# Tree Learning

COMP9417 Machine Learning and Data Mining

Term 2, 2021

## Acknowledgements

Material derived from slides for the book
"Machine Learning" by T. Mitchell
McGraw-Hill (1997)
http://www-2.cs.cmu.edu/~tom/mlbook.html

Material derived from slides by Andrew W. Moore
http:www.cs.cmu.edu/~awm/tutorials

Material derived from slides by Eibe Frank
http://www.cs.waikato.ac.nz/ml/weka

Material derived from slides for the book
"Machine Learning" by P. Flach
Cambridge University Press (2012)
http://cs.bris.ac.uk/~flach/mlbook

# Aims

This lecture will enable you to describe decision tree learning, the use of entropy and the problem of overfitting. Following it you should be able to:

- define the decision tree representation
- list representation properties of data and models for which decision trees are appropriate
- reproduce the top-down decision tree induction (TDIDT) algorithm
- define entropy and information gain for the TDIDT algorithm
- describe the inductive bias of the TDIDT algorithm
- define overfitting of a training set by a hypothesis
- describe developments of the basic TDIDT algorithm: pruning, rules, numeric attributes, many-valued attributes, missing values
- describe regression and model trees

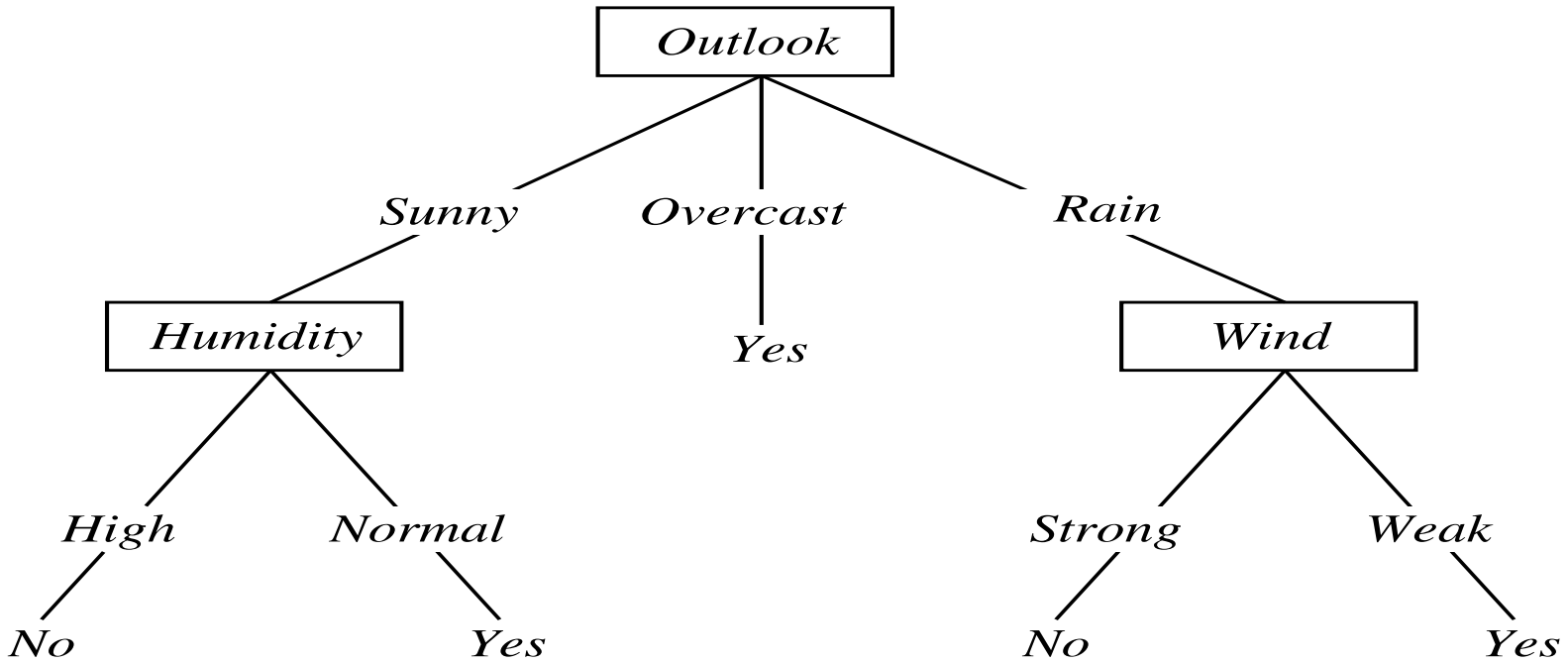
# Why use decision trees?

- Trees in some form are probably still the single most popular data mining tool
  - Easy to understand
  - Easy to implement
  - Easy to use
  - Computationally efficient (even on big data) to learn and run
- They do *classification*, i.e., predict a categorical output from categorical and/or real inputs
- Tree learning can also be used for predicting a real-valued output, i.e., they can do *regression*
- There are some drawbacks, though — e.g., can have high variance (later lecture)

# Training Examples

| 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 Trees

Decision tree representation:

- Each internal node tests an attribute (feature)
- Each branch corresponds to attribute (feature) value or threshold
- Each leaf node assigns a classification value

How would we represent the following expressions ?

- $\wedge, \vee$, XOR
- $M$ of $N$
- $(A \wedge B) \vee (C \wedge \neg D \wedge E)$

# Decision Trees

$X \wedge Y$

```
X = t:
| Y = t: true
| Y = f: no
X = f: no
```

$X \vee Y$

```
X = t: true
X = f:
| Y = t: true
| Y = f: no
```

## Decision Trees

$X \text{ XOR } Y$

```
X = t:
| Y = t: false
| Y = f: true
X = f:
| Y = t: true
| Y = f: false
```

So decision trees are, in some sense, *non-linear* classifiers (or regressors).

## Decision Trees

2 of 3

```
X = t:
| Y = t: true
| Y = f:
| | Z = t: true
| | Z = f: false
X = f:
| Y = t:
| | Z = t: true
| | Z = f: false
| Y = f: false
```

In general, decision trees represent a *disjunction of conjunctions* of constraints, or tests, on the attributes values of instances.

# When are Decision Trees the Right Model?

- With Boolean values for instances $\mathbf{X}$ and class $Y$, decision-trees represent $Y$ as a Boolean function of the $\mathbf{X}$
- Given $d$ input Boolean variables, there are $2^d$ possible input values for these variables. Any specific function assigns $Y = 1$ to some subset of these, and $Y = 0$ to the rest
- Any Boolean function can be trivially represented by a tree. Each function assigns $Y = 1$ to some subset of the $2^d$ possible values of $\mathbf{X}$. So, for each combination of values with $Y = 1$, have a path from root to a leaf with $Y = 1$. All other leaves have $Y = 0$

## When are Decision Trees the Right Model?

- This is a re-representation of the truth-table, and will have $2^d$ leaves.
- More compact trees may be possible by taking into account *what is common* between one or more rows with the same $Y$ value
- But there are some Boolean functions for which compact trees may not be possible (the parity and majority functions are examples)[1]
- In general, although possible in principle to express any Boolean function, search and prior restrictions may not allow us to find the correct tree in practice
- BUT: If you want readable models that combine logical tests with a probability-based decision, then decision trees are a good choice

[1] Finding the optimal tree is NP-complete (Hyafil and Rivest (1976)).

# Top-Down Induction of Decision Trees (TDIDT)

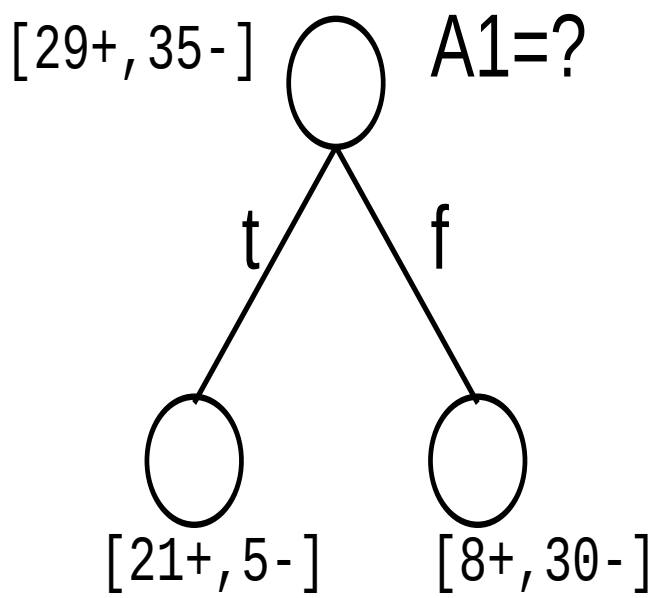
Main loop:

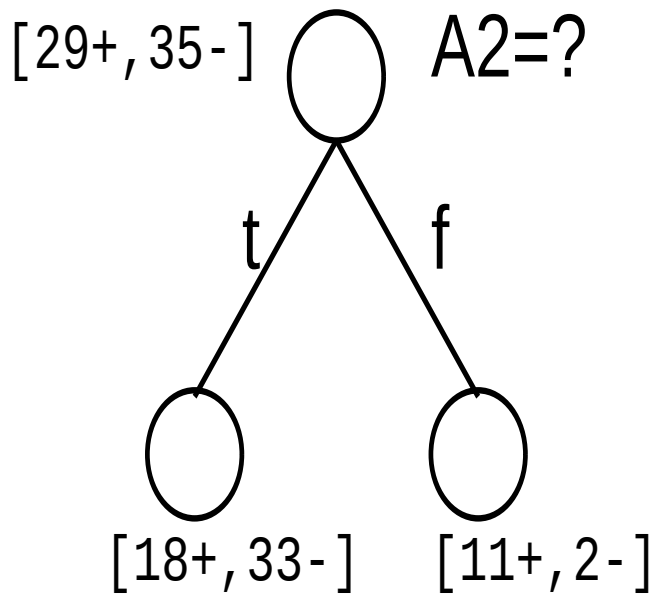
1. $A \leftarrow$ the "best" decision attribute for next $node$
2. Assign $A$ as decision attribute for $node$
3. For each value of $A$, create new descendant of $node$
4. Sort training examples to leaf nodes
5. If training examples perfectly classified, Then STOP, Else iterate over new leaf nodes

Essentially, this is the top-level of the ID3 algorithm[2] — the first efficient symbolic Machine Learning algorithm.

