and validation data
- Construct decision tree in full and consider each node for pruning
- Pruning = removing the subtree at that node, make it a leaf and assign the most common class at that node
- A node is removed if the resulting tree performs no worse then the original on the validation data
- Nodes are removed iteratively choosing the node whose removal most increases the decision tree accuracy
- Pruning continues until further pruning is harmful



Subtree Replacement

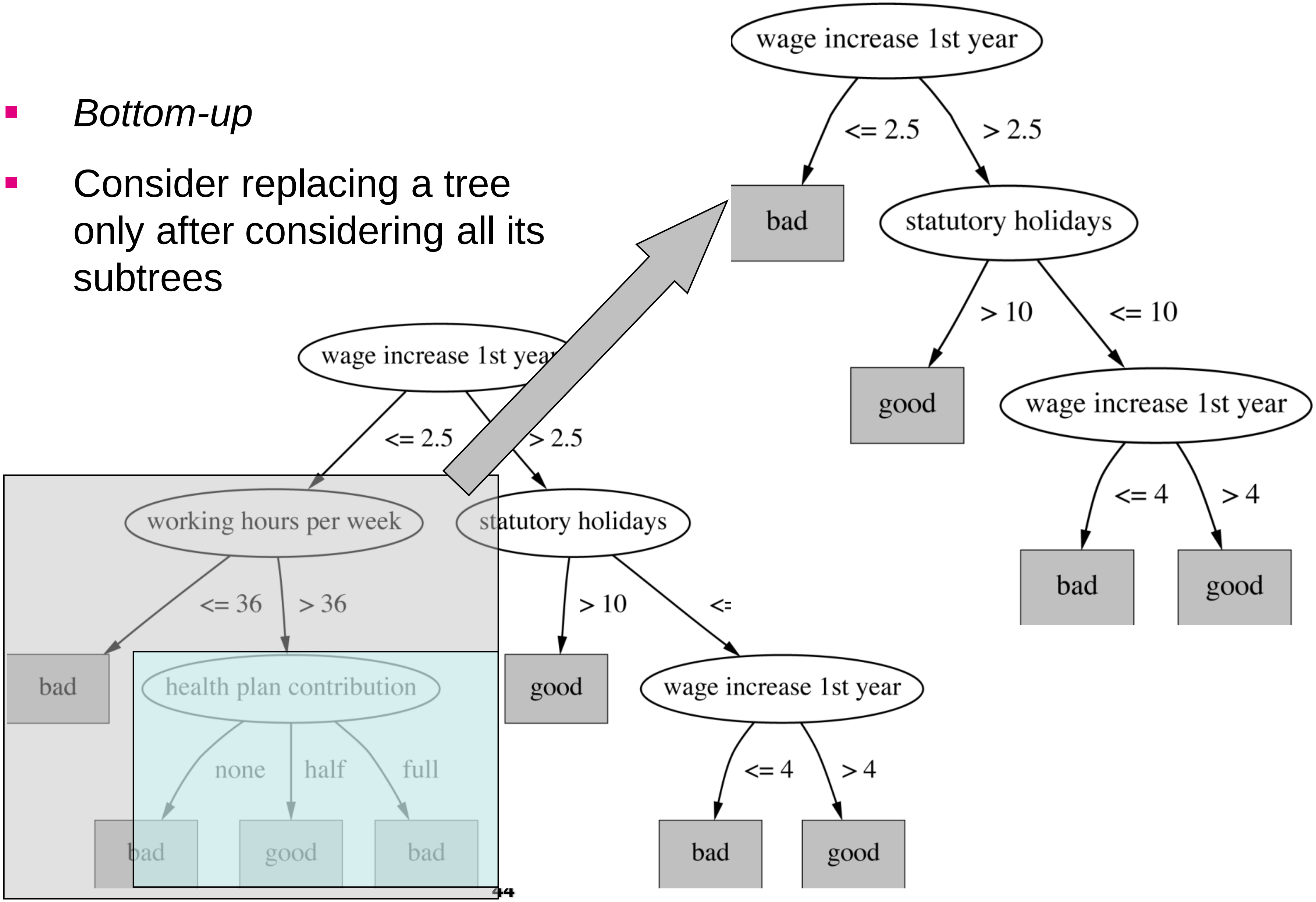
- For each node, estimate
  1) the error of the node (before pruning)
  2) the error of the node after pruning
- Prune it if the pruning decreases the estimate of error