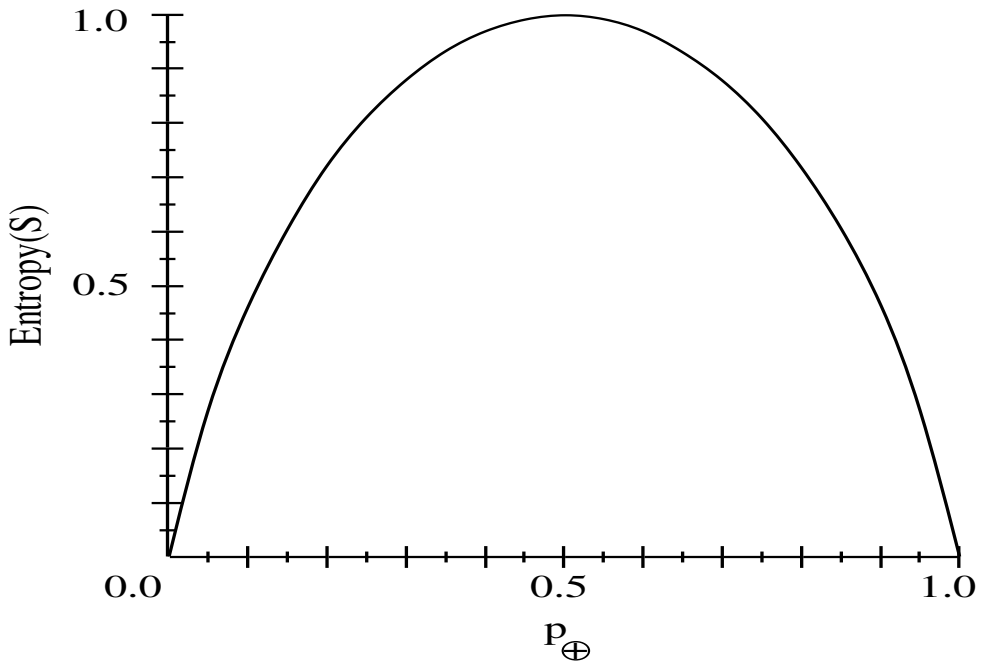
[2] See: Quinlan (1986).

# Which attribute is best?



# Entropy




Consider a 2-class distribution, where:

$S$ is a sample of training examples
$p_\oplus$ is the proportion of positive examples in $S$
$p_\ominus$ is the proportion of negative examples in $S$

## Entropy

Entropy measures the "impurity" of $S$

$$Entropy(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

A "pure" sample is one in which all examples are of the same class.

A decision tree node with low impurity means that the path from root to the node represents a combination of attribute-value tests with good classification accuracy.

## Entropy

$Entropy(S)$ = expected number of bits needed to encode class ($\oplus$ or $\ominus$) of randomly drawn member of $S$ (under the optimal, shortest-length code)

Why ?

Information theory: optimal length code assigns $-\log_2 p$ bits to message having probability $p$.

So, expected number of bits to encode $\oplus$ or $\ominus$ of random member of $S$:

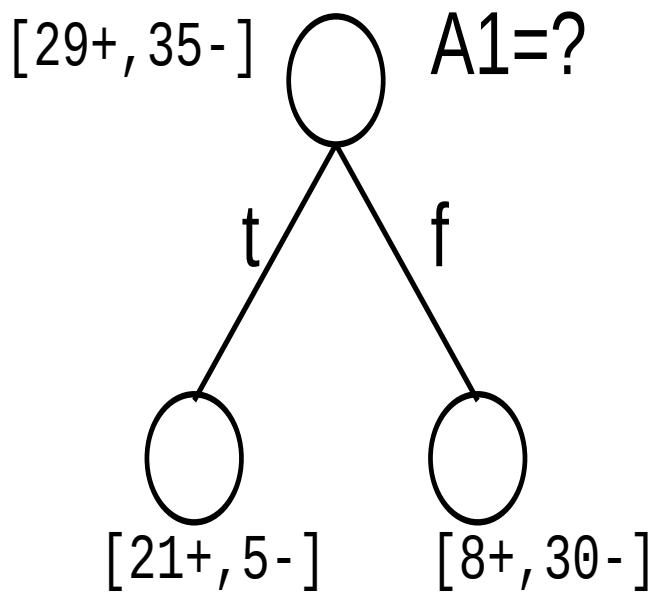
$$p_\oplus(-\log_2 p_\oplus) + p_\ominus(-\log_2 p_\ominus)$$

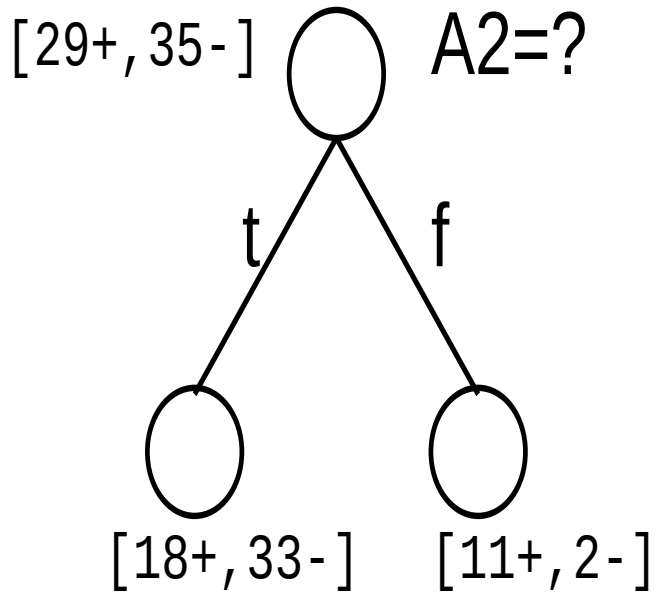
$$Entropy(S) \equiv -p_\oplus \log_2 p_\oplus - p_\ominus \log_2 p_\ominus$$

# Information Gain

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

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



# Information Gain

$$
\begin{aligned}
Gain(S, A1) &= Entropy(S) - \left( \frac{|S_t|}{|S|} Entropy(S_t) + \frac{|S_f|}{|S|} Entropy(S_f) \right) \\
&= 0.9936 - \\
&= ((\frac{26}{64}(-\frac{21}{26}\log_2(\frac{21}{26}) - \frac{5}{26}\log_2(\frac{5}{26}))) + \\
&\quad (\frac{38}{64}(-\frac{8}{38}\log_2(\frac{8}{38}) - \frac{30}{38}\log_2(\frac{30}{38})))) \\
&= 0.9936 - ( 0.2869 + 0.4408 ) \\
&= 0.2658
\end{aligned}
$$

# Information Gain

$$
\begin{aligned}
Gain(S, A2) &= 0.9936 - (\ 0.7464 + 0.0828\ ) \\
&= 0.1643
\end{aligned}
$$

## Information Gain

So we choose A1, since it gives a larger expected reduction in entropy.

# Attribute selection – impurity measures more generally

Estimate class probability of class $k$ at node $m$ of the tree as $\hat{p}_{mk} = \frac{|S_{mk}|}{|S_m|}$.
Classify at node $m$ by predicting the majority class, $\hat{p}_{mk}(m)$.

Misclassification error:

$$1 - \hat{p}_{mk}(m)$$

Entropy for $K$ class values:

$$-\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

CART (Breiman et al. (1984)) uses the "Gini index":

$$\sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

## Attribute selection – impurity measures more generally

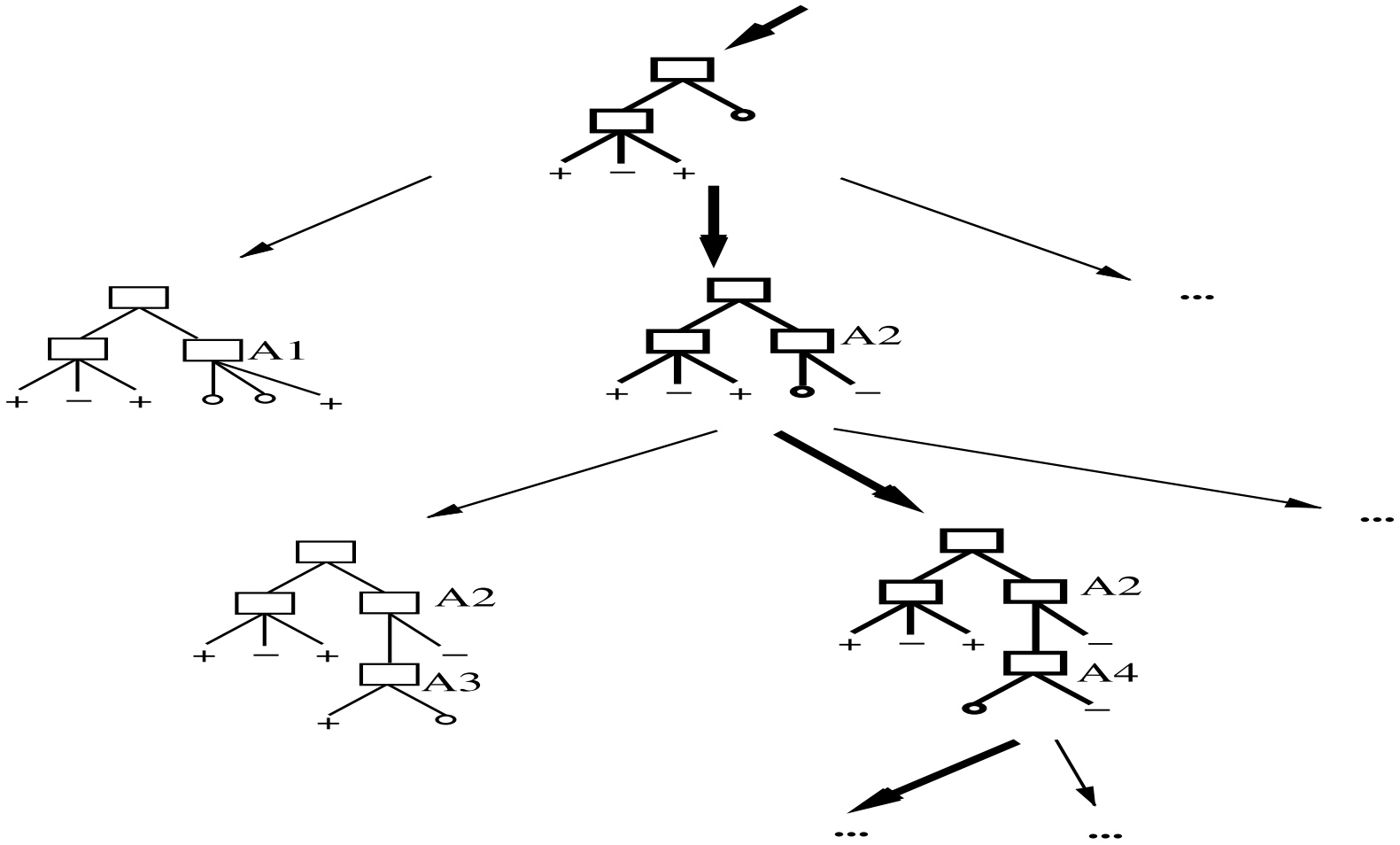
Why not just use accuracy, or misclassification error ?

In practice, not found to work as well as others

Entropy and Gini index are more sensitive to changes in the node probabilities than misclassification error.

Entropy and Gini index are differentiable, but misclassification error is not (Hastie et al. (2009)).

# Hypothesis Space Search by TDIDT




A greedy search to maximise information gain ...

# Inductive Bias of TDIDT

Note hypothesis space $H$ is complete (contains all finite discrete-valued functions w.r.t attributes)

- So $H$ can represent the power set of instances $X$ !

$\rightarrow$Unbiased?

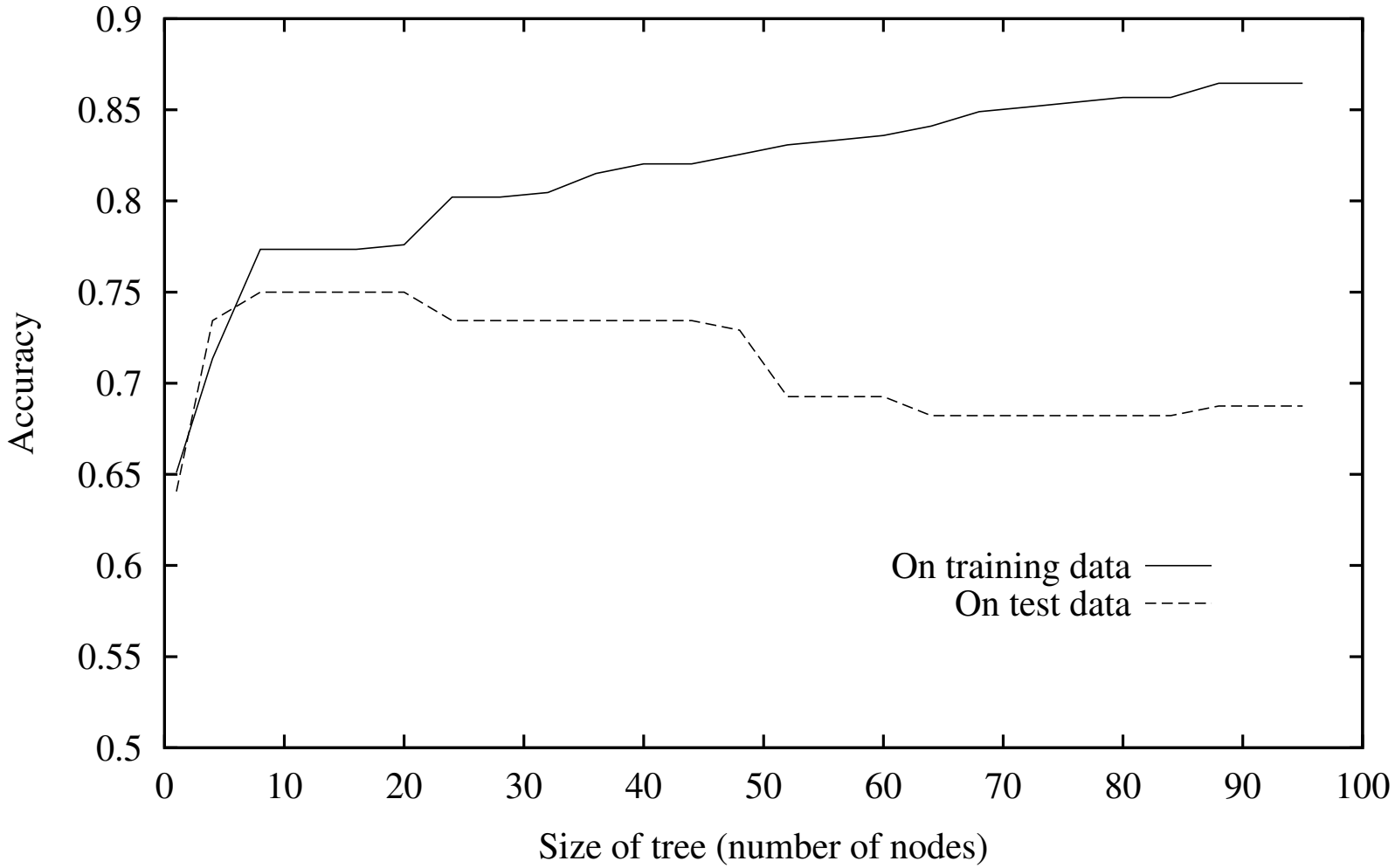
Not really...

- Preference for short trees, and for those with high information gain attributes near the root
- Inductive bias is a *preference* for some hypotheses, rather than a *restriction* of hypothesis space $H$
- An incomplete search of a complete hypothesis space *versus* a complete search of an incomplete hypothesis space
- Occam's razor: prefer the shortest hypothesis that fits the data
- Inductive bias: approximately, "prefer shortest tree"

# Why does overfitting occur?

- Greedy search can make mistakes. It can end up in local minima — so a sub-optimal choice earlier might result in a better solution later (*i.e.*, pick a test whose information gain is less than the best one)
- But there is also another kind of problem. Training error is an optimistic estimate of the true error of the model, and this optimism increases as the training error decreases
  - Suppose we could quantify the "optimism" of a learning algorithm ...
  - Say we have two models $h_1$ and $h_2$ with training errors $e_1$ and $e_2$ and optimism $o_1$ and $o_2$.
  - Let the true error of each be $E_1 = e_1 + o_1$ and $E_2 = e_2 + o_2$
  - If $e_1 < e_2$ and $E_1 > E_2$, then we will say that $h_1$ has overfit then training data
- So, a search method based purely on training data estimates may end up overfitting the training data

# Overfitting in Decision Tree Learning



# Avoiding Overfitting

How can we avoid overfitting?