- Consider replacing a tree only after considering all its subtrees
- Bottom-up approach
- Ex: labor negotiations



Subtree Replacement

- *Bottom-up*
- Consider replacing a tree only after considering all its subtrees



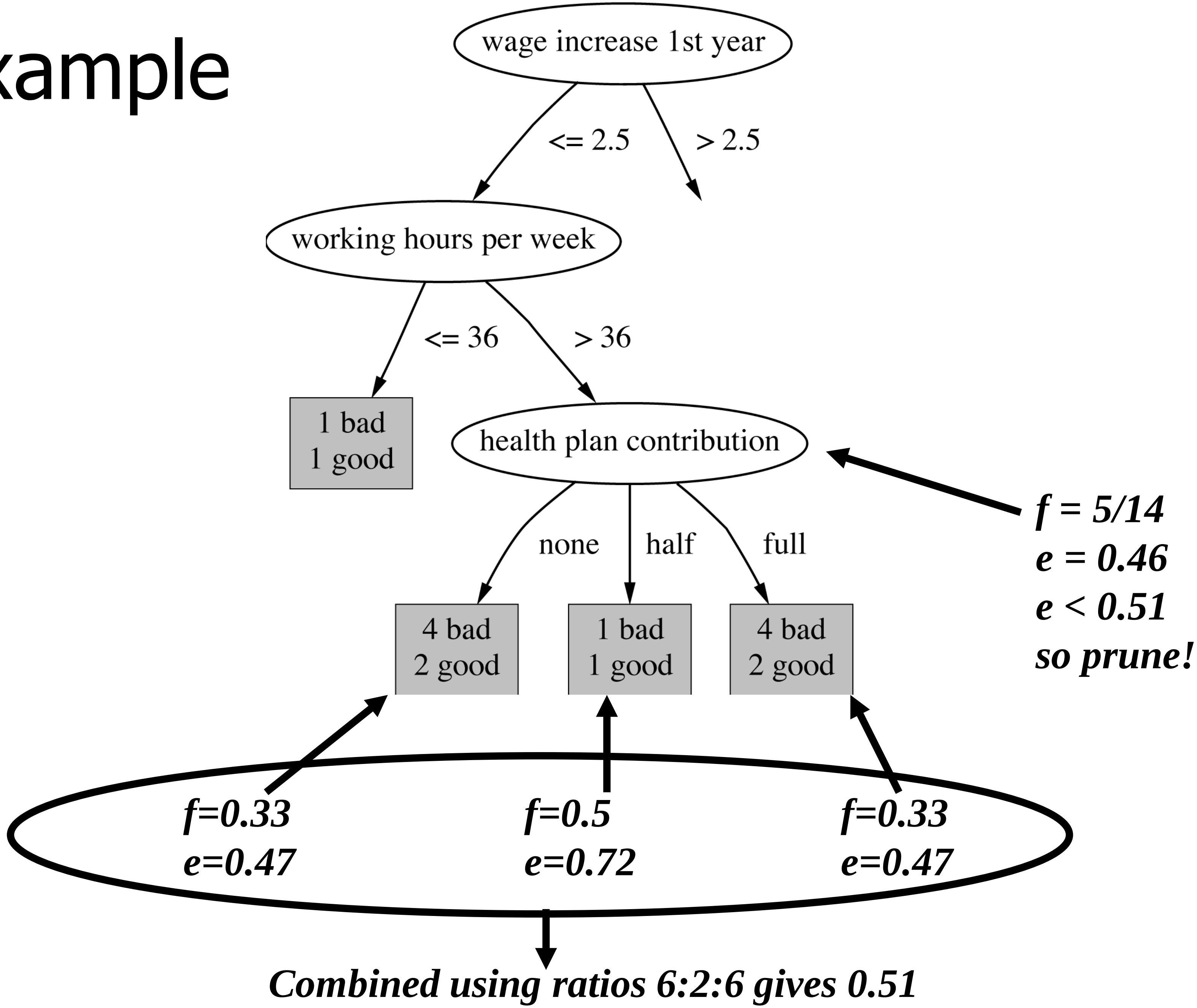
Subtree Replacement

- Error estimate for subtree is weighted sum of error estimates for all its leaves
- Error estimate for a node (upper bound): (skipped derivation)

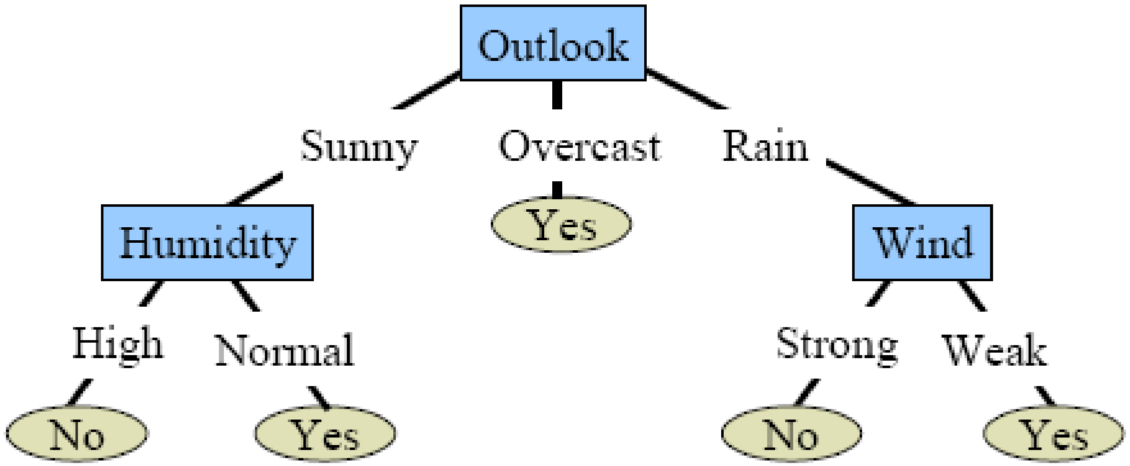
$$e = \left(f + \frac{z^2}{2N} + z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}} \right) \bigg/ \left(1 + \frac{z^2}{N} \right)$$

- If *c* = 25% then *z* = 0.69 (from normal distribution)
- *f* is the error on the training data
- *N* is the number of instances covered by the leaf

Example


wage increase 1st year
<= 2.5
> 2.5
working hours per week
<= 36
> 36
1 bad
1 good
health plan contribution
none
half
full
4 bad
2 good
1 bad
1 good
4 bad
2 good
f = 5/14
e = 0.46
e < 0.51
so prune!
f=0.33
e=0.47
f=0.5
e=0.72
f=0.33
e=0.47
Combined using ratios 6:2:6 gives 0.51

From trees to rules



If	(outlook = sunny)	$\wedge$	(humidity=high)	then	PlayTennis=No
If	(outlook = sunny)	$\wedge$	(humidity=normal)	then	PlayTennis=Yes
If	(outlook = overcast)			then	PlayTennis=Yes
If	(outlook = rain)	$\wedge$	(wind=strong)	then	PlayTennis=No
If	(outlook = rain)	$\wedge$	(wind=weak)	then	PlayTennis=Yes

- One path = one rule
- Can be used for post-pruning