For tree learning the answer is **pruning**

Two main approaches

- **pre-pruning** stop growing when further data splits are not useful
- **post-pruning** grow full tree, then remove sub-trees which may be overfitting

## Avoiding Overfitting – Pre-pruning

- Can be based on statistical significance test
- Stops growing the tree when there is no statistically significant association between any attribute and the class at a particular node
- For example, ID3 used chi-squared test plus information gain
  - only statistically significant attributes were allowed to be selected by information gain procedure
- Problem — as tree grows:

→ typical sample size at nodes get smaller

→ statistical tests become unreliable

## Avoiding Overfitting – Pre-pruning

- Simplest approach: stop growing the tree when fewer than some lower-bound on the number of examples at a leaf
- In `C5.0`[3], this parameter is the parameter `minCases = 2`
- In `sklearn`, this parameter is `min_samples_leaf`
- In `sklearn`, the parameter `min_impurity_decrease` enables stopping when the this falls below a lower-bound

[3] See: `www.rulequest.com`

## Avoiding Overfitting – Pre-pruning

- May cause early stopping: may stop the growth of tree prematurely
- Classic example: XOR/Parity-problem
  - No individual attribute exhibits a significant association with the class
  - Target structure only visible in fully expanded tree
  - Pre-pruning won't expand the root node
- But: XOR-type problems not common in practice
- And: pre-pruning faster than post-pruning
- Useful in exploratory data analysis
  - only grow top-level tree — all leaves have many examples

# Avoiding Overfitting – Post-pruning

- Builds full tree first and prunes it afterwards
- Why might this be a good idea ?
  - Attribute interactions are visible in fully-grown tree
- Problem: identification of subtrees and nodes that are due to chance effects
- Typical pruning operation:
  - Subtree replacement

## Avoiding Overfitting – Post-pruning

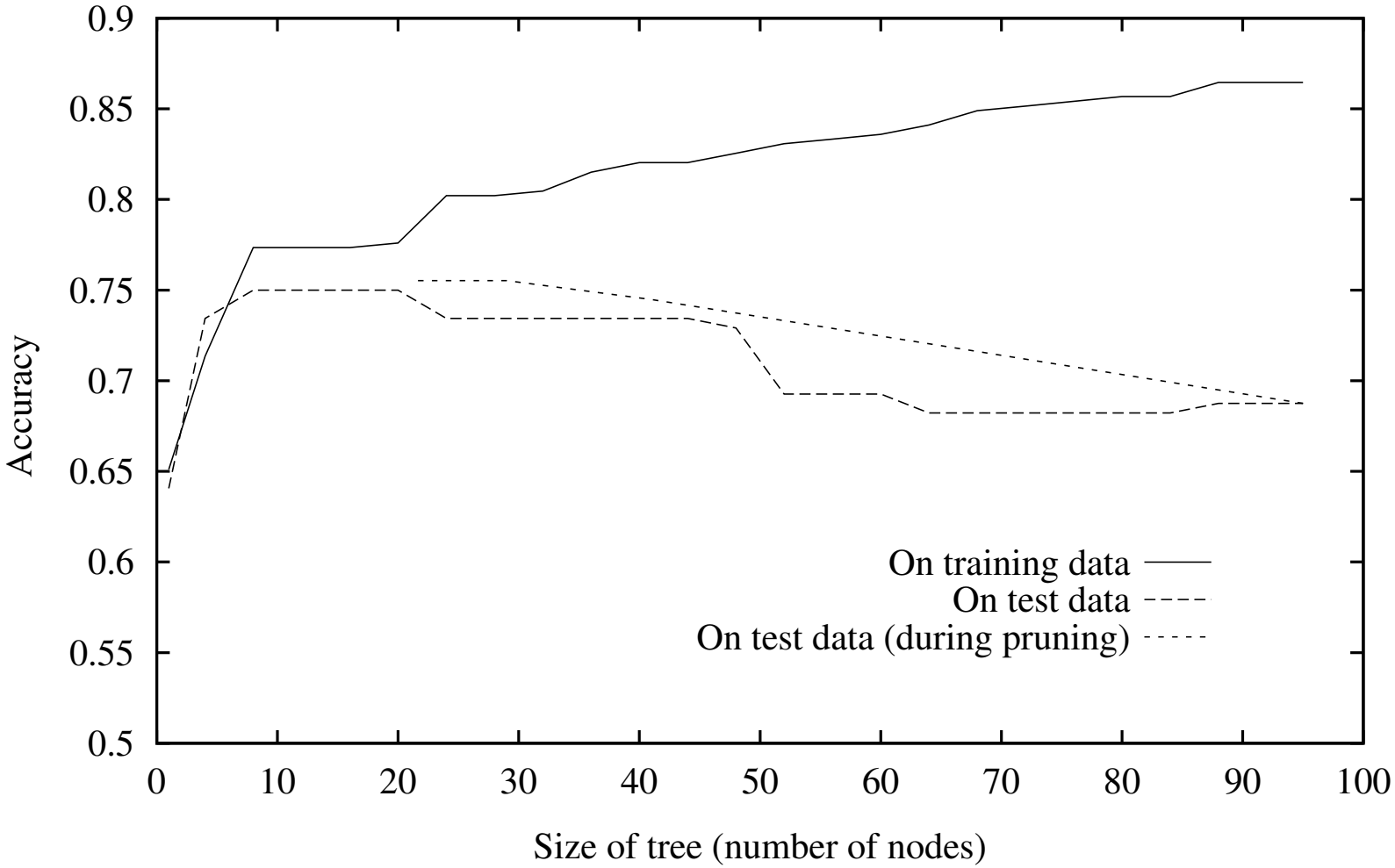
A simple greedy method (originally called "Reduced-Error Pruning")

Split data into $training$ and $validation$ set

Do until further pruning is harmful:

- Evaluate impact on $validation$ set of pruning each possible node (plus those below it)
- Remove the one that most improves $validation$ set accuracy

# Avoiding Overfitting – Post-pruning

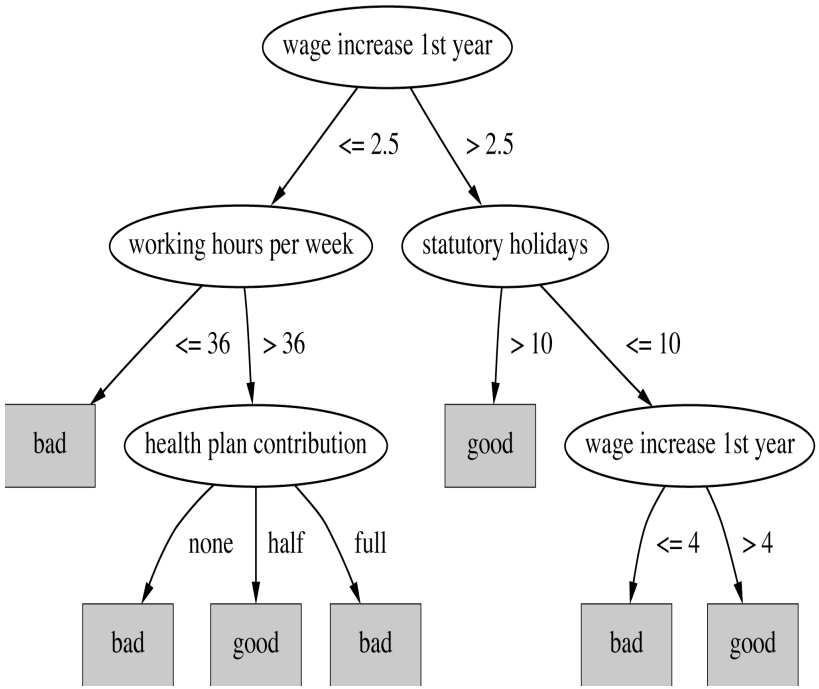


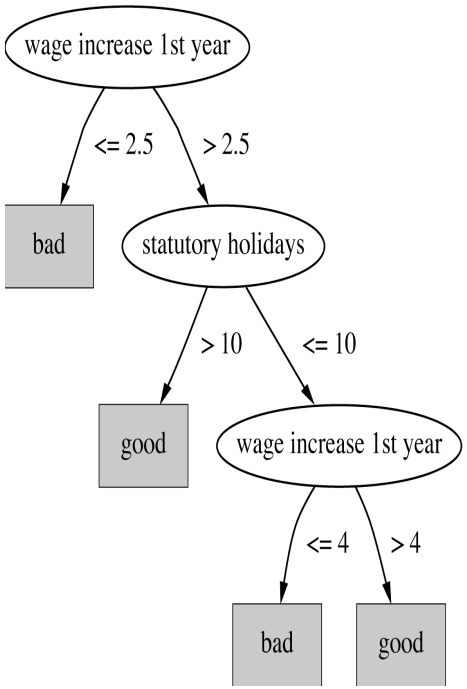

Effect of reduced-error pruning on tree learning to avoid overfitting.

# Avoiding Overfitting – Post-pruning

Pruning operator: Sub-tree replacement

Bottom-up:
tree is considered for replacement once all its sub-trees have been considered



## Avoiding Overfitting – Post-pruning

Problem with using validation set: reduces effective size of training data

Goal: to improve estimate of error on unseen data using all and only data from training set

But how can this work ?

Make the estimate of error **pessimistic** !

## Avoiding Overfitting – Post-pruning

- Apply pruning operation if this does not increase the estimated error
- C5.0's method: using upper limit of standard confidence interval derived from the training data (parameter called 'c' or 'CF')
- originally called "error-based pruning"
  - Standard Bernoulli-process-based method
  - **Note**: statistically motivated, but not statistically valid
  - **However**: works well in practice !

## Avoiding Overfitting – Post-pruning

- The error estimate for a tree node is the weighted sum of error estimates for all its subtrees (possibly leaves).
- Use upper bound error estimate $e$ for a node (simplified version):

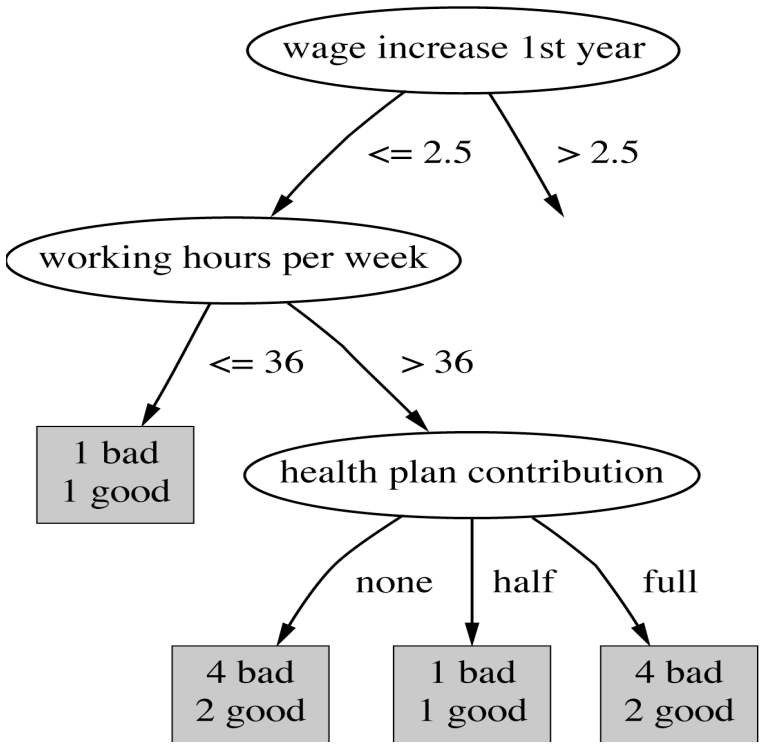
$$e = f + Z_c \cdot \sqrt{\frac{f \cdot (1 - f)}{N}}$$

- $f$ is actual (empirical) error of tree on examples at the tree node
- $N$ is the number of examples at the tree node
- $Z_c$ is a constant whose value depends on *confidence* parameter $c$
- Typical default value for confidence $c = 0.25$
- If $c = 0.25$ then $Z_c = 0.69$ (from standardized normal distribution)

## Avoiding Overfitting – Post-pruning

- How does this method implement a pessimistic error estimate ?
- What effect will the $c$ parameter have on pruning ?
- As $c \uparrow$, $z \downarrow$
- See example on next slide (note: values not calculated using exactly the above formula)

# Avoiding Overfitting – Post-pruning



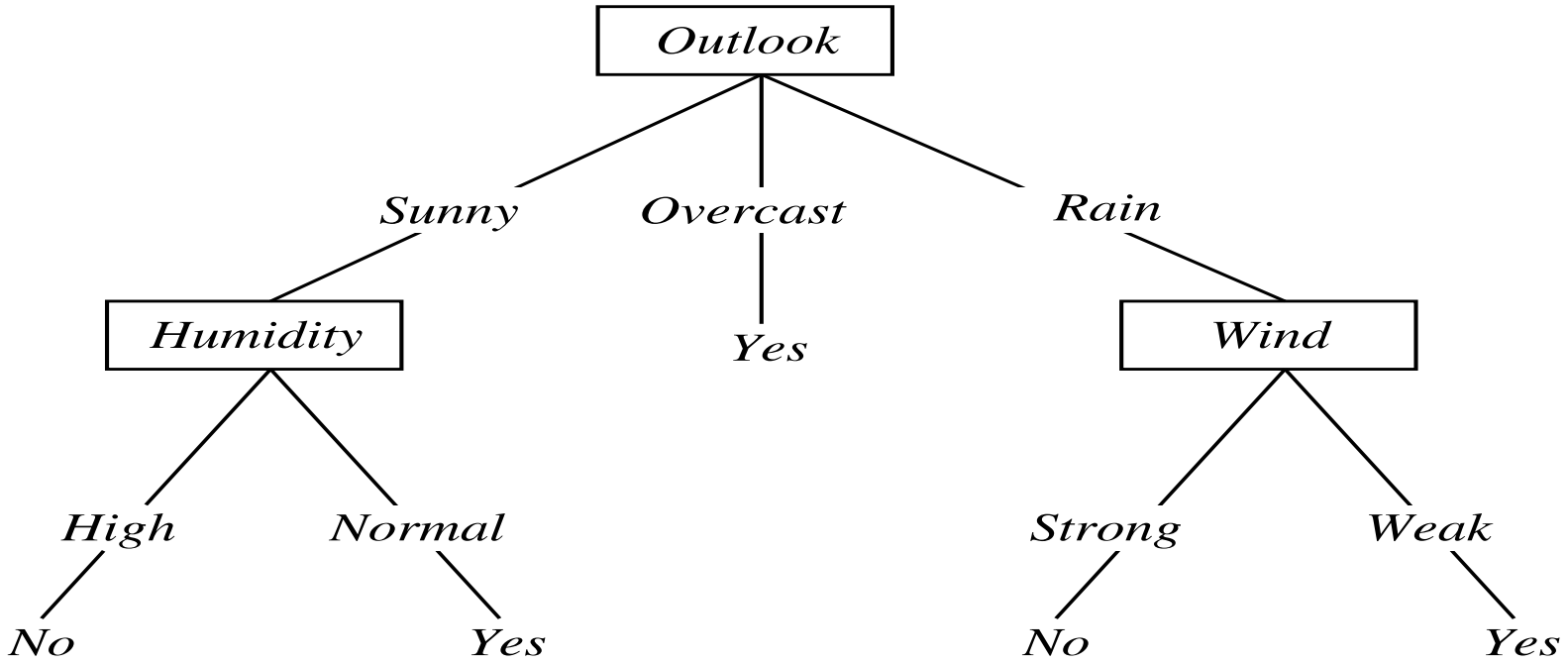

- health plan contribution: node measures $f = 0.36$, $e = 0.46$
- sub-tree measures:
  - none: $f = 0.33$, $e = 0.47$
  - half: $f = 0.5$, $e = 0.72$
  - full: $f = 0.33$, $e = 0.47$
- sub-trees combined $6 : 2 : 6$ gives $0.51$
- sub-trees estimated to give *greater* error so prune away

# Generating Rules from Trees

Rules can be more interpretable than trees, but just as accurate

- path from root to leaf in (unpruned) tree forms a *rule*
  - i.e., tree forms a *set of rules*
- can simplify rules independently by deleting conditions
  - i.e., rules can be generalized while maintaining accuracy
- A greedy rule simplification algorithm
  1. drop the condition giving lowest estimated error
  2. continue while estimated error does not increase

# Generating Rules from Trees




IF $(Outlook = Sunny) \wedge (Humidity = High)$
THEN $PlayTennis = No$

IF $(Outlook = Sunny) \wedge (Humidity = Normal)$
THEN $PlayTennis = Yes$

...

# Generating Rules from Trees

Rule "post-pruning" introduced by Quinlan[4]

- Convert tree to equivalent set of rules
- Find a subset of rules which minimises a regularised loss function
  - trade-off accuracy and complexity of rule-set
  - stochastic search using simulated annealing
- Sort final set of rules, ordered by class
  - order classes by increasing chance of making *false positive* errors
  - set as a default the class with the most training instances not covered by any rule

[4] See: Quinlan (1993).

# Generating Rules from Trees

Why do rule generation (and rule post-pruning?)

**For:** simpler classifiers, people prefer rules to trees

**For:** intrepretable models very important in many applications (medicine, finance, etc.)

**Against:** does not scale well, slow for large trees & datasets

# Numeric Attributes

Decision trees originated for **discrete** attributes only. However, data often has **numeric** attributes.

Can create a discrete attribute to test a numeric value:

- $Temperature = 82.5$
- $(Temperature > 72.3) \in \{t, f\}$
- Usual method: numeric attributes have a binary split
- Note:
  - discrete attributes – one split exhausts all values
  - numeric attributes – can have many splits in a tree

# Numeric Attributes

- Splits evaluated on all possible split points
- More computation: $n-1$ possible splits for $n$ values of an attribute in training set
- Fayyad (1991)
  - sort examples on numeric attribute
  - find midway boundaries where class changes, e.g. for *Temperature* $\frac{(48+60)}{2}$ and $\frac{(80+90)}{2}$
- Choose best split point by info gain (or evaluation of choice)
- Note: use actual values in data — more user-friendly

| *Temperature*: | 40 | 48 | 60 | 72 | 80 | 90 |
|---|---|---|---|---|---|---|
| *PlayTennis*: | No | No | Yes | Yes | Yes | No |

# Attributes with Many Values

Problem:

- If attribute has many values, $Gain$ will select it
- Why ? more likely to split instances into "pure" subsets
  - Maximised by singleton subsets
- Imagine using *Date* = June 21, 2021 as attribute
- High gain on training set, useless for prediction

## Attributes with Many Values

One approach: use $GainRatio$ instead

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$

$$SplitInformation(S, A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

where $S_i$ is subset of $S$ for which $A$ has value $v_i$

## Attributes with Many Values

Why does this help ?

- sensitive to how broadly and uniformly attribute splits instances
- actually the entropy of $S$ w.r.t. values of $A$
  - i.e., the information of the partition itself
- therefore higher for many-valued attributes, especially if mostly uniformly distributed across possible values
- a kind of normalisation

# Missing Values

What if some training set examples have missing values for attribute $A$?

Use example anyway, sort through tree. Here are 3 possible approaches

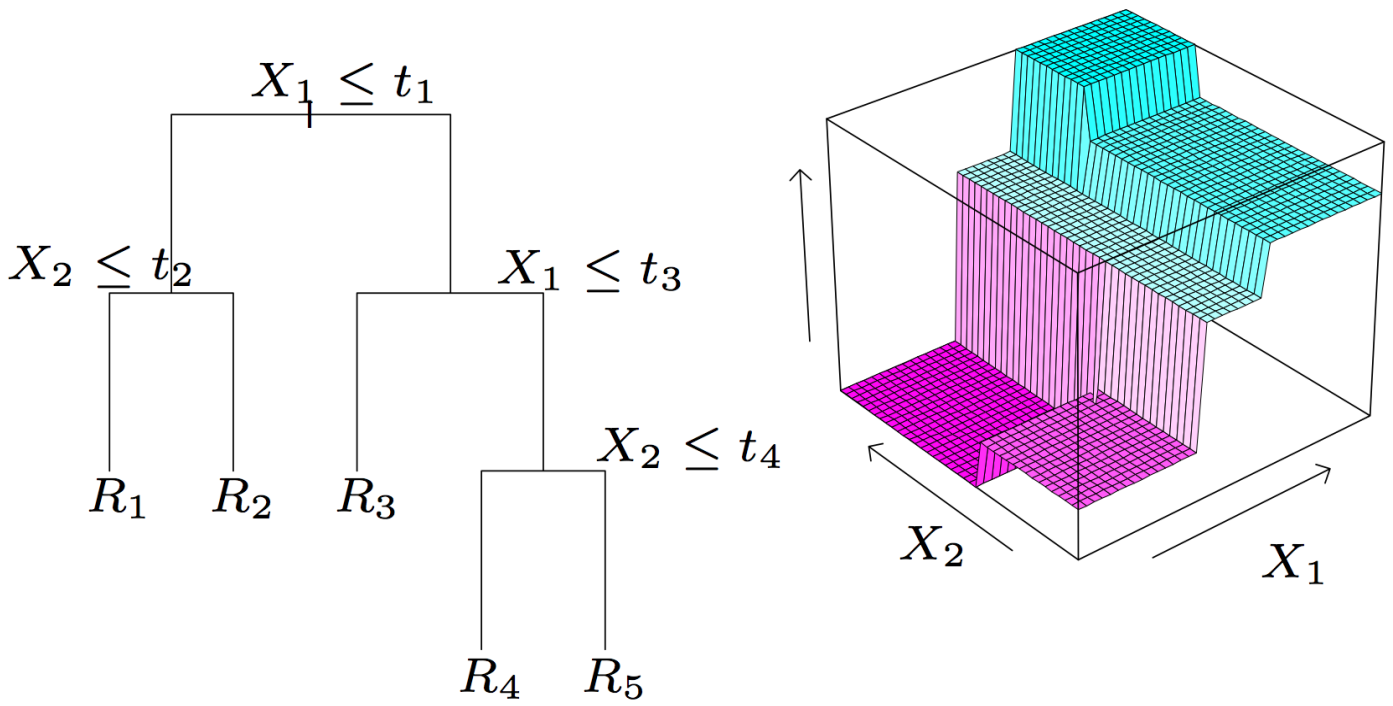
- If node $n$ tests $A$, assign most common value of $A$ among other examples sorted to node $n$
- assign most common value of $A$ among other examples with same target value
- assign probability $p_i$ to each possible value $v_i$ of $A$
  - assign fraction $p_i$ of example to each descendant in tree

Note: need to classify new (unseen) examples in the same way !

# Regression trees

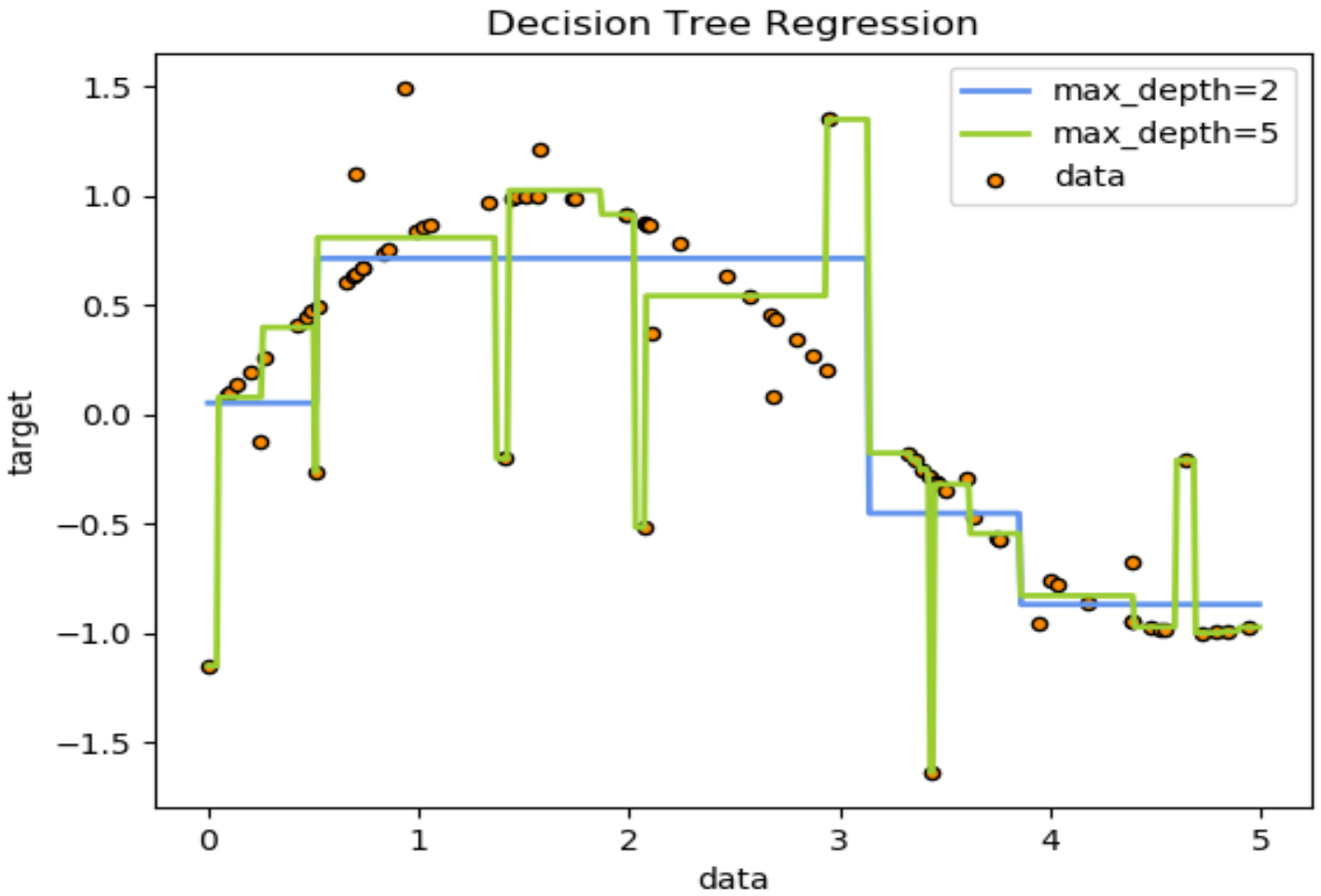
- Differences to decision trees:
  - Splitting criterion: minimizing intra-subset variation
  - Pruning criterion: based on numeric error measure
  - Leaf node predicts average class values of training instances reaching that node
- Can approximate piecewise constant functions
- Easy to interpret
- More sophisticated version: model trees

# A Regression Tree and its Prediction Surface

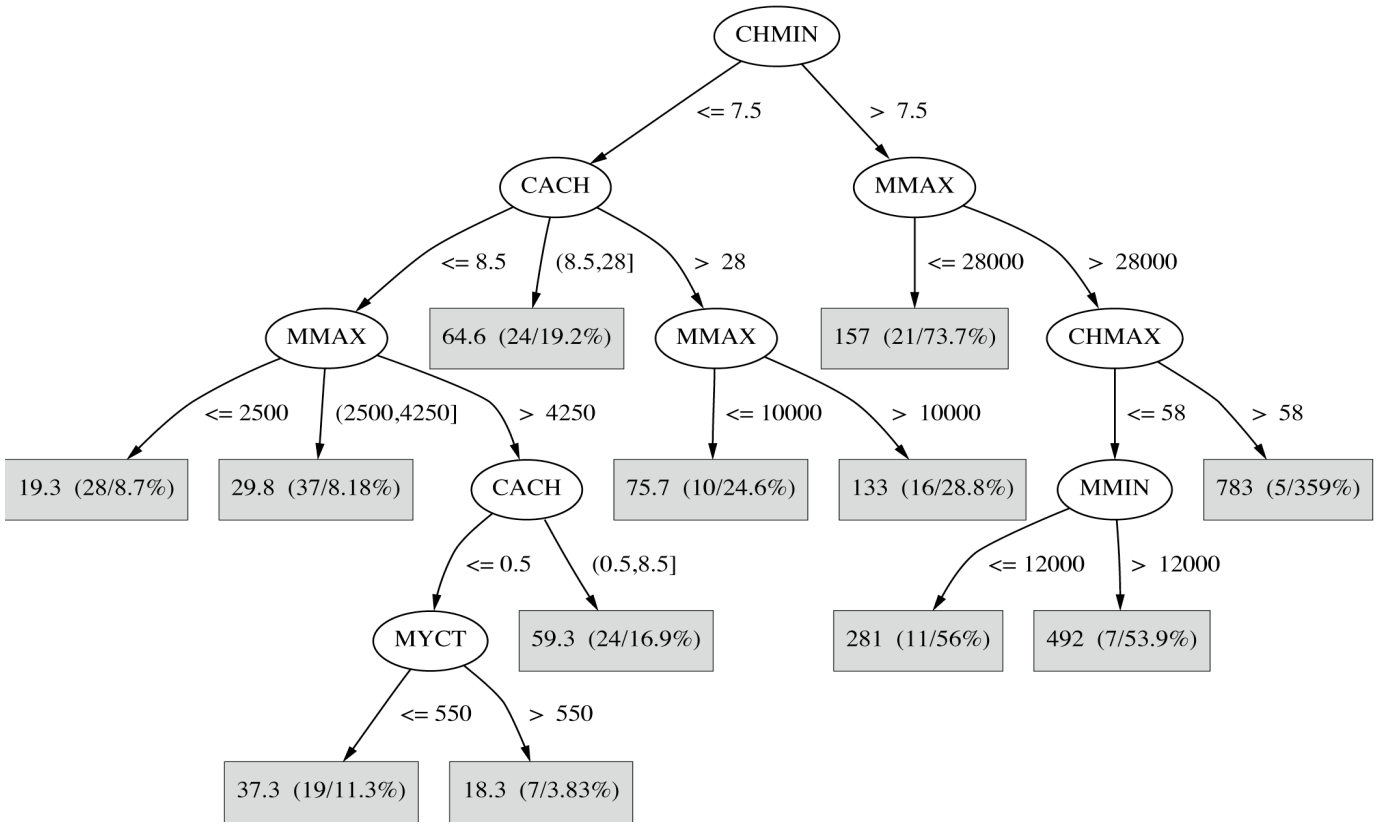



"Elements of Statistical Learning" Hastie, Tibshirani & Friedman (2001)

# Regression Tree on sine dataset



# Regression Tree on CPU dataset



# Tree learning as variance reduction

- Variance of a Boolean (i.e., Bernoulli) variable with success probability $p$ is $p(1-p)$.
- Can interpret goal of tree learning as minimising the class variance in the leaves.
- In regression problems we can define the variance in the usual way:

$$\mathrm{Var}(Y) = \frac{1}{|Y|}\sum_{y\in Y}(y-\overline{y})^2$$

If a split partitions the set of target values $Y$ into mutually exclusive sets $\{Y_1,\ldots,Y_l\}$, the weighted average variance is then

$$\mathrm{Var}(\{Y_1,\ldots,Y_l\}) = \sum_{j=1}^{l}\frac{|Y_j|}{|Y|}\mathrm{Var}(Y_j) = \ldots = \frac{1}{|Y|}\sum_{y\in Y}y^2 - \sum_{j=1}^{l}\frac{|Y_j|}{|Y|}\overline{y}_j^2$$

The first term is constant for a given set $Y$ and so we want to maximise the weighted average of squared means in the children.

# Learning a regression tree

Imagine you are a collector of vintage Hammond tonewheel organs. You have been monitoring an online auction site, from which you collected some data about interesting transactions:

| # | Model | Condition | Leslie | Price |
|---|---|---|---|---|
| 1. | B3 | excellent | no | 4513 |
| 2. | T202 | fair | yes | 625 |
| 3. | A100 | good | no | 1051 |
| 4. | T202 | good | no | 270 |
| 5. | M102 | good | yes | 870 |
| 6. | A100 | excellent | no | 1770 |
| 7. | T202 | fair | no | 99 |
| 8. | A100 | good | yes | 1900 |
| 9. | E112 | fair | no | 77 |

## Learning a regression tree

From this data, you want to construct a regression tree that will help you determine a reasonable price for your next purchase.
There are three features, hence three possible splits:

$$\text{Model} = [\text{A100}, \text{B3}, \text{E112}, \text{M102}, \text{T202}] \quad [1051, 1770, 1900][4513][77][870][99, 270, 625]$$

$$\text{Condition} = [\text{excellent}, \text{good}, \text{fair}] \quad [1770, 4513][270, 870, 1051, 1900][77, 99, 625]$$

$$\text{Leslie} = [\text{yes}, \text{no}] \quad [625, 870, 1900][77, 99, 270, 1051, 1770, 4513]$$

The means of the first split are 1574, 4513, 77, 870 and 331, and the weighted average of squared means is $3.21 \cdot 10^6$. The means of the second split are 3142, 1023 and 267, with weighted average of squared means $2.68 \cdot 10^6$; for the third split the means are 1132 and 1297, with weighted average of squared means $1.55 \cdot 10^6$. We therefore branch on Model at the top level. This gives us three single-instance leaves, as well as three A100s and three T202s.

## Learning a regression tree

For the A100s we obtain the following splits:

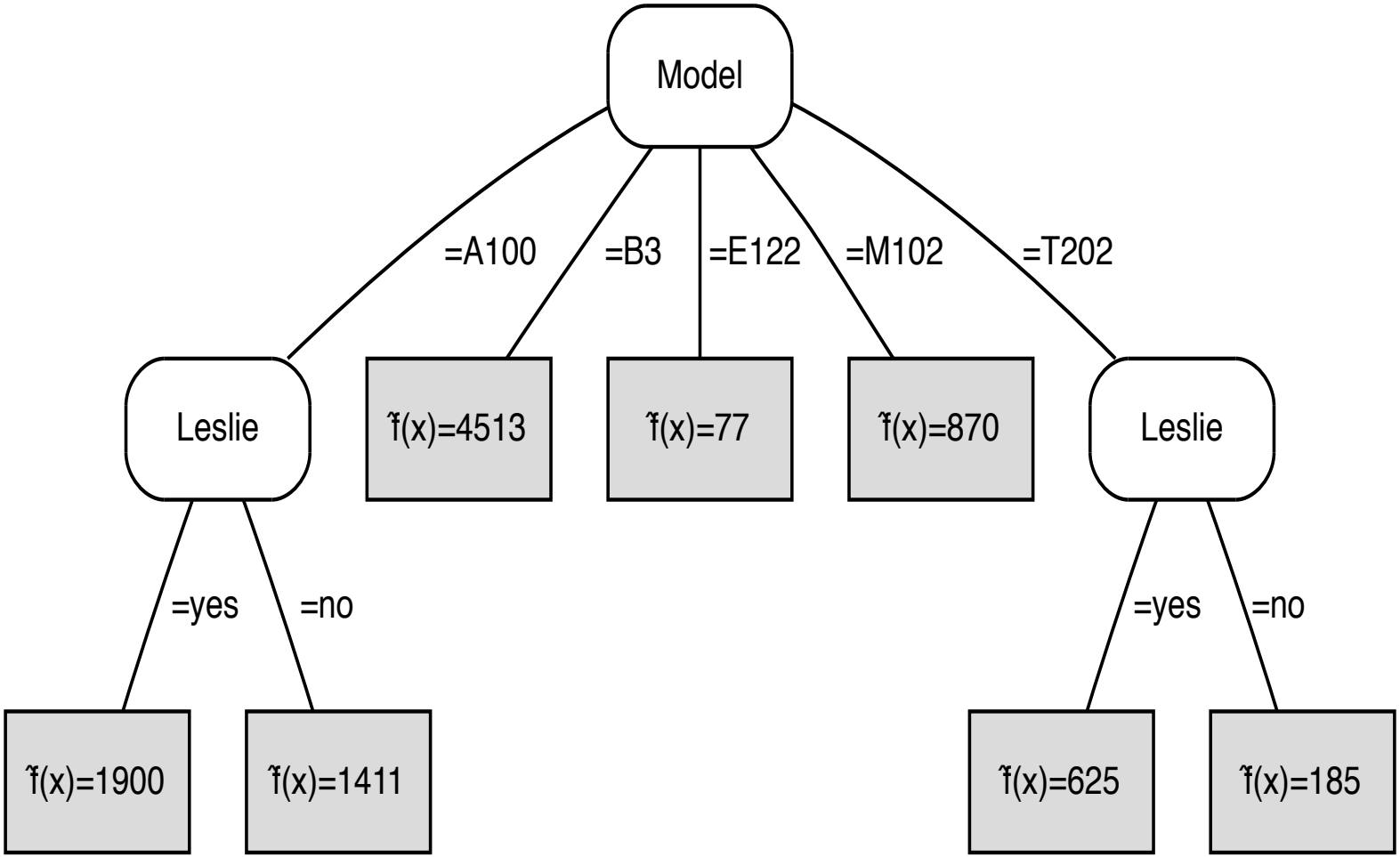
Condition = [excellent, good, fair] $[1770][1051, 1900][]$
Leslie = [yes, no] $[1900][1051, 1770]$

Without going through the calculations we can see that the second split results in less variance (to handle the empty child, it is customary to set its variance equal to that of the parent). For the T202s the splits are as follows:

Condition = [excellent, good, fair] $[][270][99, 625]$
Leslie = [yes, no] $[625][99, 270]$

Again we see that splitting on Leslie gives tighter clusters of values. The learned regression tree is depicted on the next slide.

# A regression tree




A regression tree learned from the Hammond organ dataset.

# Model trees

- Like regression trees but with linear regression functions at each node
- Linear regression applied to instances that reach a node after full tree has been built
- Only a subset of the attributes is used for LR
  - Attributes occurring in subtree (+maybe attributes occurring in path to the root)
- Fast: overhead for Linear Regression (LR) not large because usually only a small subset of attributes is used in tree

# Two uses of features (Flach (2012))

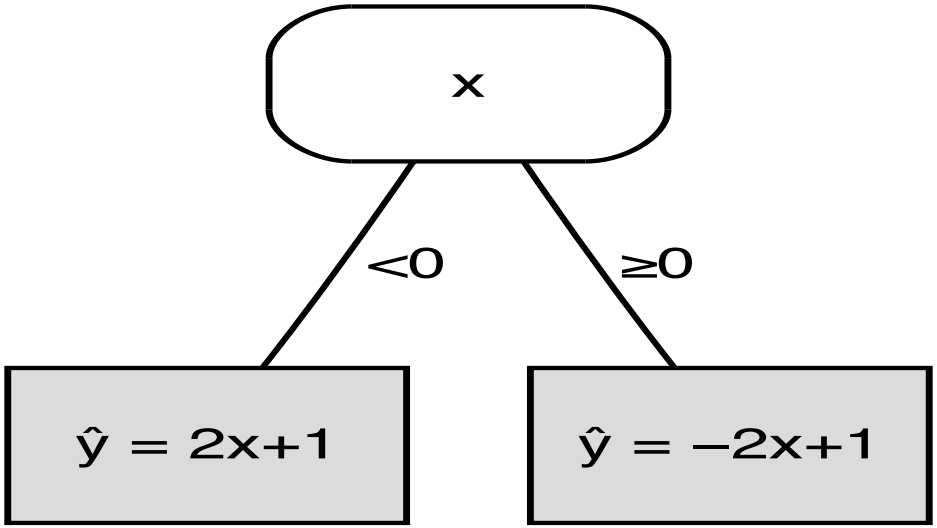
Suppose we want to approximate $y = \cos \pi x$ on the interval $-1 \leq x \leq 1$.

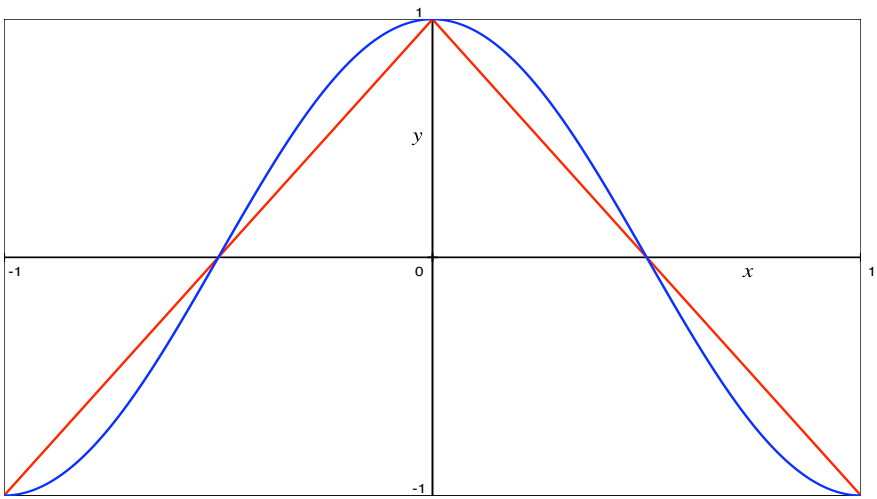
A linear approximation is not much use here, since the best fit would be $y = 0$.

However, if we split the $x$-axis in two intervals $-1 \leq x < 0$ and $0 \leq x \leq 1$, we could find reasonable linear approximations on each interval.

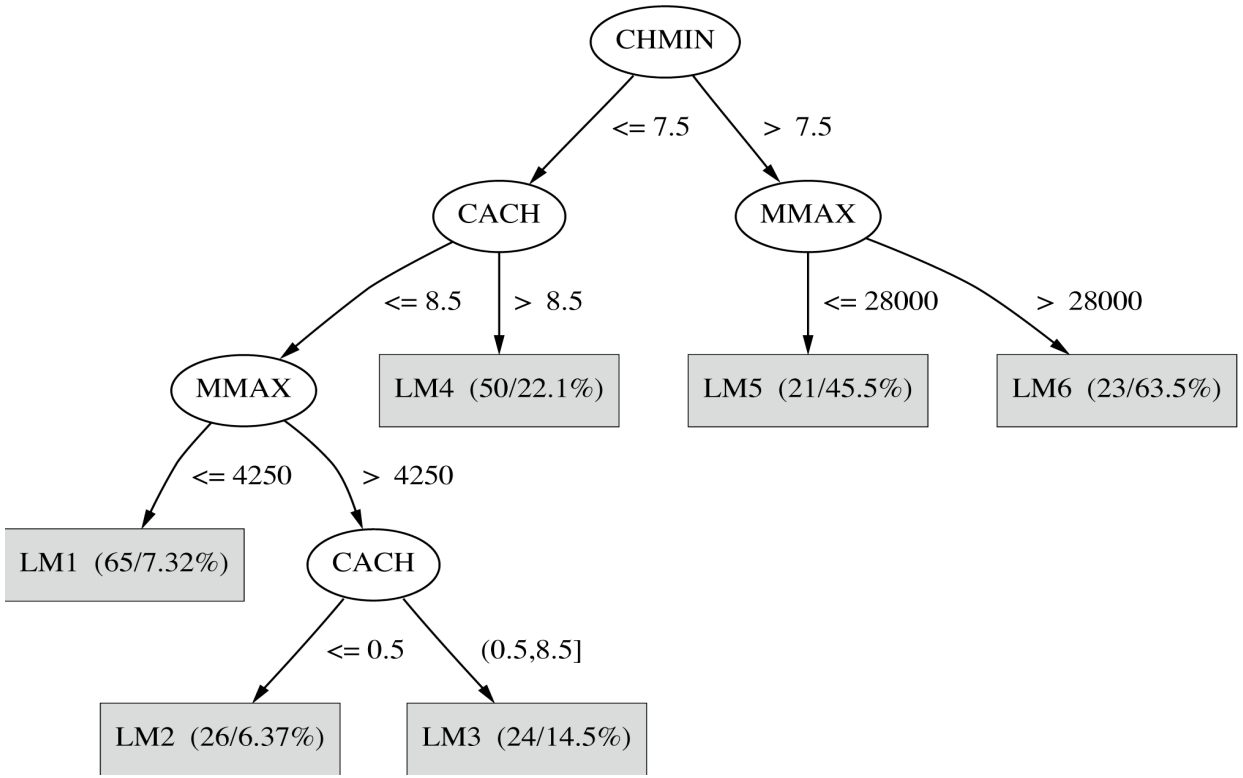
We can achieve this by using $x$ both as a splitting feature and as a regression variable (next slide).

# A small model tree



# Model Tree on CPU dataset



# Smoothing

- Naïve prediction method – output value of LR model at corresponding leaf node
- Improve performance by *smoothing* predictions with *internal* LR models
  - Predicted value is weighted average of LR models along path from root to leaf
- Smoothing formula: $p' = \frac{np+kq}{n+k}$ where
  - $p'$ prediction passed up to next higher node
  - $p$ prediction passed to this node from below
  - $q$ value predicted by model at this node
  - $n$ number of instances that reach node below
  - $k$ smoothing constant
- Same effect can be achieved by incorporating the internal models into the leaf nodes

# Pruning the tree

- Pruning is based on estimated absolute error of LR models
- Heuristic estimate:

$$\frac{n+v}{n-v} \times \text{average\_absolute\_error}$$

  where $n$ is number of training instances that reach the node, and $v$ is the number of parameters in the linear model
- LR models are pruned by greedily removing terms to minimize the estimated error
- Model trees allow for heavy pruning: often a single LR model can replace a whole subtree
- Pruning proceeds bottom up: error for LR model at internal node is compared to error for subtree

# Summary – decision trees

- Decision tree learning is a practical method for many classifier learning tasks – still a "Top 10" data mining algorithm – see `sklearn.tree.DecisionTreeClassifier`
- TDIDT family descended from ID3 searches complete hypothesis space - the hypothesis is there, somewhere...
- Uses a search or *preference* bias, search for optimal tree is, in general, not tractable
- Overfitting is inevitable with an expressive hypothesis space and noisy data, so pruning is important
- Decades of research into extensions and refinements of the general approach, e.g., for numerical prediction, logical trees
- Often the "try-first" machine learning method in applications, illustrates many general issues
- Performance can be improved with use of "ensemble" methods

# Summary – regression and model trees

- Regression trees were introduced in CART – R's implementation is close to CART, but see `sklearn.tree.DecisionTreeRegressor` for a basic version
- Quinlan proposed the M5 model tree inducer
- M5′: slightly improved version that is publicly available (M5Pin Weka is based on this)
- Quinlan also investigated combining instance-based learning with M5
- CUBIST: Quinlan's rule learner for numeric prediction `www.rulequest.com`
- Interesting comparison: Neural nets vs. model trees — both do *non-linear regression*
- other methods also can learn non-linear models

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth, Belmont.

Flach, P. (2012). *Machine Learning*. Cambridge University Press.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning*. Springer, 2nd edition.

Hyafil, L. and Rivest, R. (1976). Constructing Optimal Binary Decision Trees is NP-Complete. *Information Processing Letters*, 5(1):15–17.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA.