



Boosting

Machine Learning – CSEP546

Carlos Guestrin

University of Washington

February 3, 2014

©Carlos Guestrin 2005-2014

Fighting the bias-variance tradeoff

■ Simple (a.k.a. weak) learners are good

- e.g., naïve Bayes, logistic regression, decision stumps (or shallow decision trees) *thresholding*
- Low variance, don't usually overfit too badly

■ Simple (a.k.a. weak) learners are bad

- High bias, can't solve hard learning problems

■ Can we make weak learners always good???

- No!!!
- But often yes...

The Simplest Weak Learner: Thresholding, a.k.a. Decision Stumps

■ Learn: $h: \mathbf{X} \mapsto Y$

□ \mathbf{X} – features

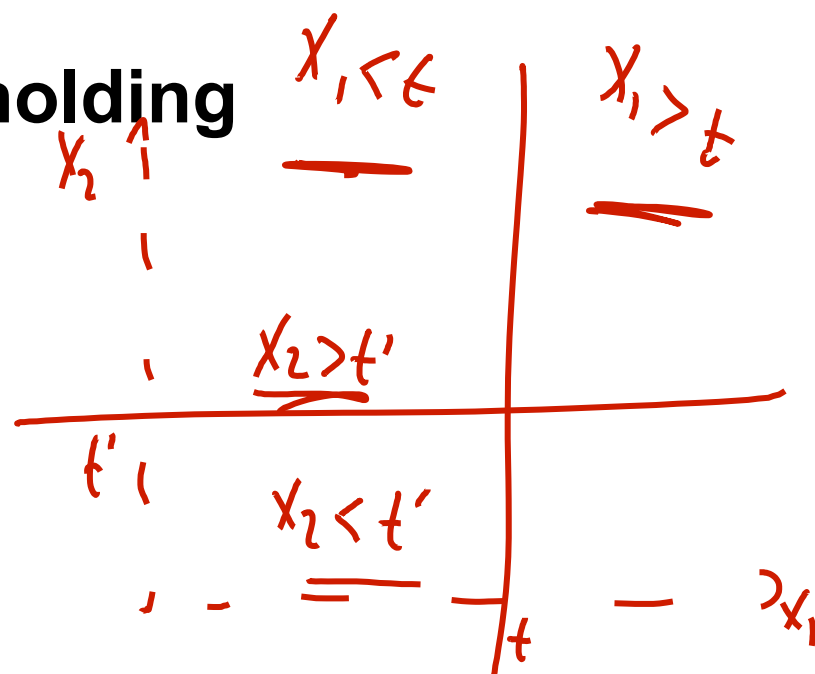
□ Y – target classes

$\mathbf{x} = (\text{GPA}, \text{grade}, \dots)$

$Y = \{\text{hired}, \text{not hired}\}$

■ Simplest case: Thresholding

$h(\mathbf{x}) = \begin{cases} \text{hired} & \text{if GPA} > 3.9 \\ \text{not hired} & \text{otherwise} \end{cases}$



Voting (Ensemble Methods)

- Instead of learning a single (weak) classifier, learn **many weak classifiers** that are **good at different parts of the input space**
- **Output class:** (Weighted) vote of each classifier

- Classifiers that are most “sure” will vote with more conviction
- Classifiers will be most “sure” about a particular part of the space
- On average, do better than single classifier!

$$H(x) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(x)\right)$$

classifier learned at iteration t

e.g. $h_t(x) = \text{GPA} > 3.9?$

the weight of classifier

■ But how do you ???

- force classifiers to learn about different parts of the input space?
- weigh the votes of different classifiers?

Boosting [Schapire, 1989]

- Idea: given a weak learner, run it multiple times on (reweighted) training data, then let learned classifiers vote

$$h_t(x) \rightarrow \{-1, +1\} = y$$

- On each iteration t :

- ☐ weight each training example by how incorrectly it was classified
- ☐ Learn a hypothesis h_t
- ☐ A strength for this hypothesis α_t

- Final classifier:

$$H(x) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(x)\right)$$

weigh up
hypotheses
examples

- **Practically useful**
- **Theoretically interesting**

Learning from weighted data

- Sometimes not all data points are equal

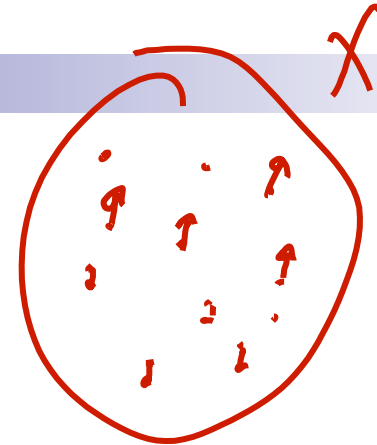
- Some data points are more equal than others

- Consider a weighted dataset

- $D(j)$ – weight of j th training example (\mathbf{x}^j, y^j)

- Interpretations:

- j th training example counts as $D(j)$ examples
- If I were to “resample” data, I would get more samples of “heavier” data points

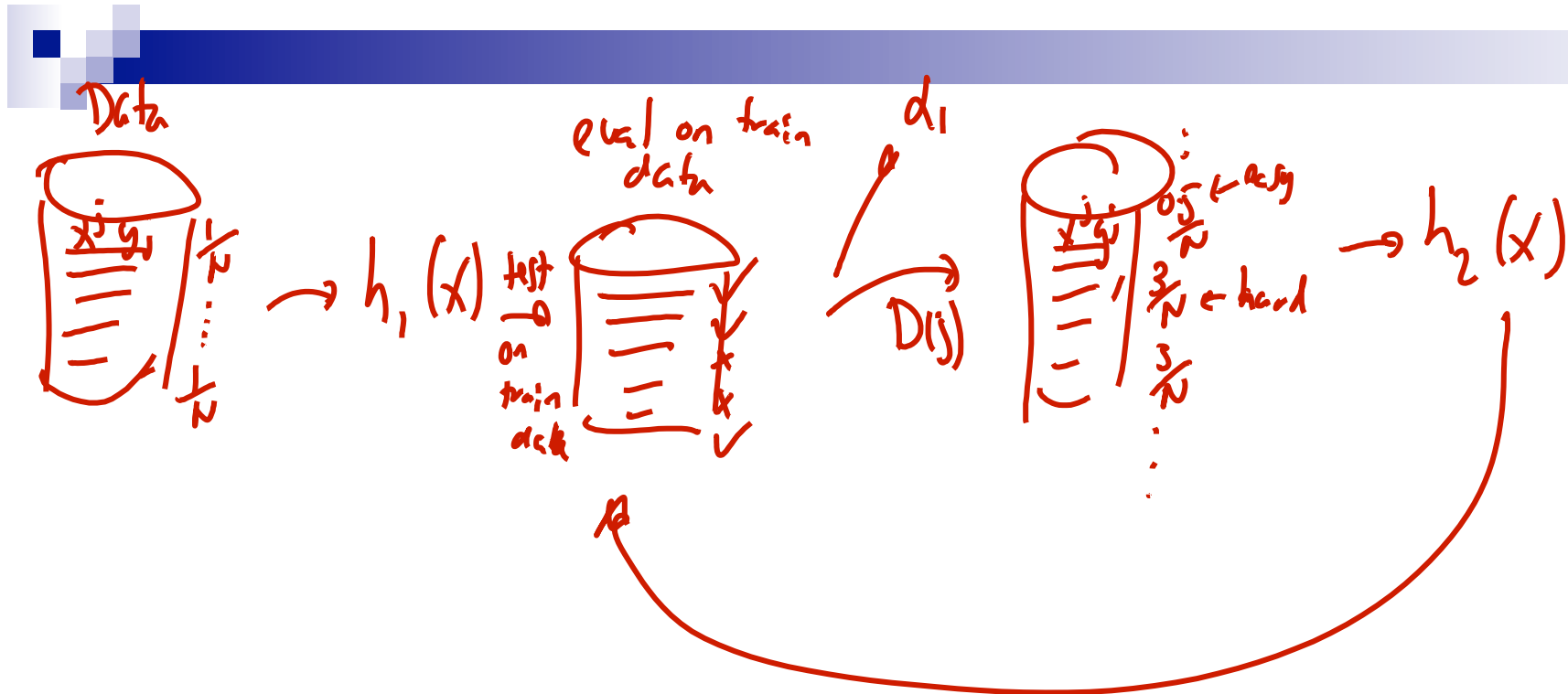


- Now, in all calculations, whenever used, j th training example counts as $D(j)$ “examples”

For example with approaches that use Gradient
Standard: $w \leftarrow w - \eta \sum_{j=1}^N \nabla_w f(\mathbf{x}^j)$

Weighted data: $w \leftarrow w - \eta \sum_{j=1}^N D(j) \nabla_w f(\mathbf{x}^j)$

Boosting Cartoon



AdaBoost

- Initialize weights to uniform dist: $D_1(j) = 1/N$
- For $t = 1 \dots T$

- Train weak learner h_t on distribution D_t over the data

- Choose weight α_t ← Magic, from next slide based on quality of h_t

- Update weights:

$$D_{t+1}(j) = \frac{D_t(j) \exp(-\alpha_t y^j h_t(x^j))}{Z_t}$$

- Where Z_t is normalizer:

$$Z_t = \sum_{j=1}^N D_t(j) \exp(-\alpha_t y^j h_t(x^j))$$

- Output final classifier:

$$H(x) = \text{Sign} \left(\sum_{t=1}^T \alpha_t h_t(x) \right)$$

uniform weights

✓ learned from weighted data

new weight

old weight $\exp(\alpha_t)$

if $y^j h_t(x^j) < 0$

\Rightarrow mistake on x^j

\Rightarrow weight $D(j)$ increases

exponentially based on α_t

similarly if not mistake weight decreases

make sure weights add up to 1

Picking Weight of Weak Learner

- Weigh h_t higher if it did well on training data (weighted by D_t):

Magic: $\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$

if $\epsilon_t = 1 \Rightarrow \alpha_t = -\infty$
 $\Rightarrow h_t$ is exactly wrong
 $\Rightarrow -h_t$ is exactly right

if $\epsilon_t = \frac{1}{2} \Rightarrow \alpha_t = 0$
 \hookrightarrow classifier is as good as random
 no point including it

if $\epsilon_t = 0 \Rightarrow \alpha_t = +\infty$
 \Rightarrow perfect classification on weighted data \Rightarrow perfect on all (unweighted) data

□ Where ϵ_t is the weighted training error:

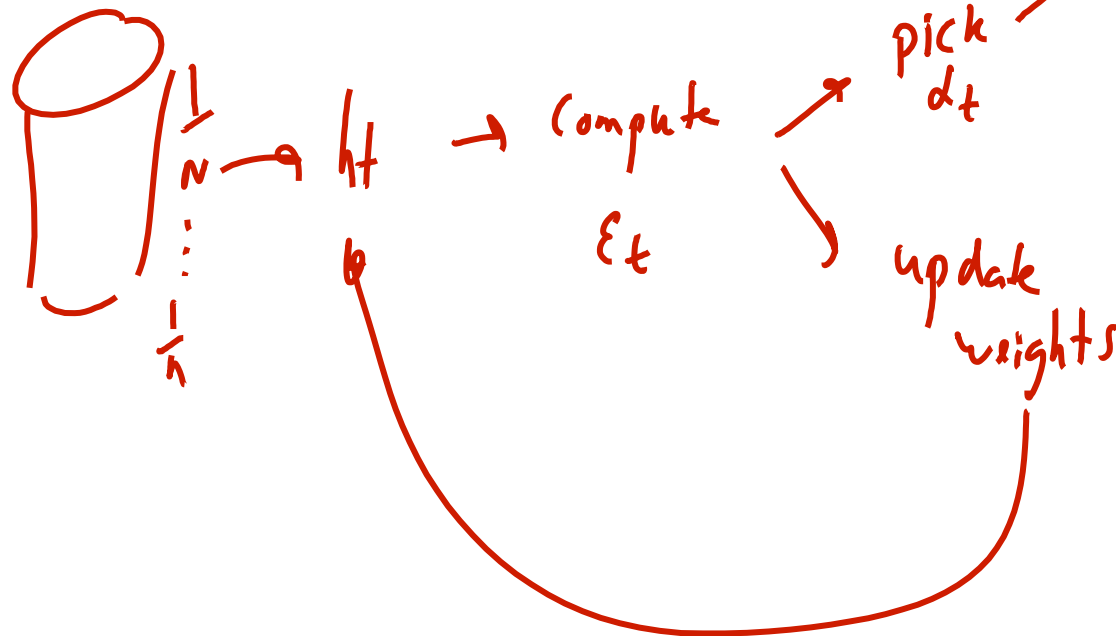
$$\epsilon_t = \sum_{j=1}^N D_t(j) \mathbb{1}[h_t(x^j) \neq y^j]$$

in practice
 $0 < \alpha_t < +\infty$

AdaBoost Cartoon

$$D_{t+1}(j) = \frac{D_t(j) \exp(-\alpha_t y^j h_t(x^j))}{Z_t}$$

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$



Why choose α_t for hypothesis h_t this way?

$$H(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(x) \right)$$

[Schapire, 1989]

$$Z_t = \sum_{j=1}^N D_t(j) \exp(-\alpha_t y^j h_t(x^j))$$

■ Simple theoretical analysis:

- Training error upper-bounded by product of normalizers

$$\frac{1}{N} \sum_{j=1}^N \mathbb{1}[H(x^j) \neq y^j] \leq \prod_{t=1}^T Z_t$$

train error

product of normalizers
over iterations

if $\forall t, Z_t < 1 \Rightarrow$ train error decreases with t
exponentially

- Pick α_t to minimize upper-bound

- Take derivative and set to zero!

with respect to α_t

Strong, weak classifiers

How do we guarantee that $Z_t < 1$

- If each classifier is (at least slightly) better than random

□ $\epsilon_t < 0.5$

$\exists \gamma_t > 0$ such that $\epsilon_t < 0.5 - \gamma_t$

- AdaBoost will achieve zero *training* error (exponentially fast):

$$\frac{1}{N} \sum_{j=1}^N \mathbb{1}[H(x^j) \neq y^j] \leq \prod_{t=1}^T Z_t \leq \exp\left(-2 \sum_{t=1}^T (1/2 - \epsilon_t)^2\right)$$

train error

always making progress

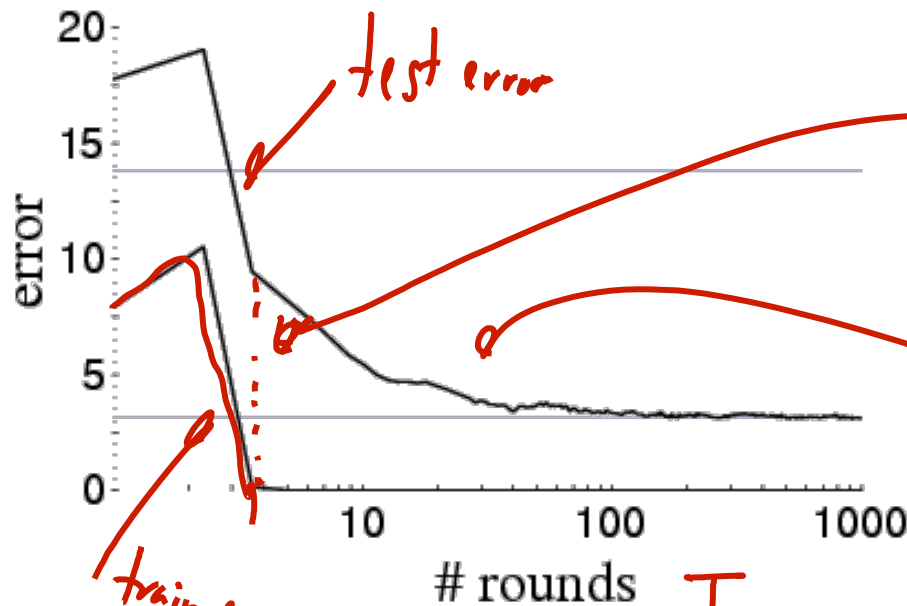
Easy in first iteration

with weighted data, you may not always win

- Is it hard to achieve better than random training error?

Boosting results – Digit recognition

[Schapire, 1989]



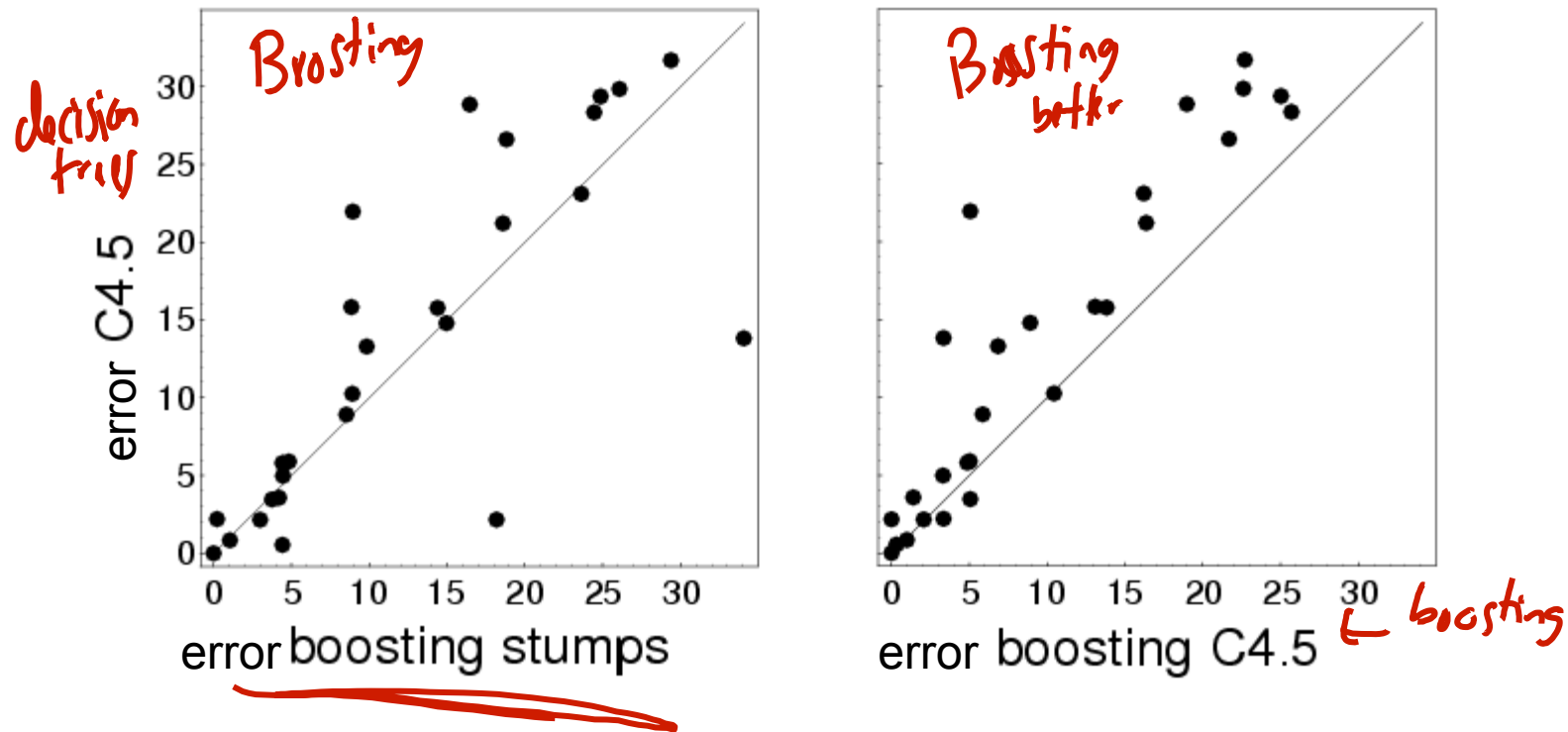
■ Boosting often

- ☐ Robust to overfitting ✓ Often
- ☐ Test set error decreases even after training error is zero

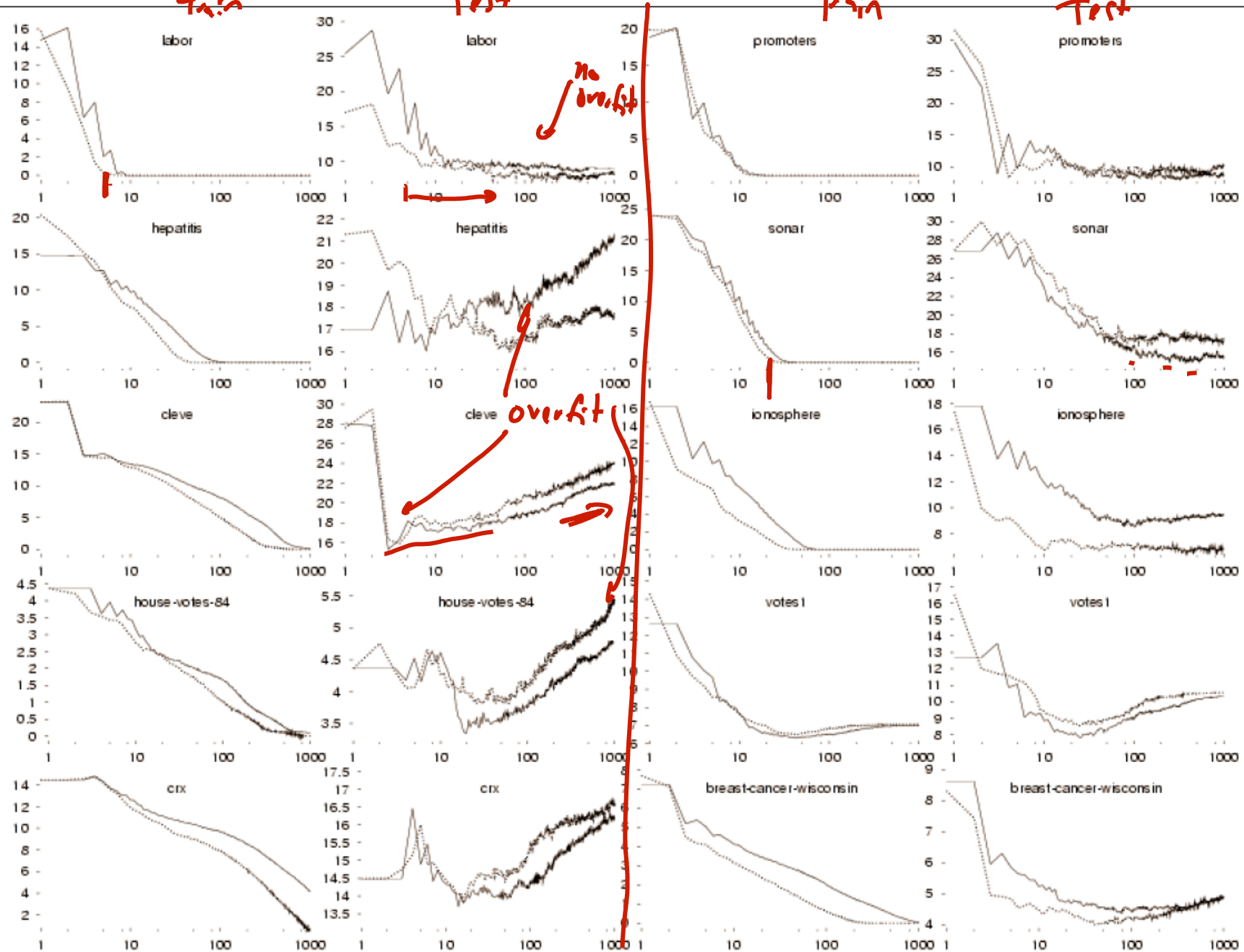
Boosting: Experimental Results

[Freund & Schapire, 1996]

Comparison of C4.5, Boosting C4.5, Boosting decision stumps (depth 1 trees), 27 benchmark datasets



AdaBoost and AdaBoost.MH on Train (left) and Test (right) data from Irvine repository. [Schapire and Singer, ML 1999]



What you need to know about Boosting

- Combine weak classifiers to obtain very strong classifier
 - Weak classifier – slightly better than random on training data
 - Resulting very strong classifier – can eventually provide zero training error
- AdaBoost algorithm
- Most popular application of Boosting:
 - Boosted decision stumps!
 - Very simple to implement, very effective classifier

Boosting prediction is $H(x) = \text{Sign} \left(\sum_{t=1}^T \alpha_t h_t(x) \right)$

learns a "linear" classifier in T dimensional space
"features" $h_t(x)$ are "discovered" from weighted data



Decision Trees

Machine Learning – CSEP546

Carlos Guestrin

University of Washington

February 3, 2014

©Carlos Guestrin 2005-2014

Linear separability

- A dataset is **linearly separable** iff there exists a **separating hyperplane**:

- Exists \mathbf{w} , such that:

- $w_0 + \sum_i w_i x_i > 0$; if $\mathbf{x}=\{x_1, \dots, x_k\}$ is a positive example
- $w_0 + \sum_i w_i x_i < 0$; if $\mathbf{x}=\{x_1, \dots, x_k\}$ is a negative example



Not linearly separable data

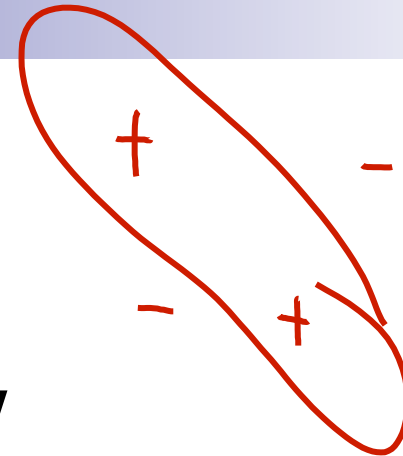
- Some datasets are **not linearly separable**!



$$x_1 \oplus x_2 \equiv x_1 \wedge \neg x_2 \vee \neg x_1 \wedge x_2$$

Addressing non-linearly separable data – Option 1, non-linear features

- Choose non-linear features, e.g.,
 - Typical linear features: $w_0 + \sum_i w_i x_i$
 - Example of non-linear features:
 - Degree 2 polynomials, $w_0 + \sum_i w_i x_i + \sum_{ij} w_{ij} x_i x_j$
- Classifier $h_{\mathbf{w}}(\mathbf{x})$ still linear in parameters \mathbf{w}
 - As easy to learn
 - Data is linearly separable in higher dimensional spaces



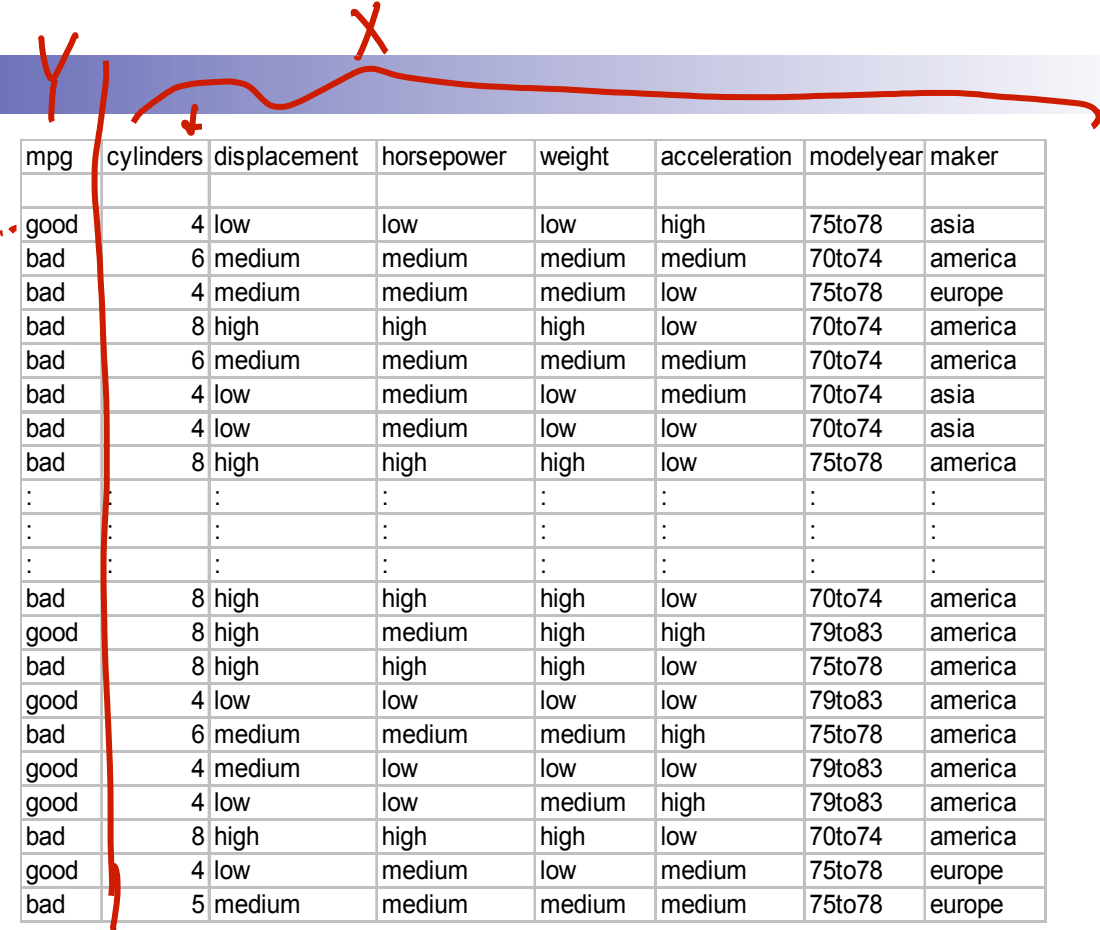
Addressing non-linearly separable data – Option 2, non-linear classifier

- Choose a classifier $h_{\mathbf{w}}(\mathbf{x})$ that is non-linear in parameters \mathbf{w} , e.g.,
 - Decision trees, boosting, nearest neighbor, neural networks...
- More general than linear classifiers
- But, can often be harder to learn (non-convex/concave optimization required)
- But, but, often very useful

A small dataset: Miles Per Gallon

Suppose we want
to predict MPG

$X \rightarrow Y : \text{MPG}$

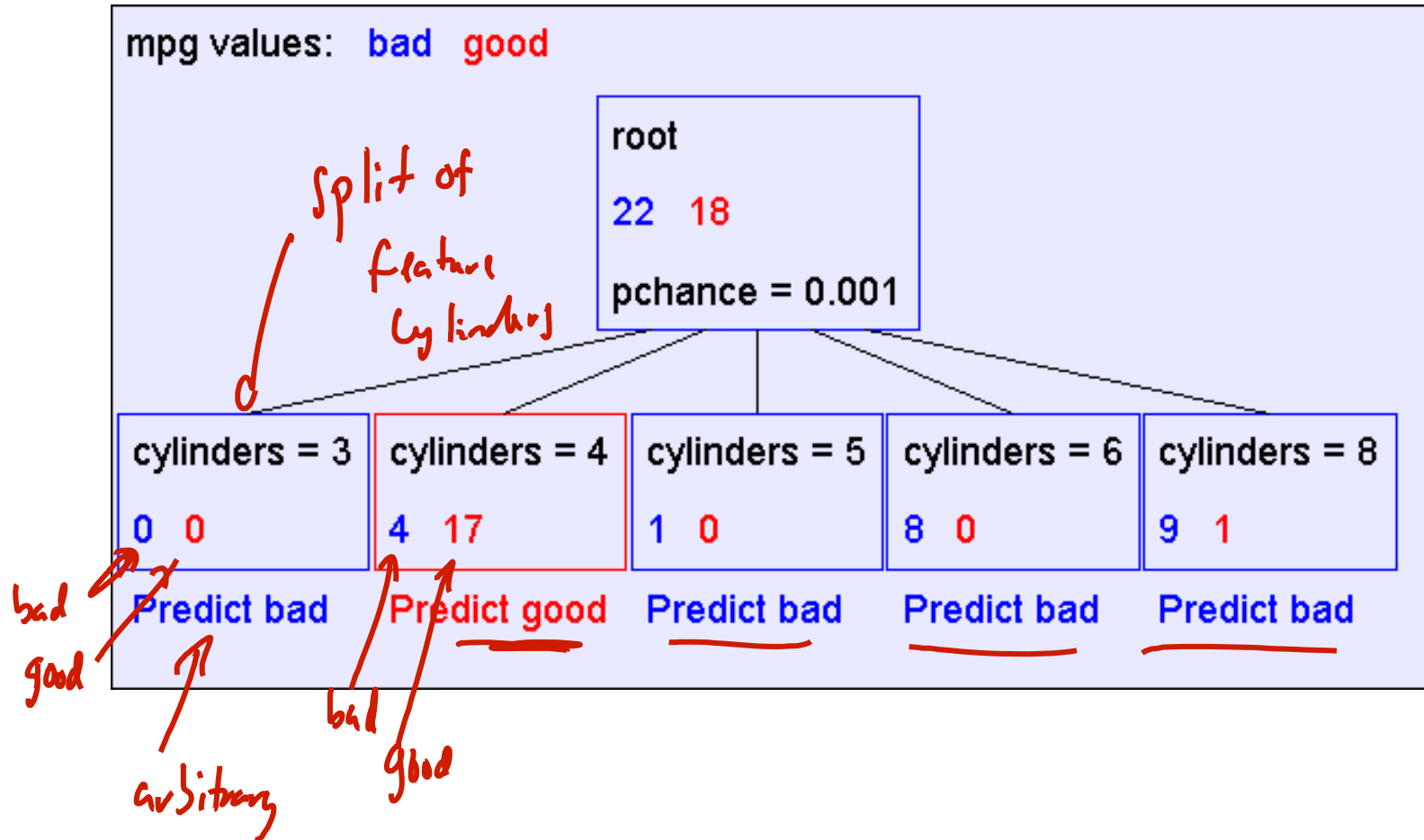


mpg	cylinders	displacement	horsepower	weight	acceleration	modelyear	maker
good	4	low	low	low	high	75to78	asia
bad	6	medium	medium	medium	medium	70to74	america
bad	4	medium	medium	medium	low	75to78	europa
bad	8	high	high	high	low	70to74	america
bad	6	medium	medium	medium	medium	70to74	america
bad	4	low	medium	low	medium	70to74	asia
bad	4	low	medium	low	low	70to74	asia
bad	8	high	high	high	low	75to78	america
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
bad	8	high	high	high	low	70to74	america
good	8	high	medium	high	high	79to83	america
bad	8	high	high	high	low	75to78	america
good	4	low	low	low	low	79to83	america
bad	6	medium	medium	medium	high	75to78	america
good	4	medium	low	low	low	79to83	america
good	4	low	low	medium	high	79to83	america
bad	8	high	high	high	low	70to74	america
good	4	low	medium	low	medium	75to78	europa
bad	5	medium	medium	medium	medium	75to78	europa

40 training
examples

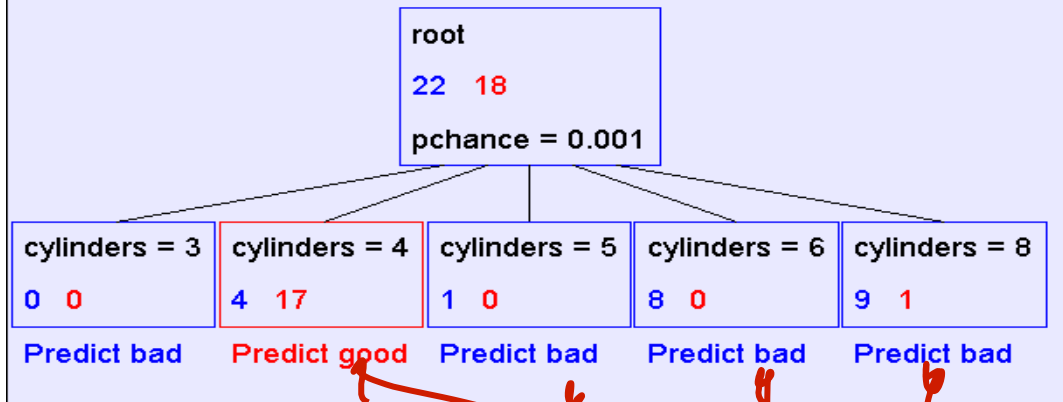
From the UCI repository (thanks to Ross Quinlan)

A Decision Stump



Recursion Step

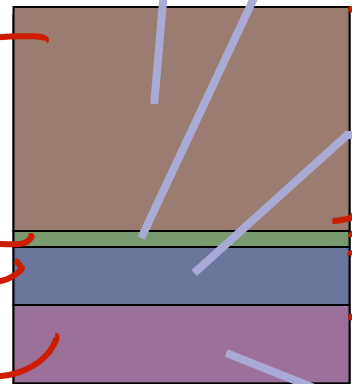
mpg values: bad good



Take the
Original
Dataset..



And partition it
according
to the value of
the attribute we
split on



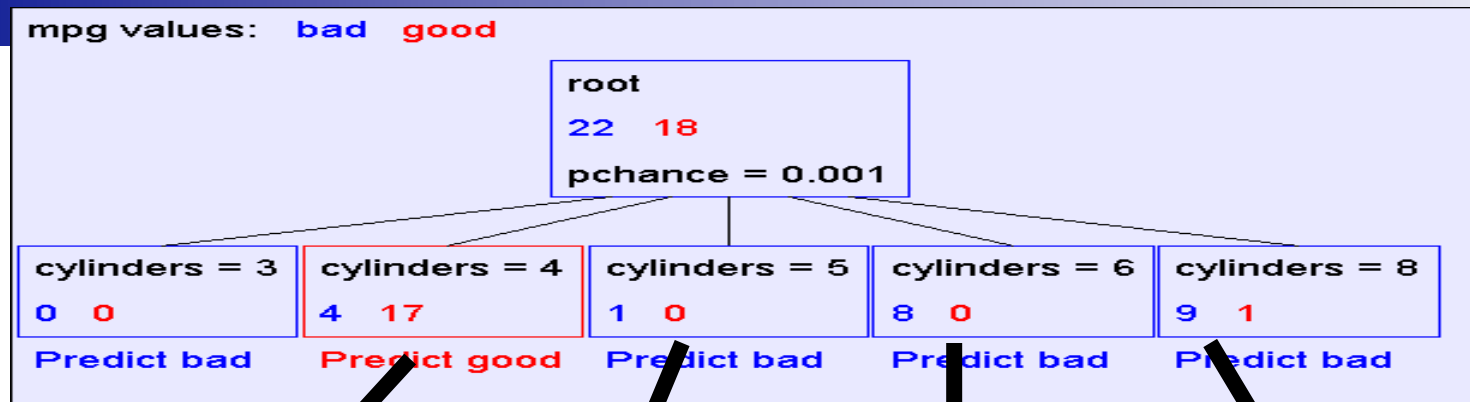
Examples
in which
cylinders
= 4

Examples
in which
cylinders
= 5

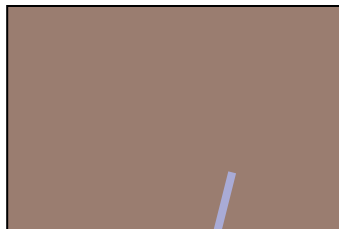
Examples
in which
cylinders
= 6

Examples
in which
cylinders
= 8

Recursion Step



Build tree from
These examples..



Records in
which cylinders
= 4

Build tree from
These examples..



Records in
which cylinders
= 5

Build tree from
These examples..



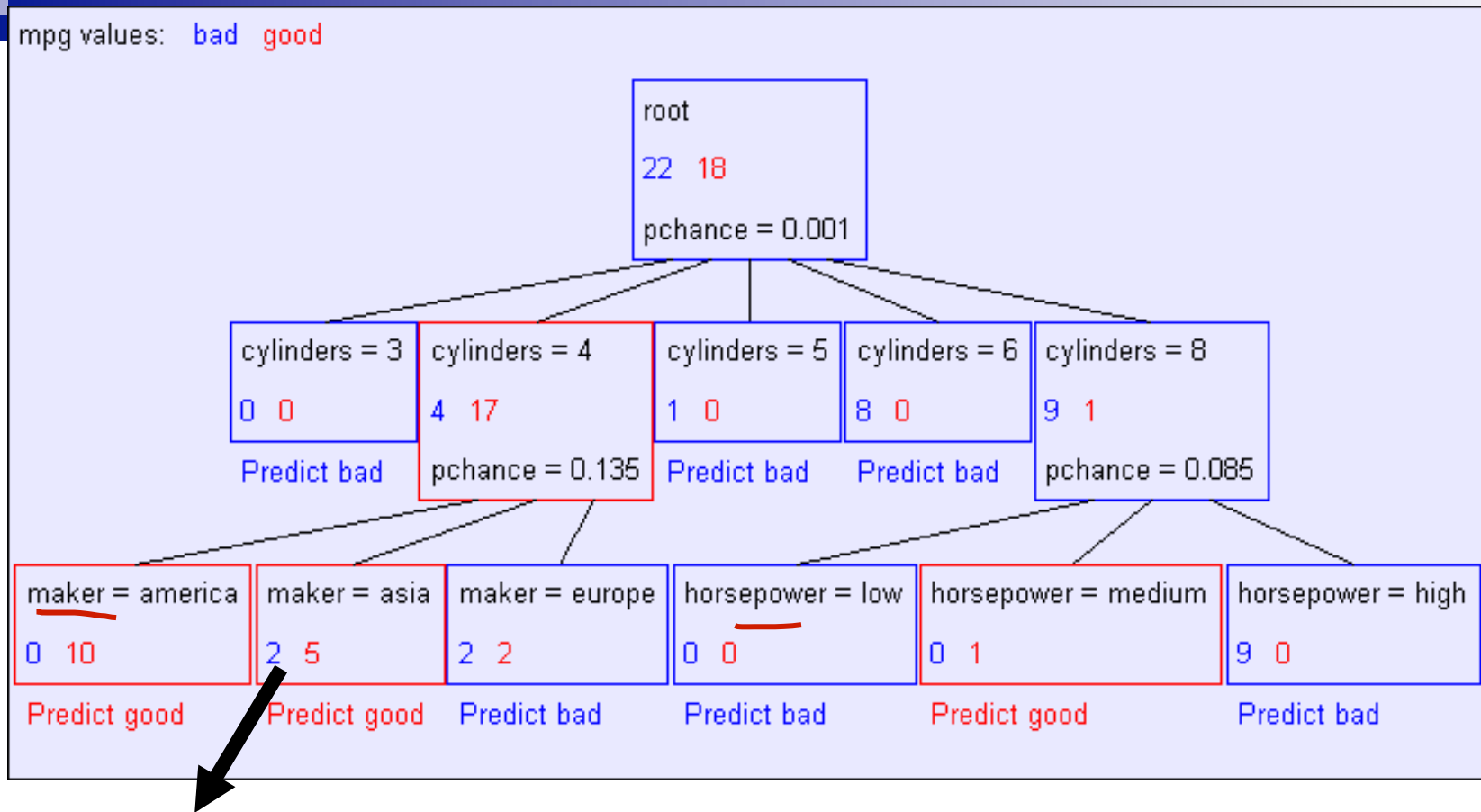
Records in
which cylinders
= 6

Build tree from
These examples..



Records in
which cylinders
= 8

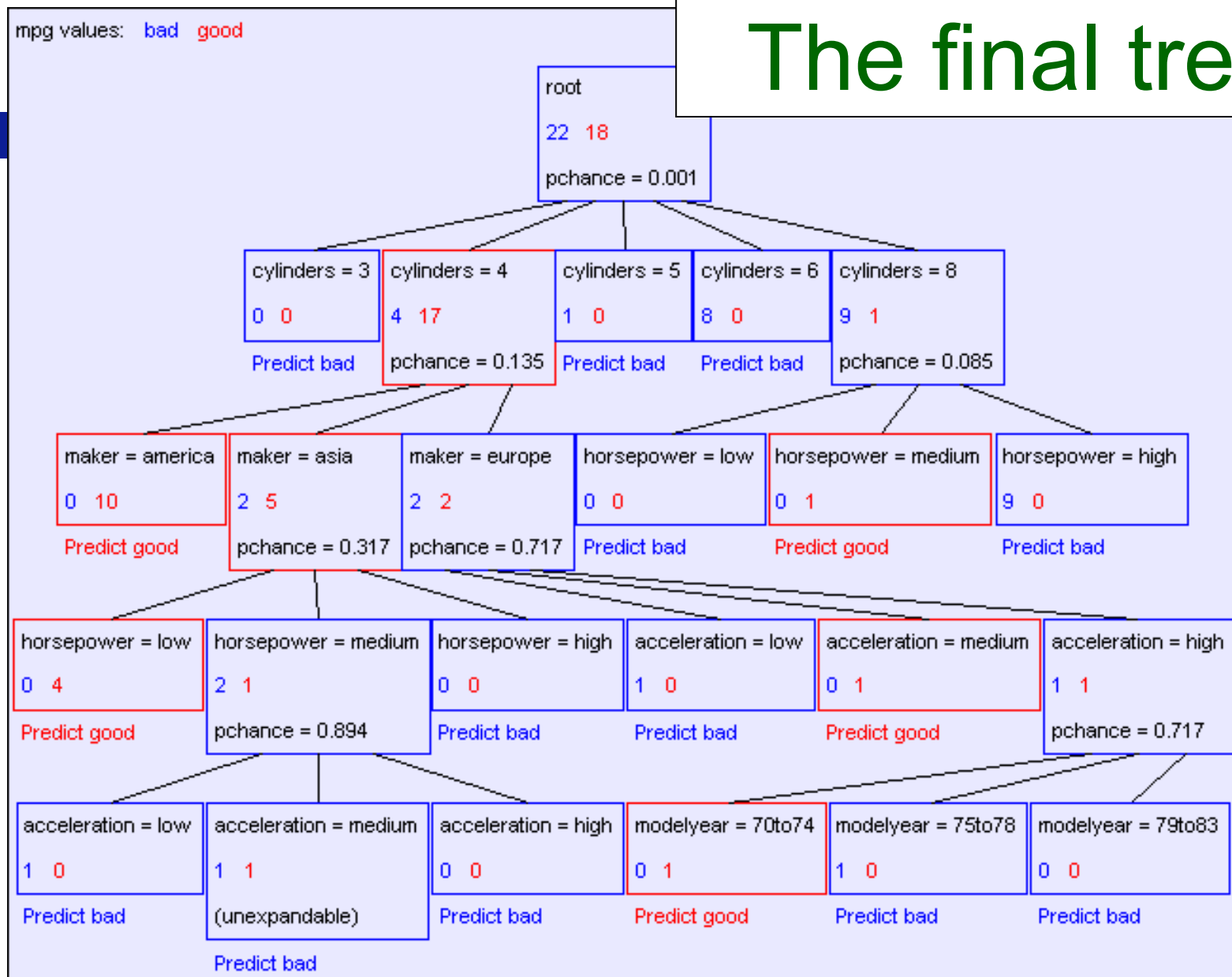
Second level of tree



Recursively build a tree from the seven records in which there are four cylinders and the maker was based in Asia

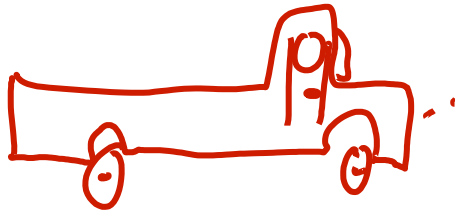
(Similar recursion in the other cases)

The final tree

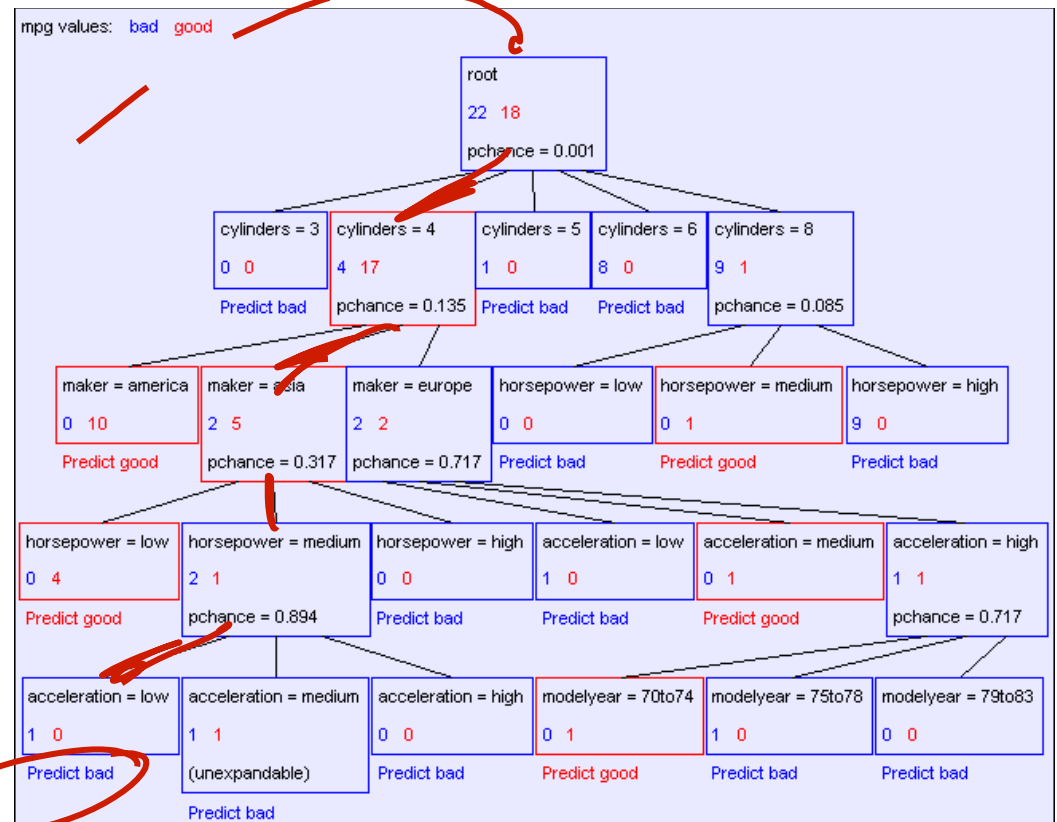


Classification of a new example

- Classifying a test example – traverse tree and report leaf label

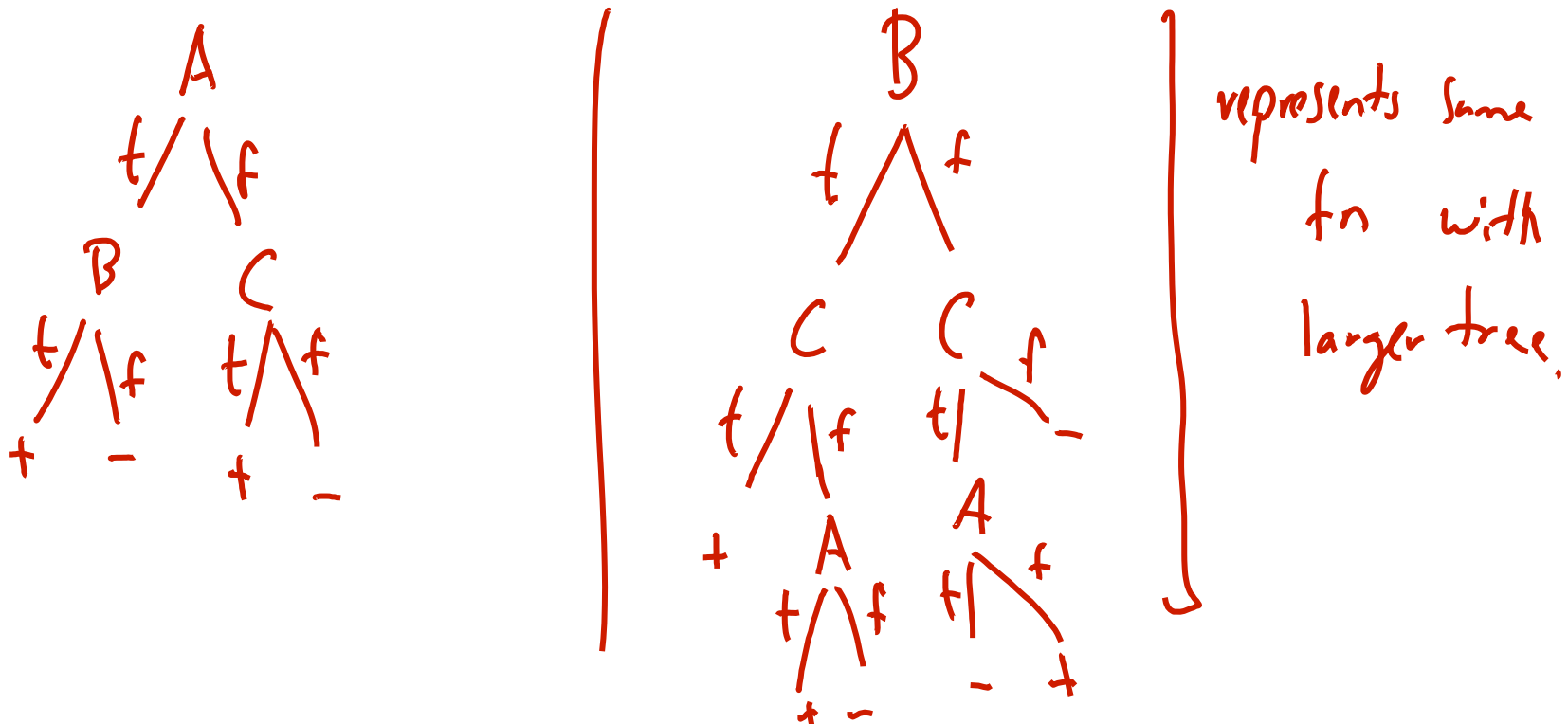


bad
MPG



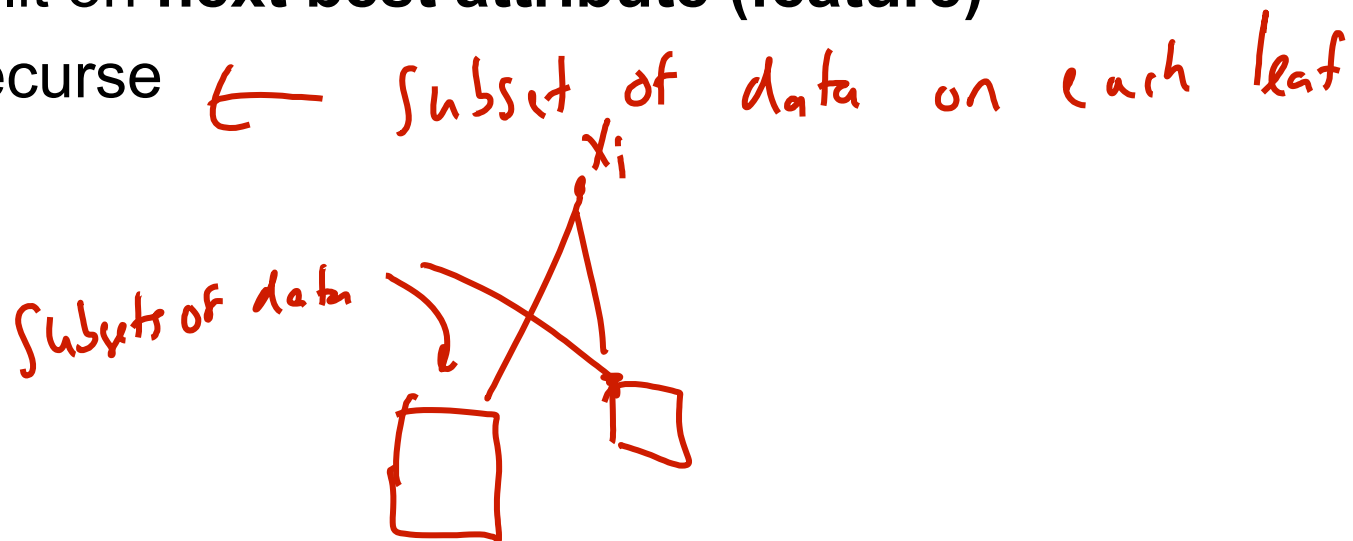
Are all decision trees equal?

- Many trees can represent the same concept
- But, not all trees will have the same size!
 - e.g., $\phi = A \wedge B \vee \neg A \wedge C$ ((A and B) or (not A and C))



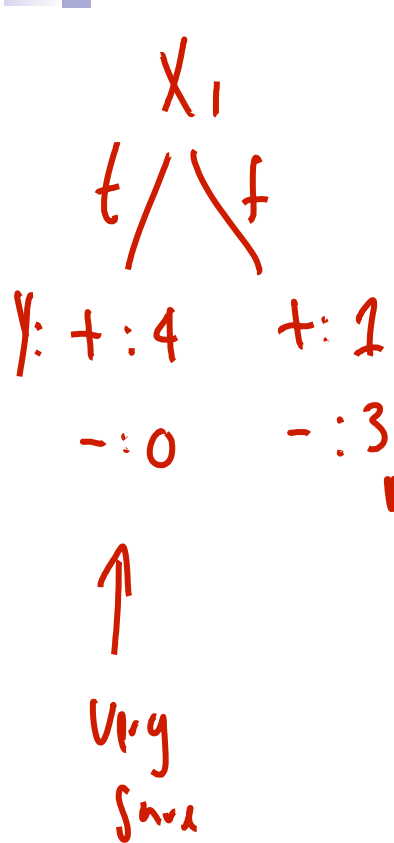
Learning decision trees is hard!!!

- Learning the simplest (smallest) decision tree is an NP-complete problem [Hyafil & Rivest '76]
- Resort to a greedy heuristic:
 - Start from empty decision tree
 - Split on **next best attribute (feature)**
 - Recurse

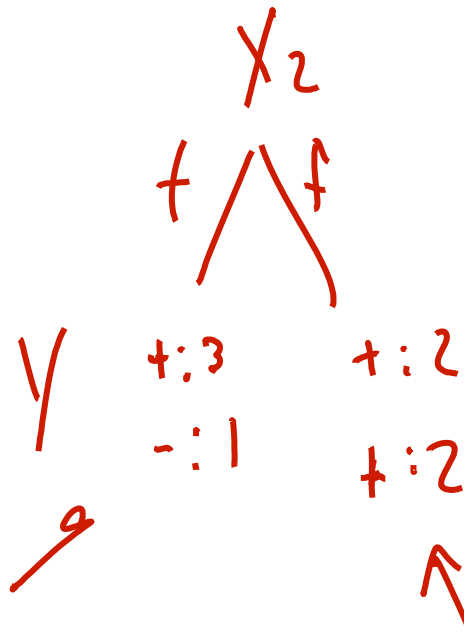


Choosing a good attribute

X_1	X_2	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F
F	T	F
F	F	F



Kind of
Sure



After split, X_1 makes me more sure than X_2

Measuring uncertainty

- Good split if we are more certain about classification after split
 - Deterministic good (all true or all false)
 - Uniform distribution bad

$P(Y=A) = 1/2$	$P(Y=B) = 1/4$	$P(Y=C) = 1/8$	$P(Y=D) = 1/8$
----------------	----------------	----------------	----------------

More sure
↓

$P(Y=A) = 1/4$	$P(Y=B) = 1/4$	$P(Y=C) = 1/4$	$P(Y=D) = 1/4$
----------------	----------------	----------------	----------------

Uniform
bad
✓

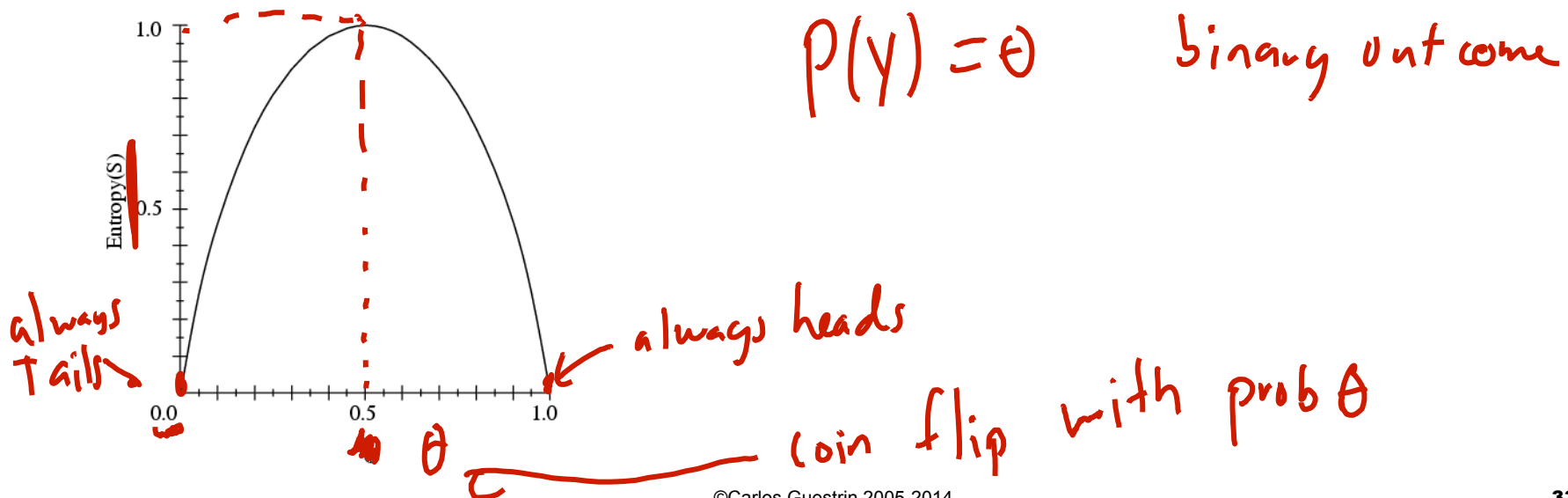
Entropy

Entropy $H(X)$ of a random variable Y

$$H(Y) = - \sum_{i=1}^k P(Y = y_i) \log_2 P(Y = y_i)$$

More uncertainty, more entropy!

Information Theory interpretation: $H(Y)$ is the expected number of bits needed to encode a randomly drawn value of Y (under most efficient code)



Andrew Moore's Entropy in a nutshell



Low Entropy



High Entropy

Andrew Moore's Entropy in a nutshell



Low Entropy

..the values (locations of soup) sampled entirely from within the soup bowl



High Entropy

..the values (locations of soup) unpredictable... almost uniformly sampled throughout our dining room

$x \log x \rightarrow 0$ as $x \rightarrow 0$

Information gain

X_1	X_2	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F

- Advantage of attribute – decrease in uncertainty

- Entropy of Y before you split $H(Y) = - \sum_y p(y) \log p(y)$
- Entropy after split $= -\frac{5}{6} \log \frac{5}{6} - \frac{1}{6} \log \frac{1}{6} = 0.65$

- Weight by probability of following each branch, i.e., normalized number of records

$$H(Y | X) = - \sum_{j=1}^v P(X = x_j) \sum_{i=1}^k P(Y = y_i | X = x_j) \log_2 P(Y = y_i | X = x_j)$$

1/6 prob
of going left
 X_1
+ f
+ : 1
- : 0

$$H(Y | X_1) = \frac{4}{6} \left(-\frac{4}{4} \log \frac{4}{4} - \frac{0}{4} \log \frac{0}{4} \right) + \frac{2}{6} \left(-\frac{1}{2} \log \frac{1}{2} - \frac{1}{2} \log \frac{1}{2} \right) = \frac{1}{3}$$

- Information gain is difference $IG(X) = H(Y) - H(Y | X)$

$$IG(X_1) = H(Y) - H(Y | X_1) = 0.65 - \frac{1}{3} \approx 0.32$$

Learning decision trees

- Start from empty decision tree
- Split on **next best attribute (feature)**
 - Use, for example, information gain to select attribute
 - Split on $\arg \max_i IG(X_i) = \arg \max_i H(Y) - H(Y | X_i)$

- Recurse *for each split*

when do I stop?

1. when info gain is small ?????

2. entropy in leaf is 0, perfect classification

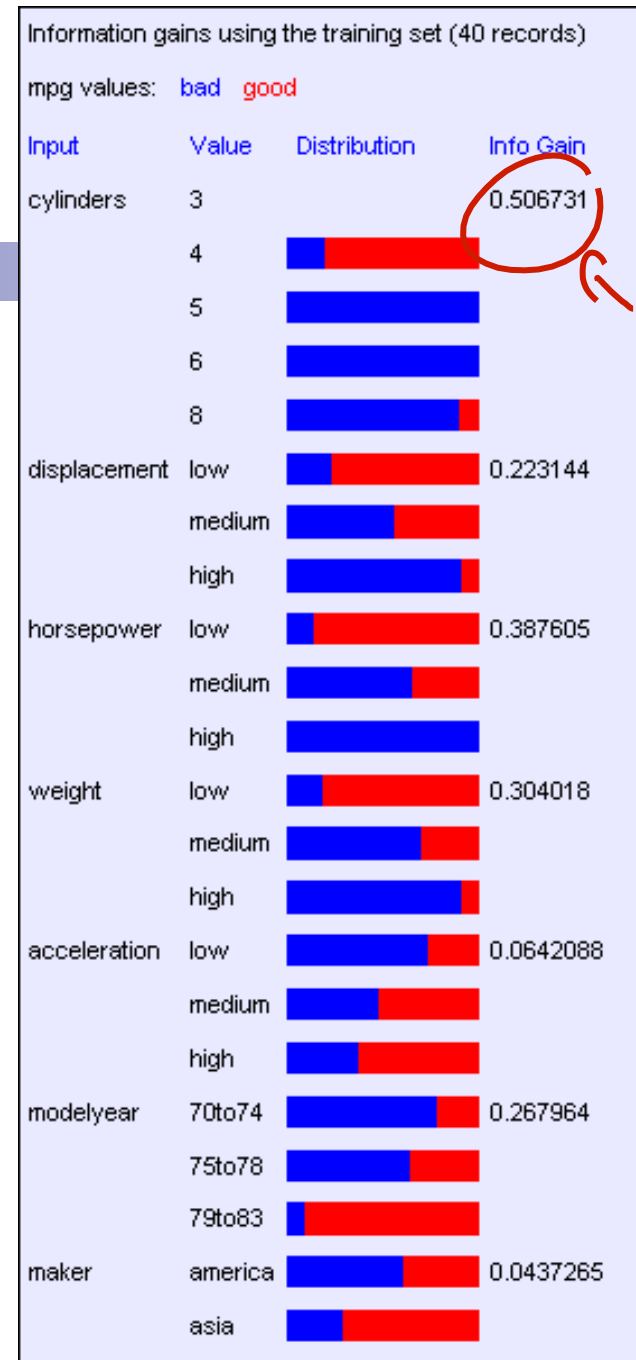
3. nothing to split on

done with all vars
no features split data

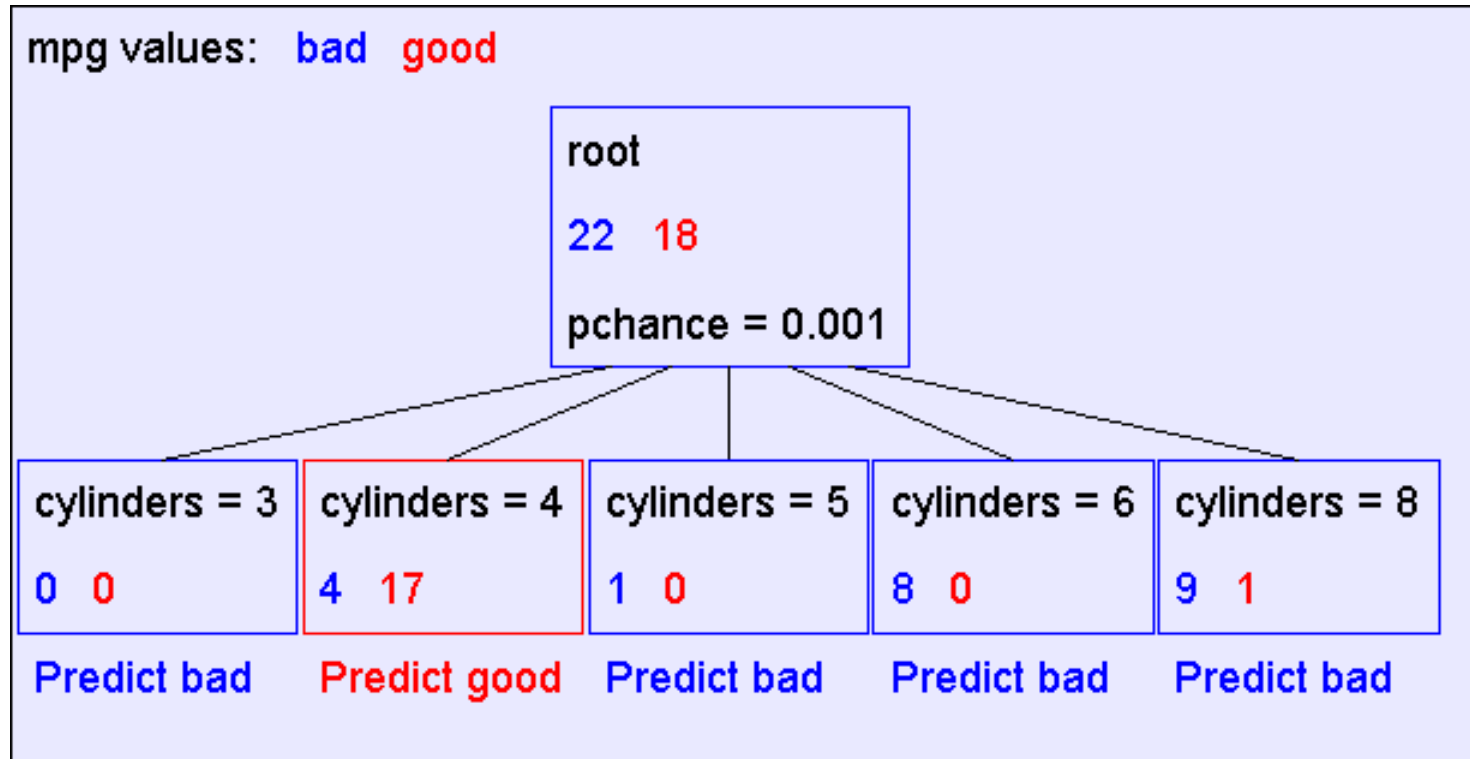
*too many?
may not be a
good idea*

Suppose we want
to predict MPG

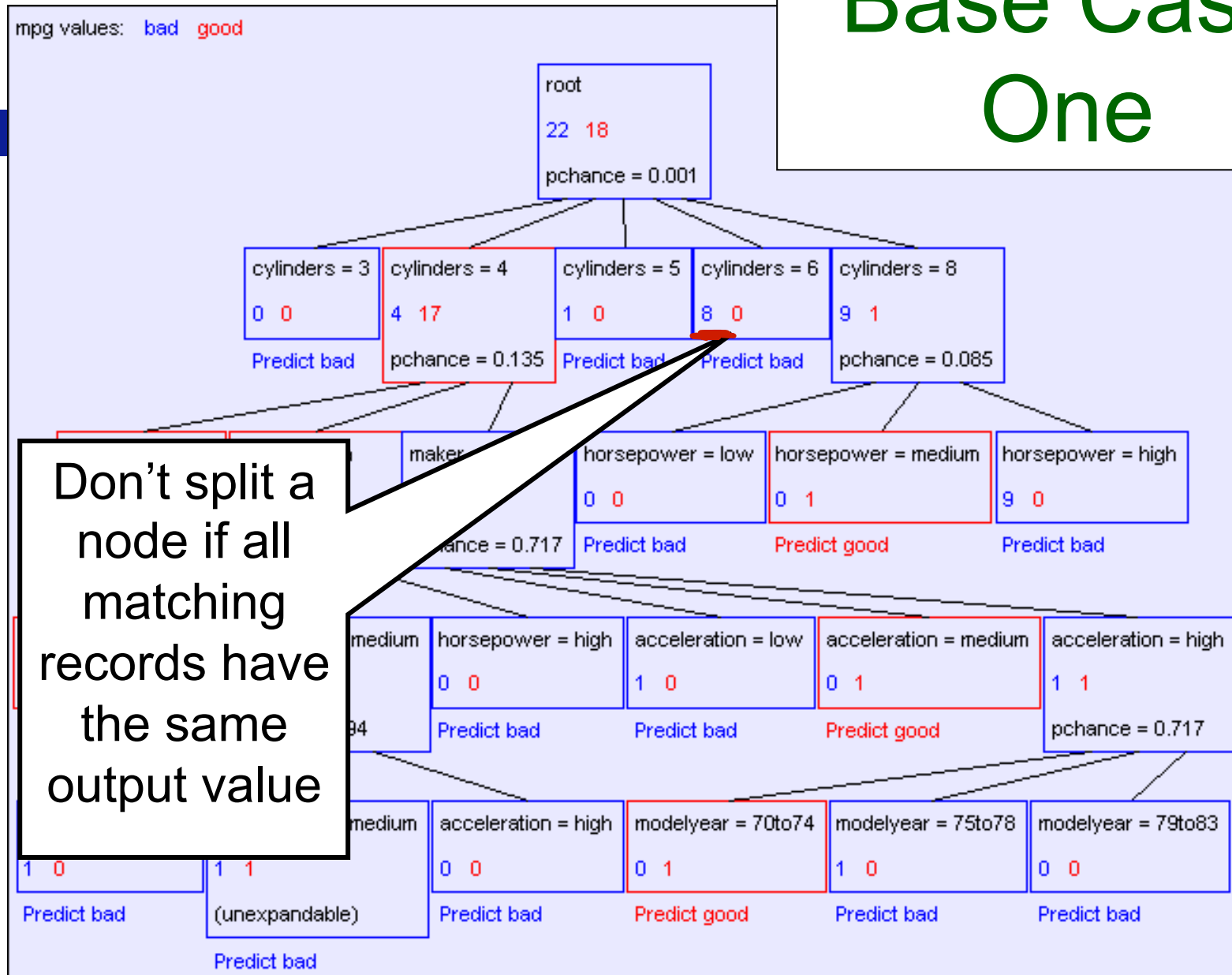
Look at all the
information
gains...



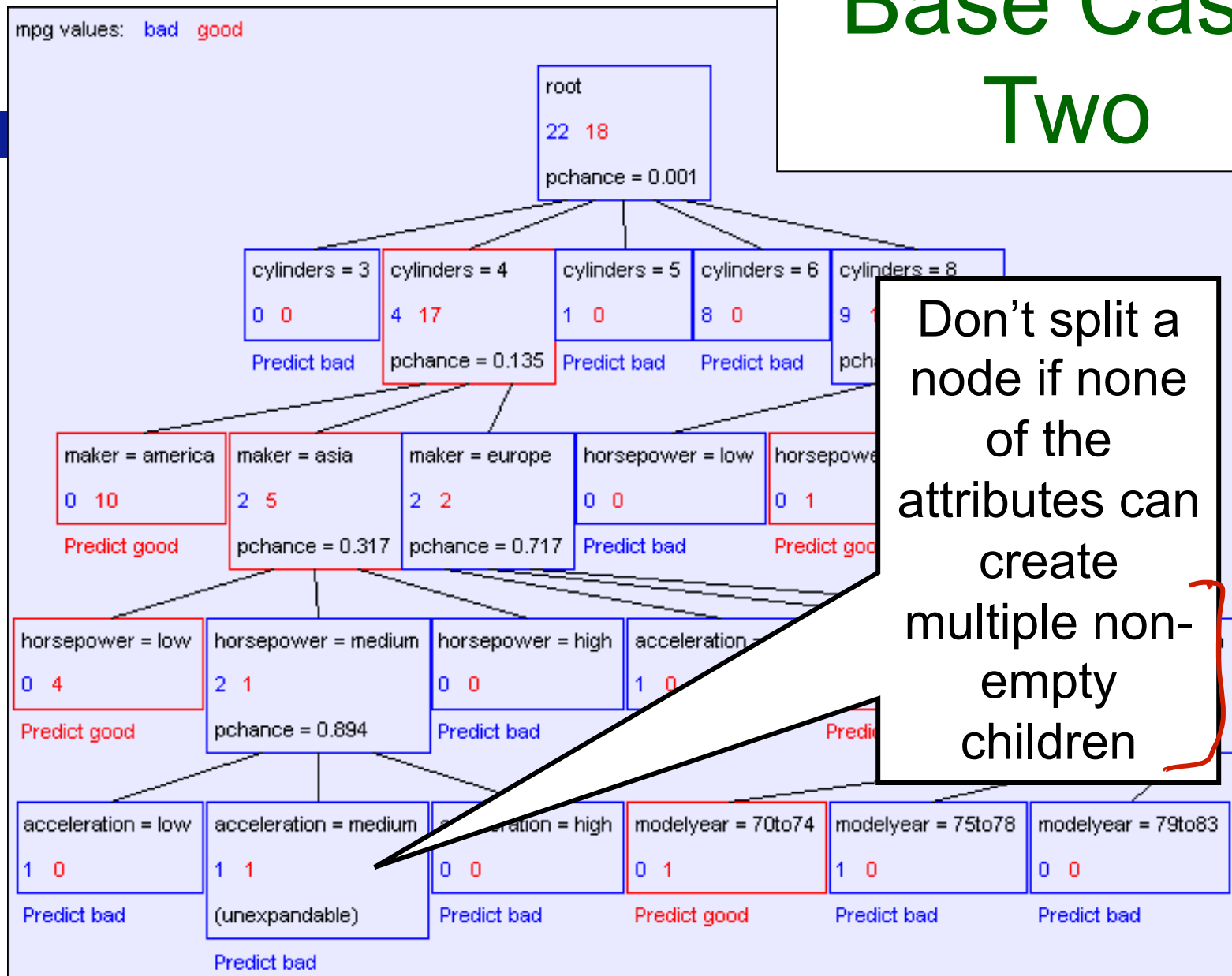
A Decision Stump



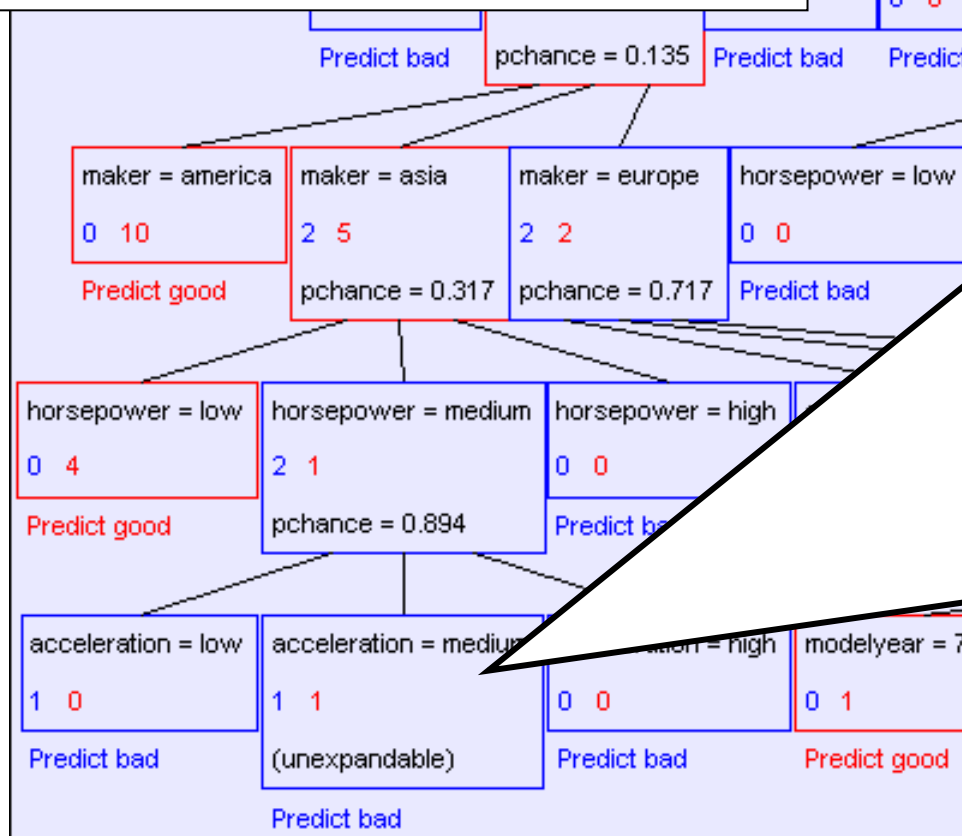
Base Case One



Base Case Two



Base Case Two: No attributes can distinguish



Information gains using the training set (2 records)

mpg values: bad good

Input	Value	Distribution	Info Gain
cylinders	3		0
	4		
	5		
	6		
	8		
displacement	low		0
	medium		
	high		
horsepower	low		0
	medium		
	high		
weight	low		0
	medium		
	high		
acceleration	low		0
	medium		
	high		
modelyear	70to74		0
	75to78		
	79to83		
maker	america		0
	asia		
	europe		

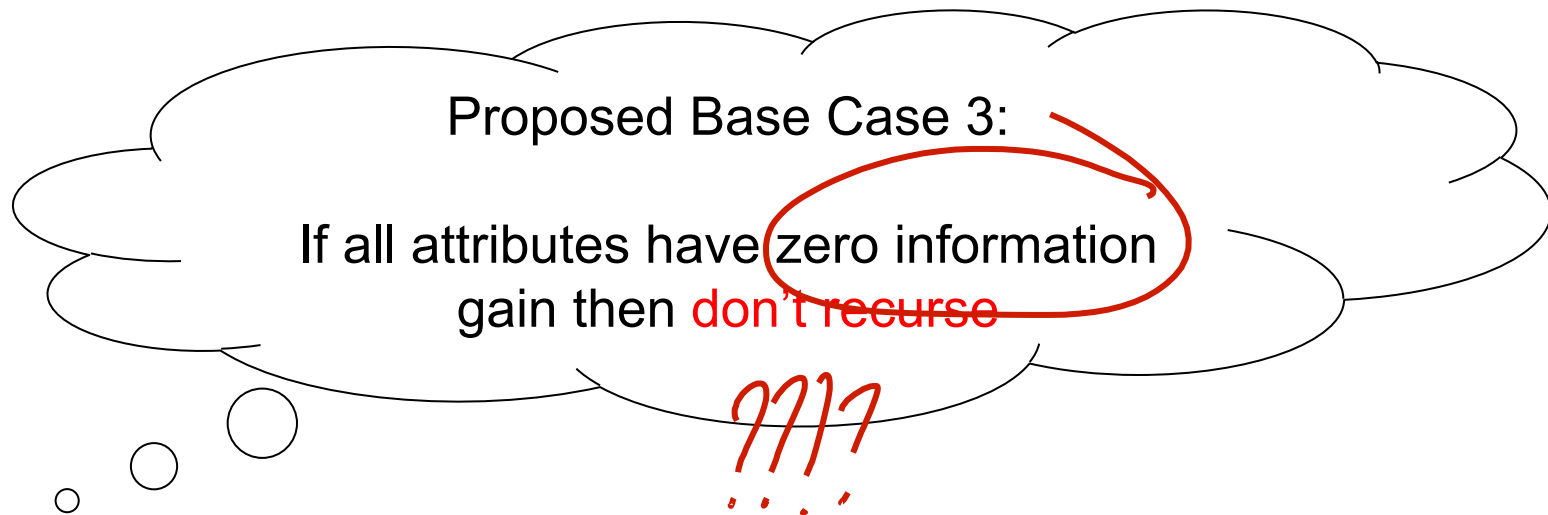
Base Cases



- Base Case One: If all records in current data subset have the same output then **don't recurse**
- Base Case Two: If all records have exactly the same set of input attributes then **don't recurse**

Base Cases: An idea

- Base Case One: If all records in current data subset have the same output then **don't recurse**
- Base Case Two: If all records have exactly the same set of input attributes then **don't recurse**







• *Is this a good idea?*

The problem with Base Case 3

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

$$Y = A \text{ XOR } B$$

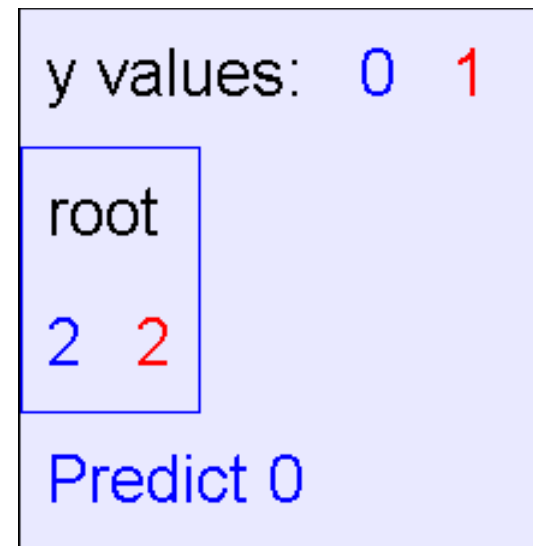
The information gains:

Information gains using the training set (4 records)				
y values: 0 1				
Input	Value	Distribution	Info Gain	
a	0		0	
	1			
b	0		0	
	1			

$$IG(A) = 0$$

$$IG(B) = 0$$

The resulting bad decision tree:



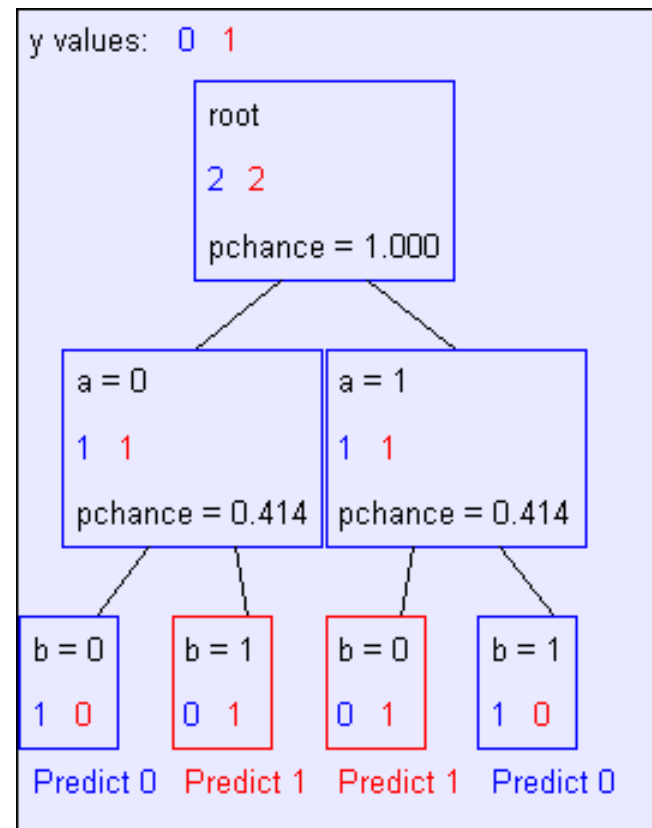
If we omit Base Case 3:

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

$$y = a \text{ XOR } b$$

The resulting decision tree:

*low info gain not a
good stopping criteria*



Basic Decision Tree Building Summarized

BuildTree(*DataSet*, *Output*)

- If all output values are the same in *DataSet*, return a leaf node that says “predict this unique output”
- If all input values are the same, return a leaf node that says “predict the majority output”
- Else find attribute *X* with highest Info Gain
- Suppose *X* has n_X distinct values (i.e. *X* has arity n_X).
 - Create and return a non-leaf node with n_X children.
 - The *i*th child should be built by calling

BuildTree(DS_i , *Output*)

Where DS_i built consists of all those records in *DataSet* for which *X* = *i*th distinct value of *X*.

go on for ever ...

MPG Test set error

mpg values: bad good

root
22 18
pchance = 0.001

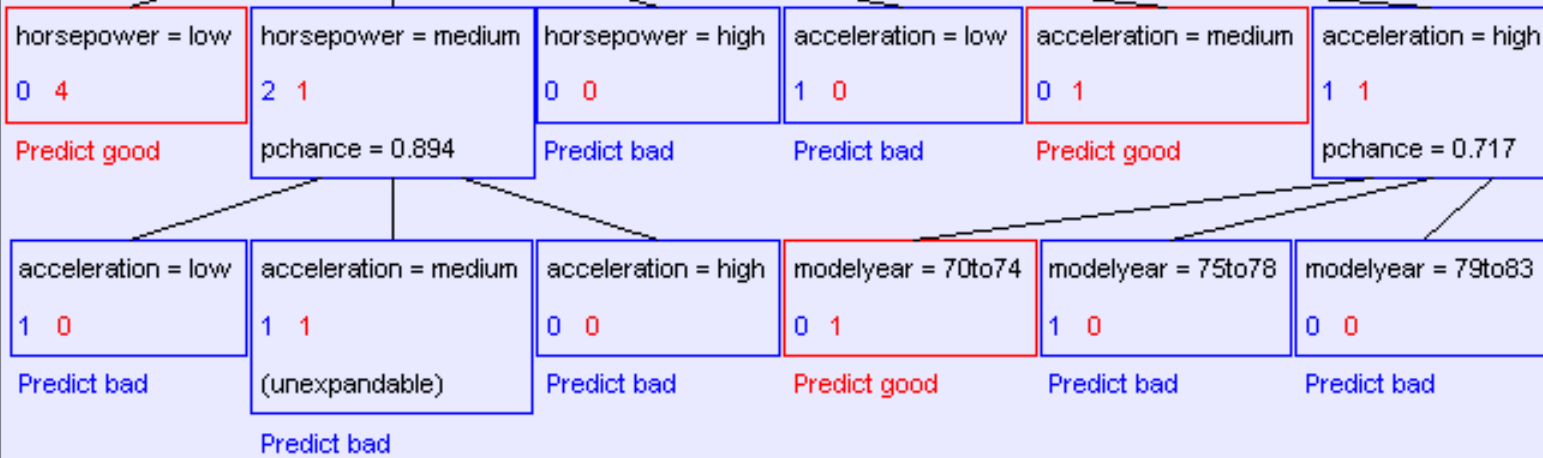
	Num Errors	Set Size	Percent Wrong
Training Set	<u>1</u>	<u>40</u>	<u>2.50%</u>
Test Set	74	352	<u>21.02%</u>

overfit

error test

horsepower = high

Predict bad



MPG Test set error

mpg values: bad good

root
22 18
pchance = 0.001

	Num Errors	Set Size	Percent Wrong
Training Set	1	40	2.50
Test Set	74	352	21.02

horsepower = low

horsepower = medium

horsepower = high

acceleration = low

acceleration = medium

acceleration = high

The test set error is much worse than the training set error...

...why?

Predict bad

(unexpandable)

Predict bad

Predict good

Predict bad

Predict bad

Predict bad

Decision trees & Learning Bias



Suppose no "label noise"
 ↳ two data points
 same x different y

↳ Decision trees eventually have
 zero train error

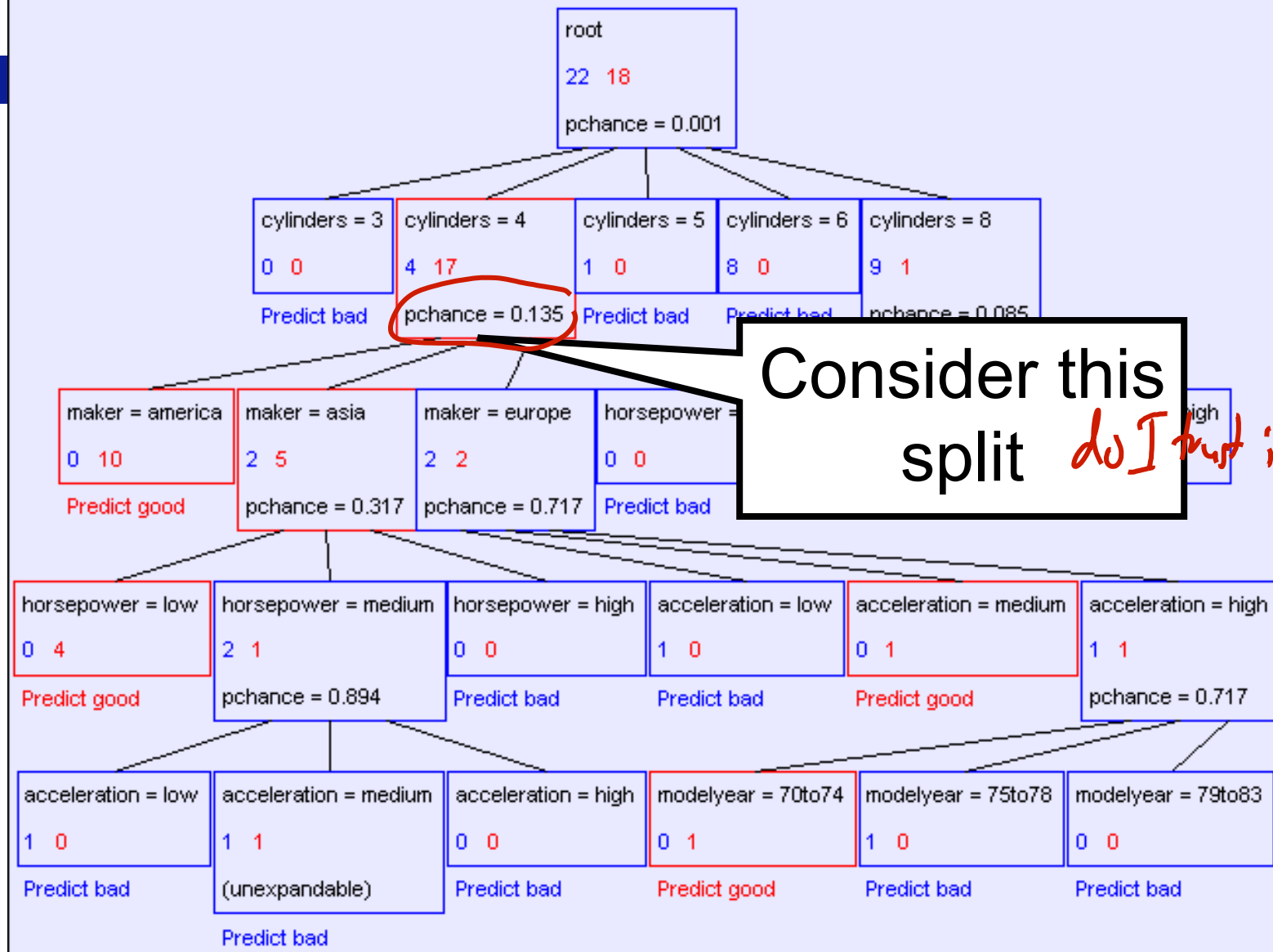
↳ overfit!!
 ∩

mpg	cylinders	displacement	horsepower	weight	acceleration	modelyear	maker
good	4	low	low	low	high	75to78	asia
bad	6	medium	medium	medium	medium	70to74	america
bad	4	medium	medium	medium	low	75to78	europa
bad	8	high	high	high	low	70to74	america
bad	6	medium	medium	medium	medium	70to74	america
bad	4	low	medium	low	medium	70to74	asia
bad	4	low	medium	low	low	70to74	asia
bad	8	high	high	high	low	75to78	america
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
bad	8	high	high	high	low	70to74	america
good	8	high	medium	high	high	79to83	america
bad	8	high	high	high	low	75to78	america
good	4	low	low	low	low	79to83	america
bad	6	medium	medium	medium	high	75to78	america
good	4	medium	low	low	low	79to83	america
good	4	low	low	medium	high	79to83	america
bad	8	high	high	high	low	70to74	america
good	4	low	medium	low	medium	75to78	europa
bad	5	medium	medium	medium	medium	75to78	europa

Decision trees will overfit

- Standard decision trees ~~are~~ have no learning bias
 - Training set error is always zero!
 - (If there is no label noise)
 - Lots of variance
 - Will definitely overfit!!!
 - Must bias towards simpler trees
- Many strategies for picking simpler trees:
 - Fixed depth
 - Fixed number of leaves
 - Or something smarter...

mpg values: bad good



A chi-square test

mpg values: bad good

maker	america	0	10		$H(\text{mpg} \mid \text{maker} = \text{america}) = 0$
	asia	2	5		$H(\text{mpg} \mid \text{maker} = \text{asia}) = 0.863121$
	europa	2	2		$H(\text{mpg} \mid \text{maker} = \text{europa}) = 1$




$H(\text{mpg}) = 0.702467$ $H(\text{mpg} \mid \text{maker}) = 0.478183$
 $IG(\text{mpg} \mid \text{maker}) = 0.224284$

- Suppose that MPG was completely uncorrelated with maker.
- What is the chance we'd have seen data of at least this apparent level of association anyway?

if prob of chance correlation high, don't keep split

A chi-square test

mpg values: bad good

maker	america	0	10			$H(\text{mpg} \mid \text{maker} = \text{america}) = 0$
	asia	2	5			$H(\text{mpg} \mid \text{maker} = \text{asia}) = 0.863121$
	europa	2	2			$H(\text{mpg} \mid \text{maker} = \text{europa}) = 1$

$H(\text{mpg}) = 0.702467$ $H(\text{mpg} \mid \text{maker}) = 0.478183$

$IG(\text{mpg} \mid \text{maker}) = 0.224284$

- Suppose that mpg was completely uncorrelated with maker.
- What is the chance we'd have seen data of at least this apparent level of association anyway?

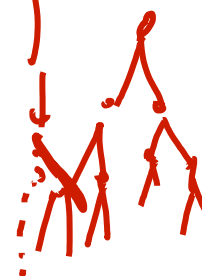
By using a particular kind of chi-square test, the answer is 7.2%

(Such simple hypothesis tests are very easy to compute, unfortunately, not enough time to cover in the lecture, but see readings...)

Using Chi-squared to avoid overfitting

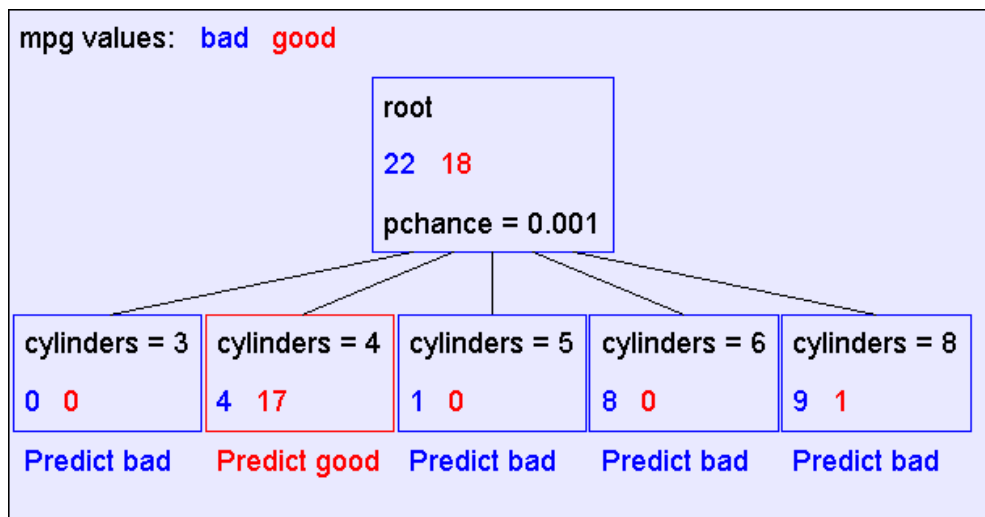
- Build the full decision tree as before
- But when you can grow it no more, start to prune:
 - Beginning at the bottom of the tree, delete splits in which $p_{\text{chance}} > \text{MaxPchance}$
 - Continue working your way up until there are no more prunable nodes

MaxPchance is a magic parameter you must specify to the decision tree, indicating your willingness to risk fitting noise



Pruning example

- With MaxPchance = 0.1, you will see the following MPG decision tree:



Note the improved test set accuracy compared with the unpruned tree

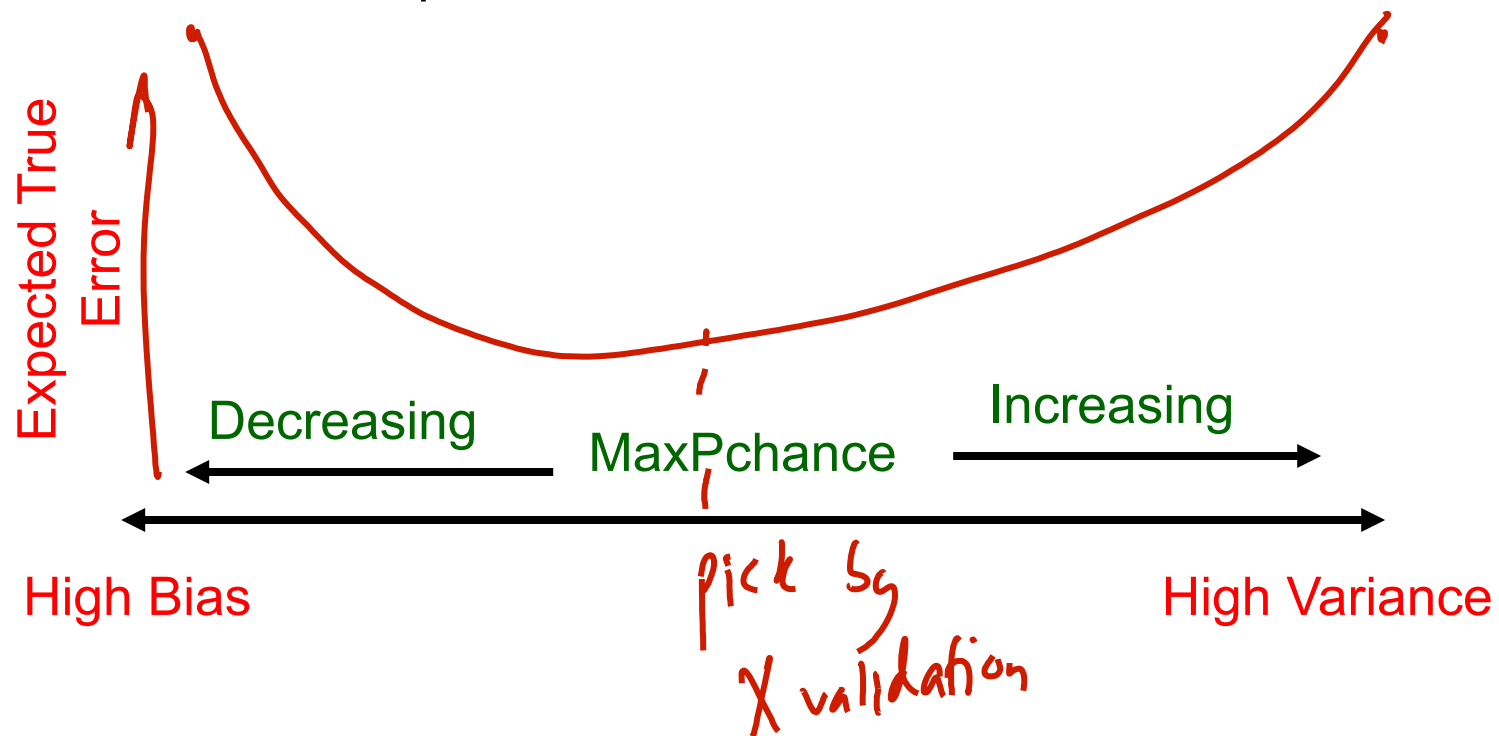
	Num Errors	Set Size	Percent Wrong
Training Set	5	40	12.50
Test Set	56	352	15.91

higher train error (pointing to 12.50)

lower test error (pointing to 15.91)

MaxPchance

- Technical note MaxPchance is a regularization parameter that helps us bias towards simpler models



Real-Valued inputs

- What should we do if some of the inputs are real-valued?

mpg	cylinders	displacement	horsepower	weight	acceleration	modelyear	maker
good	4	97	75	2265	18.2	77	asia
bad	6	199	90	2648	15	70	america
bad	4	121	110	2600	12.8	77	europa
bad	8	350	175	4100	13	73	america
bad	6	198	95	3102	16.5	74	america
bad	4	108	94	2379	16.5	73	asia
bad	4	113	95	2228	14	71	asia
bad	8	302	139	3570	12.8	78	america
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
good	4	120	79	2625	18.6	82	america
bad	8	455	225	4425	10	70	america
good	4	107	86	2464	15.5	76	europa
bad	5	131	103	2830	15.9	78	europa

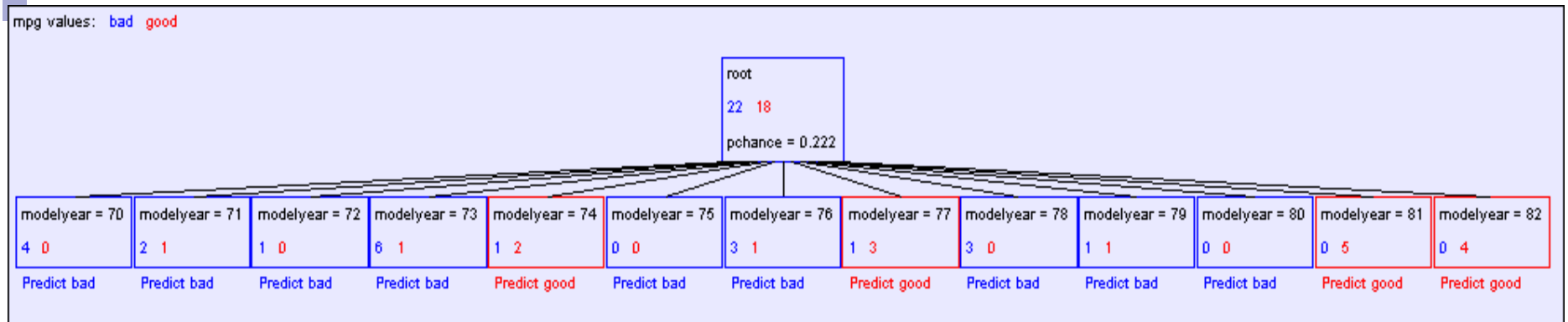


Infinite number of possible split values!!!

Finite dataset, only finite number of relevant splits!

Idea One: Branch on each possible real value

“One branch for each numeric value” idea:



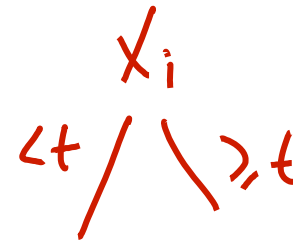
Hopeless: with such high branching factor will shatter the dataset and overfit

Threshold splits

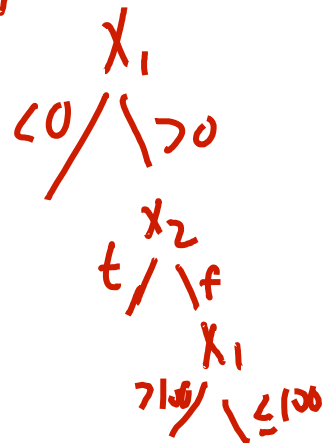
- Binary tree, split on attribute X_i

- One branch: $X_i < t$

- Other branch: $X_i \geq t$



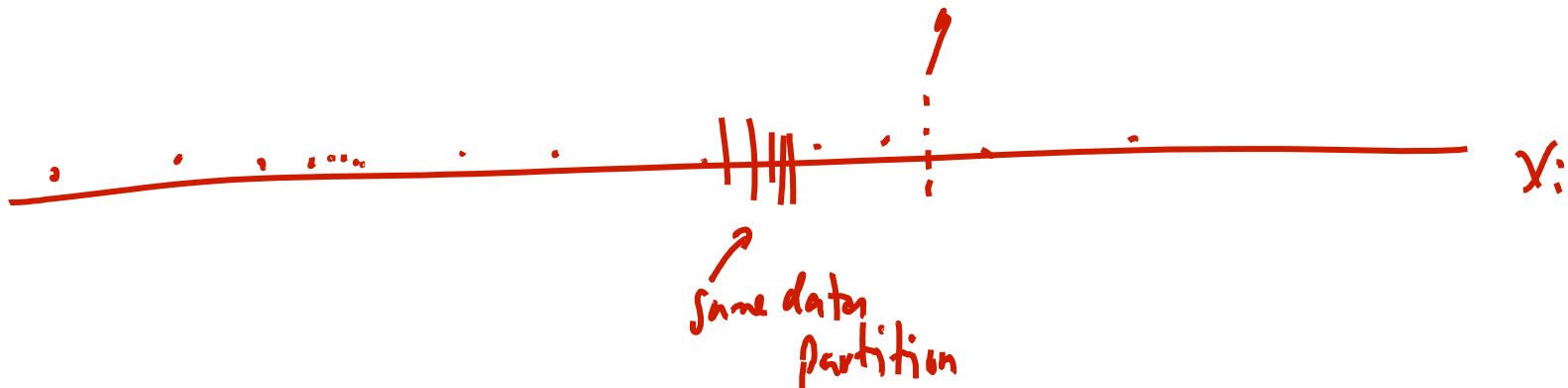
could split on x_i multiple times



Choosing threshold split

- Binary tree, split on attribute X_i
 - One branch: $X_i < t$
 - Other branch: $X_i \geq t$
- Search through possible values of t
 - Seems hard!!!
- But only finite number of t 's are important
 - Sort data according to X into $\{x_1, \dots, x_m\}$
 - Consider split points of the form $x_i + (x_{i+1} - x_i)/2$

info gain like a discrete binary variable


















A better idea: thresholded splits

- Suppose X is real valued
- Define $IG(Y|X:t)$ as $H(Y) - H(Y|X:t)$
- Define $H(Y|X:t) = \boxed{H(Y|X < t)} P(X < t) + \boxed{H(Y|X \geq t)} P(X \geq t)$
 - $IG(Y|X:t)$ is the information gain for predicting Y if all you know is whether X is greater than or less than t
- Then define $IG^*(Y|X) = \max_t IG(Y|X:t)$
- For each real-valued attribute, use $IG^*(Y|X)$ for assessing its suitability as a split
- Note, may split on an attribute multiple times, with different thresholds

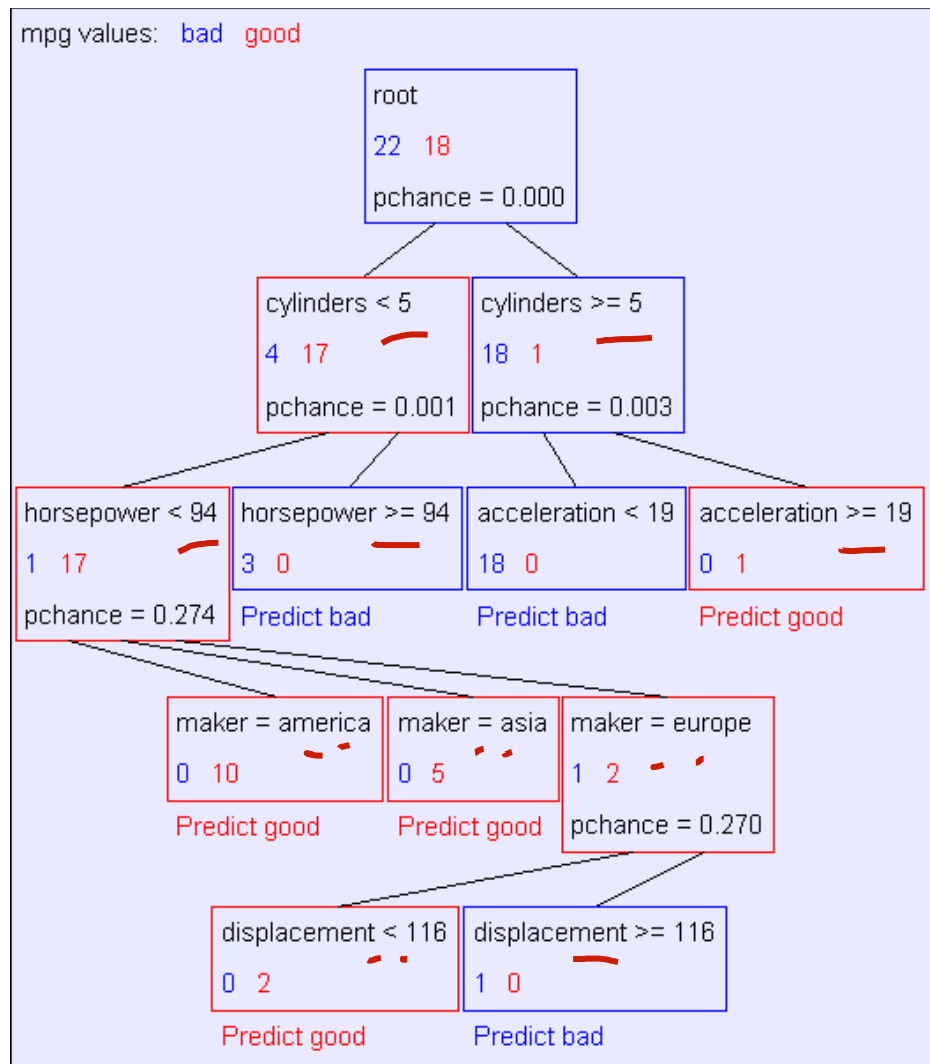
Example with MPG

Information gains using the training set (40 records)

mpg values: bad good

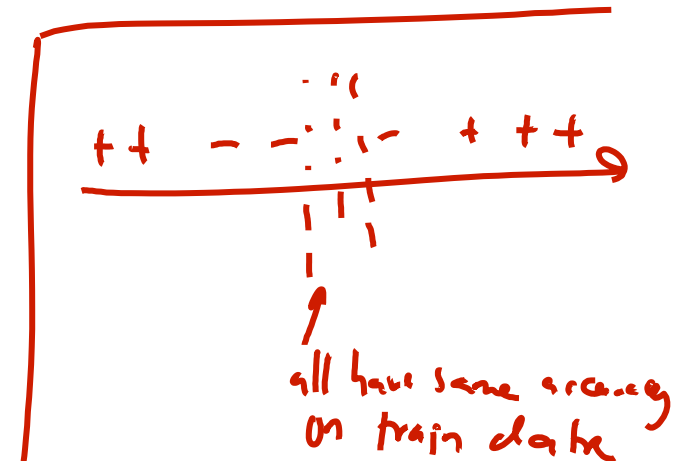
Input	Value	Distribution	Info Gain
cylinders	< 5		0.48268
	>= 5		
displacement	< 198		0.428205
	>= 198		
horsepower	< 94		0.48268
	>= 94		
weight	< 2789		0.379471
	>= 2789		
acceleration	< 18.2		0.159982
	>= 18.2		
modelyear	< 81		0.319193
	>= 81		
maker	america		0.0437265
	asia		
	europa		

Example tree using reals



What you need to know about decision trees

- Decision trees are one of the most popular data mining tools
 - Easy to understand
 - Easy to implement
 - Easy to use
 - Computationally cheap (to solve heuristically)
- Information gain to select attributes (ID3, C4.5,...)
- Presented for classification, can be used for regression and density estimation too
- Decision trees will overfit!!!
 - Zero bias classifier ! Lots of variance
 - Must use tricks to find “simple trees”, e.g.,
 - Fixed depth/Early stopping
 - Pruning
 - Hypothesis testing



Acknowledgements



- Some of the material in the decision trees presentation is courtesy of Andrew Moore, from his excellent collection of ML tutorials:
 - <http://www.cs.cmu.edu/~awm/tutorials>



Instance-based Learning

Nearest Neighbors/Non-
Parametric Methods

Machine Learning – CSEP546

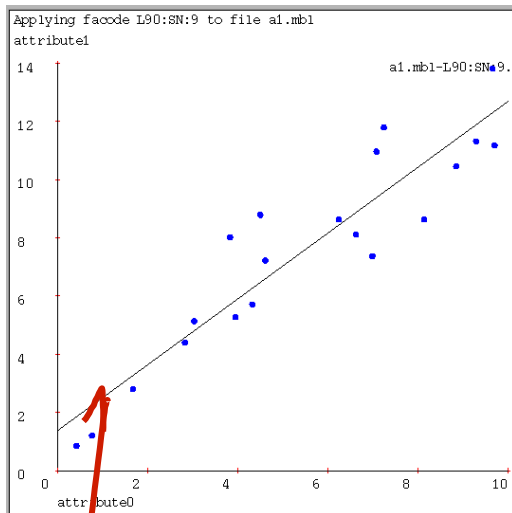
Carlos Guestrin

University of Washington

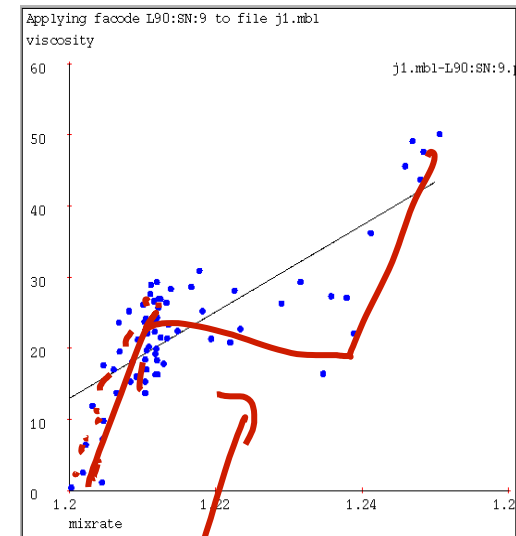
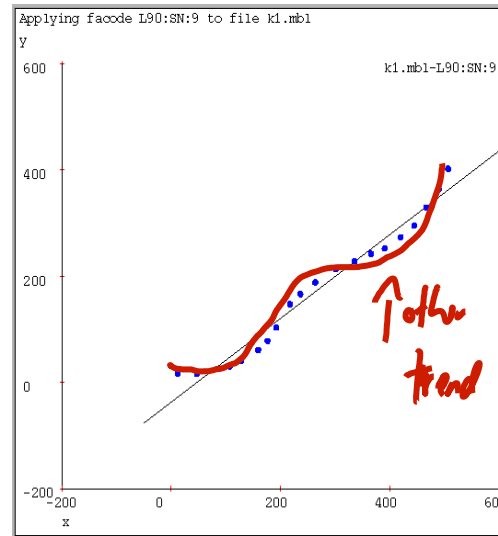
February 3, 2014

©Carlos Guestrin 2005-2014

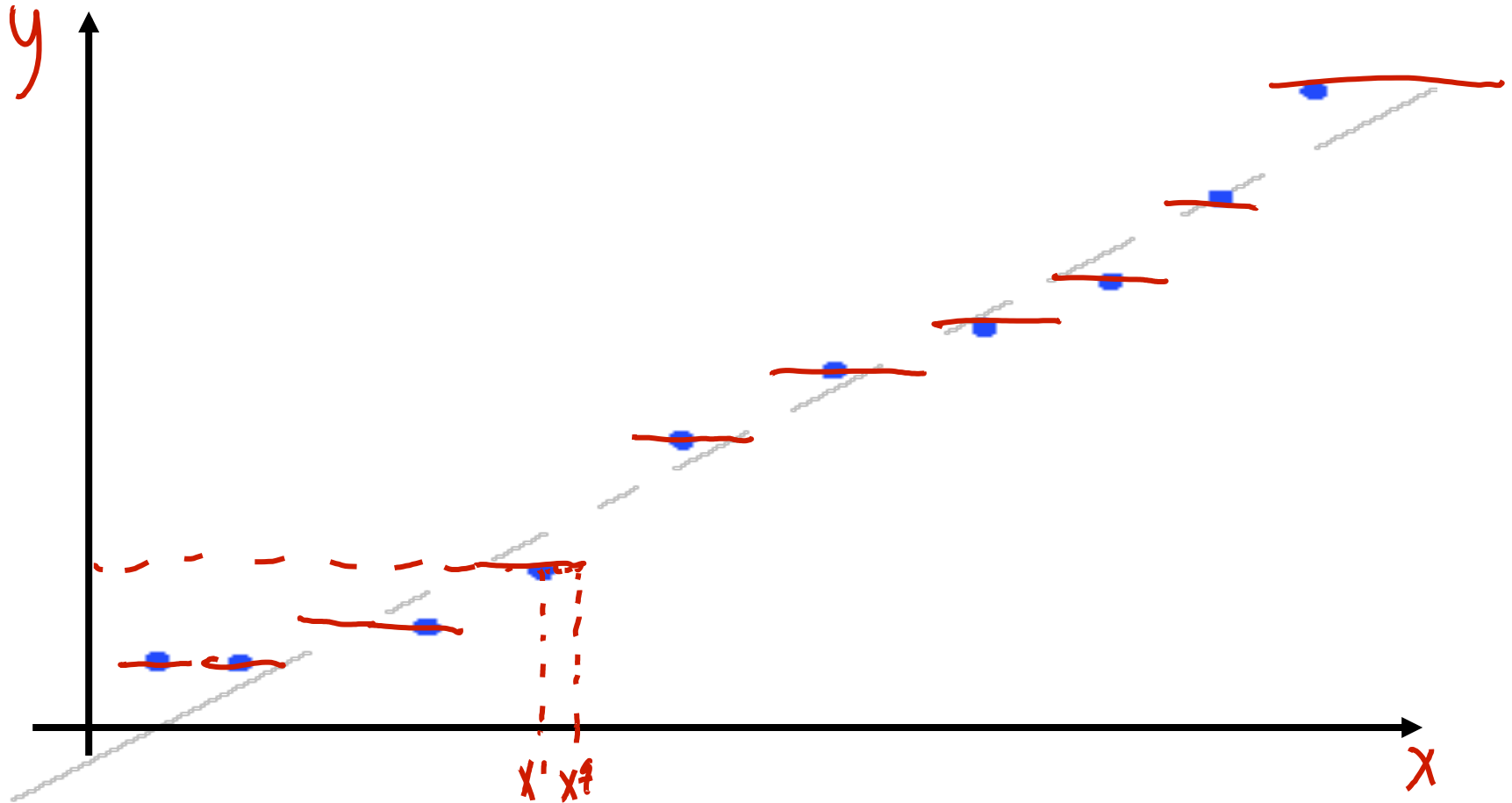
Why not just use Linear Regression?



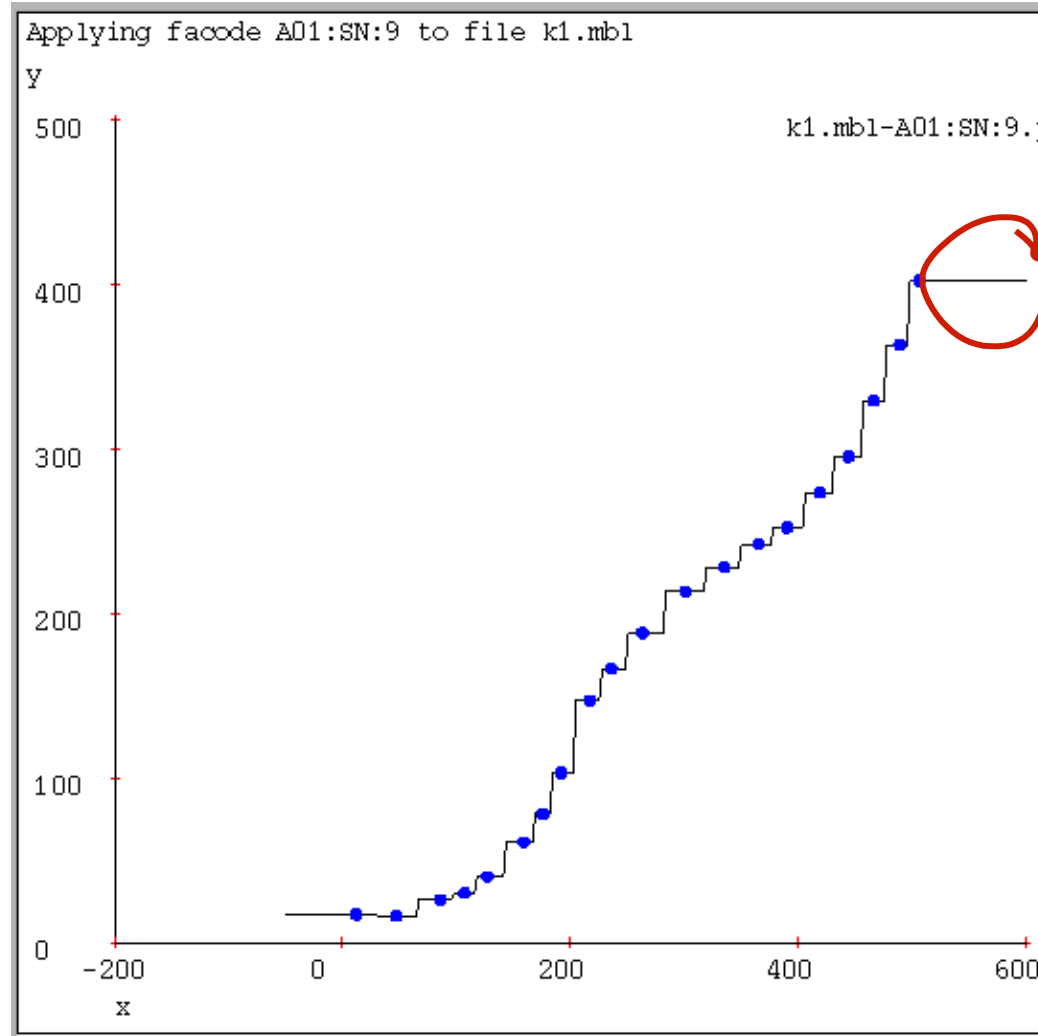
look ok



Using data to predict new data



Nearest neighbor



best??

Univariate 1-Nearest Neighbor

Given datapoints $(x^1, y^1) (x^2, y^2) \dots (x^N, y^N)$, where we assume $y^i = f(x^i)$ for some unknown function f .

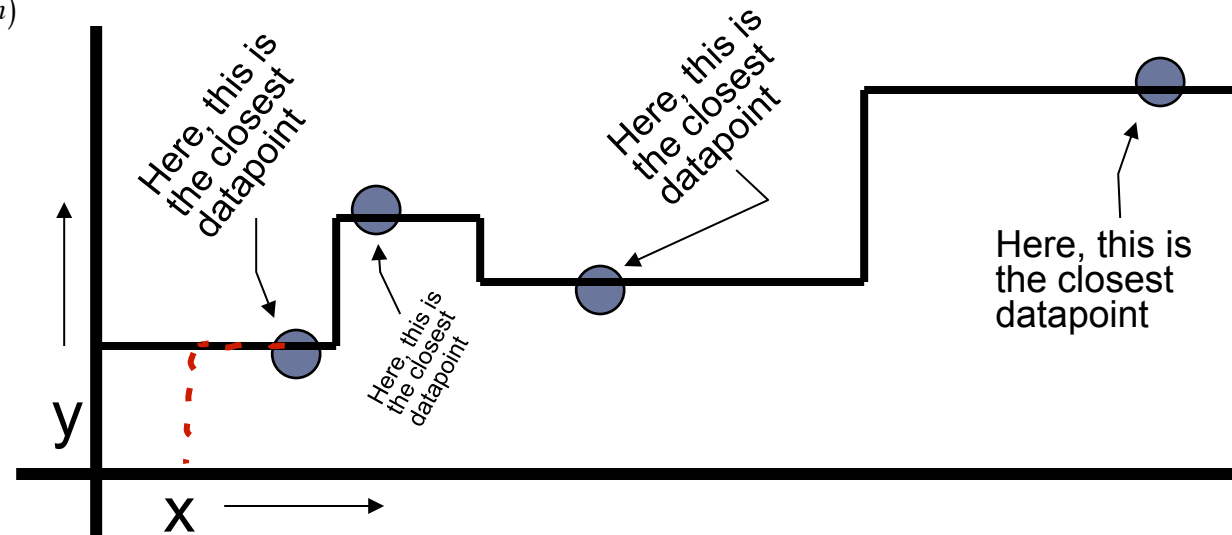
Given query point x^q , your job is to predict $\hat{y} \approx f(x^q)$
Nearest Neighbor:

1. Find the closest x_i in our set of datapoints

$$j(nn) = \underset{j}{\operatorname{argmin}} |x^j - x^q|$$

2. Predict $\hat{y} = y^{i(nn)}$

Here's a dataset with one input, one output and four datapoints.

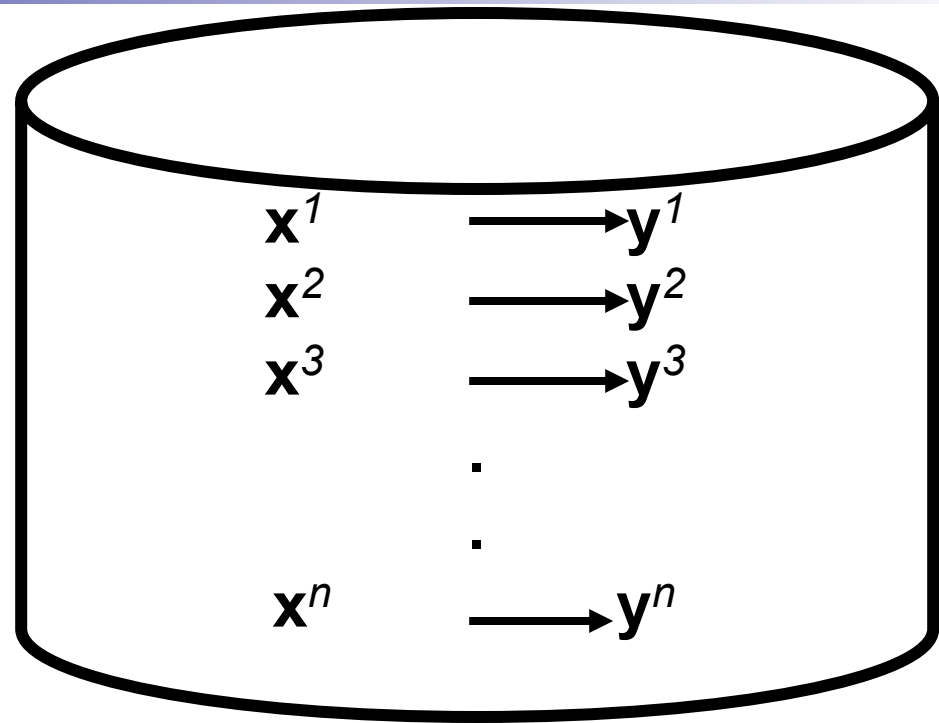


1-Nearest Neighbor is an example of....

Instance-based learning

A function approximator that has been around since about 1910.

To make a prediction, search database for similar datapoints, and fit with the local points.



Four things make a memory based learner:

- A distance metric
- How many nearby neighbors to look at?
- A weighting function (optional)
- How to fit with the local points?

1-Nearest Neighbor

Four things make a memory based learner:

1. *A distance metric*

Euclidian (and many more)

2. *How many nearby neighbors to look at?*

One

3. *A weighting function (optional)*

Unused

4. *How to fit with the local points?*

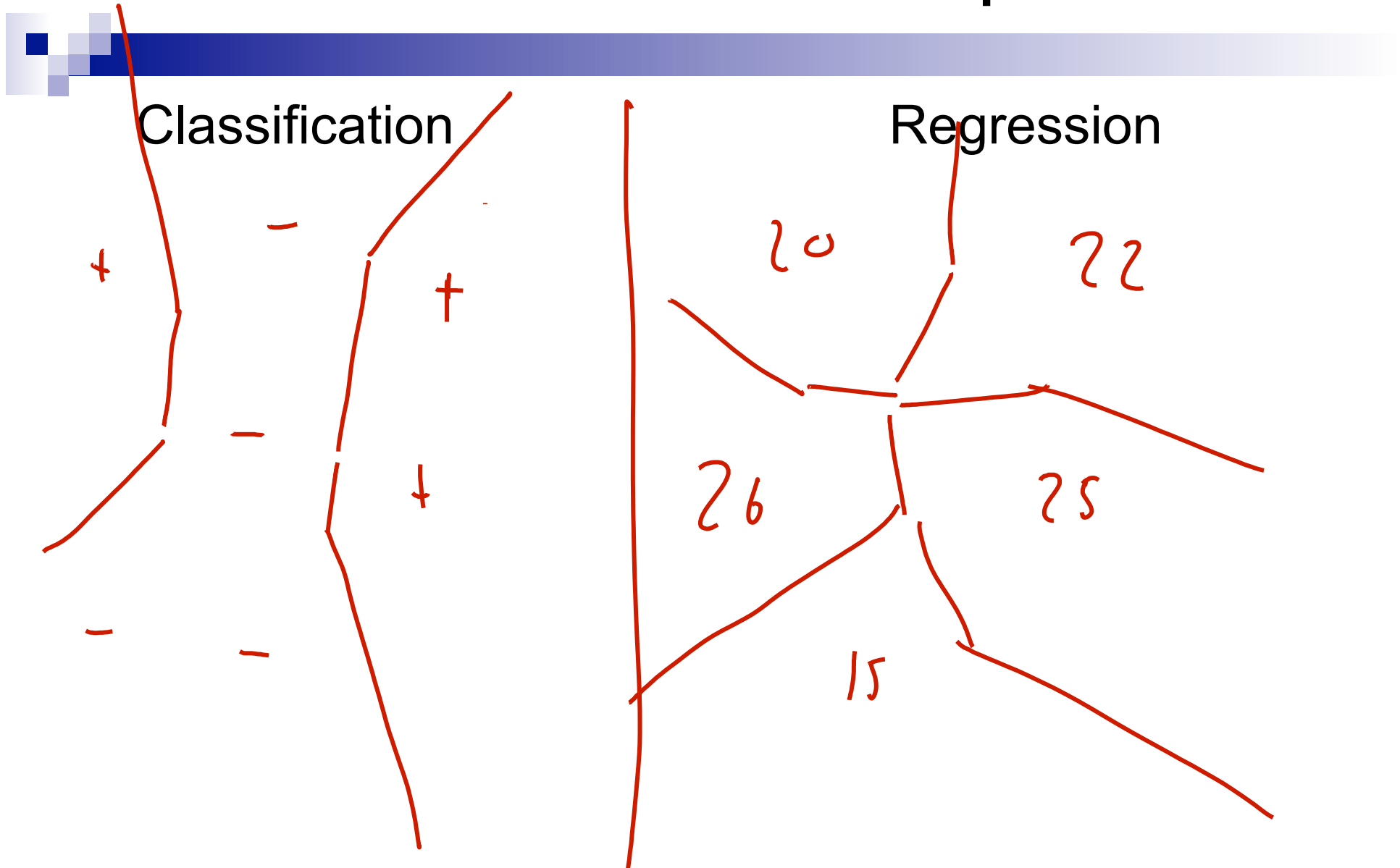
Just predict the same output as the nearest neighbor.

$$i = \underset{j}{\operatorname{argmin}} \|x^j - x^t\|$$

every point

$$\text{predict } g \equiv g^i$$

Multivariate 1-NN examples

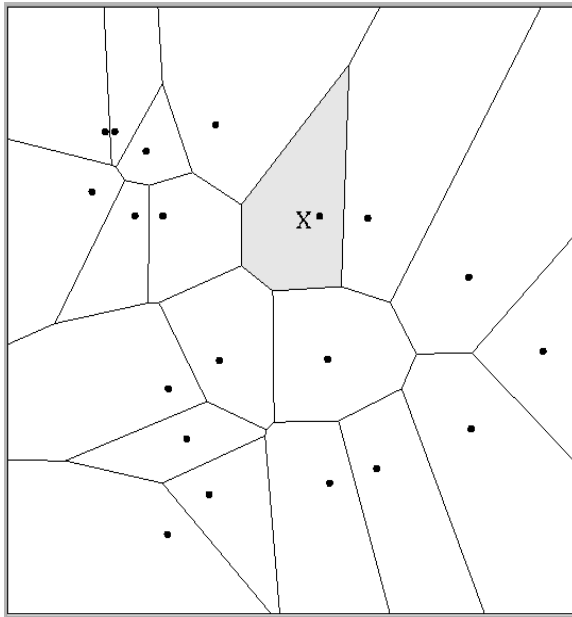


Multivariate distance metrics

Suppose the input vectors $\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^N$ are two dimensional:

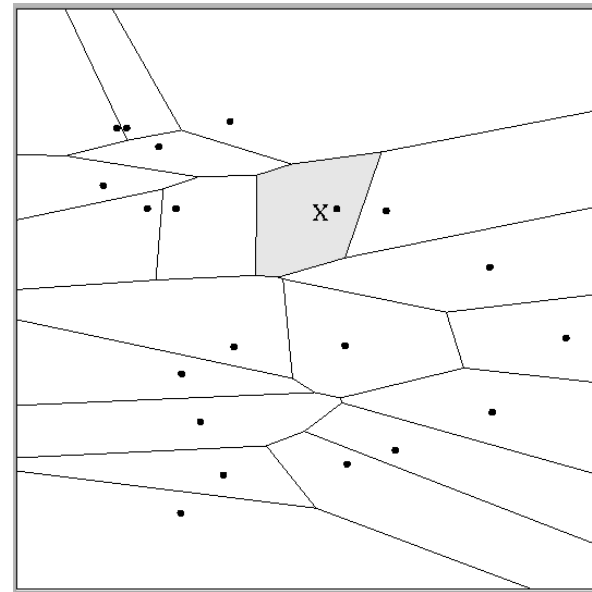
$$\mathbf{x}^1 = (x^1_1, x^1_2), \mathbf{x}^2 = (x^2_1, x^2_2), \dots, \mathbf{x}^N = (x^N_1, x^N_2).$$

One can draw the nearest-neighbor regions in input space.



$$Dist(\mathbf{x}^i, \mathbf{x}^j) = (x^i_1 - x^j_1)^2 + (x^i_2 - x^j_2)^2$$

$x_2 \uparrow$



$$Dist(\mathbf{x}^i, \mathbf{x}^j) = (x^i_1 - x^j_1)^2 + (3x^i_2 - 3x^j_2)^2$$

$x_1 \rightarrow$

The relative scalings in the distance metric affect region shapes

Euclidean distance metric

Or equivalently,

$$D(\mathbf{x}, \mathbf{x}') = \sqrt{\sum_i \sigma_i^2 (x_i - x'_i)^2}$$

where

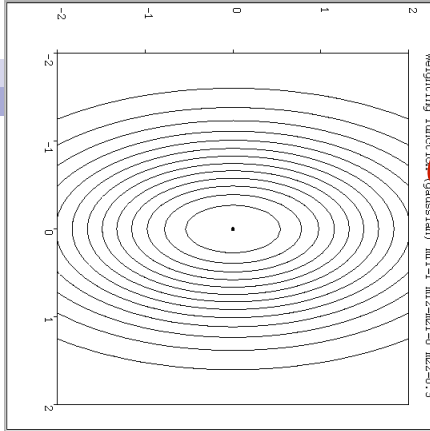
$$D(\mathbf{x}, \mathbf{x}') = \sqrt{(\mathbf{x} - \mathbf{x}')^T \Sigma (\mathbf{x} - \mathbf{x}')}$$

$$\Sigma = \begin{bmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & \sigma_N^2 \end{bmatrix}$$

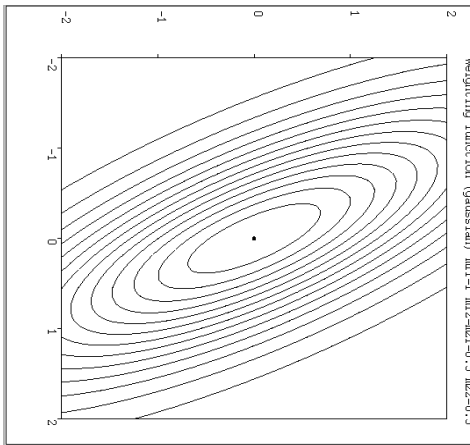
Other Metrics...

- Mahalanobis, Rank-based, Correlation-based,...

Notable distance metrics (and their level sets)



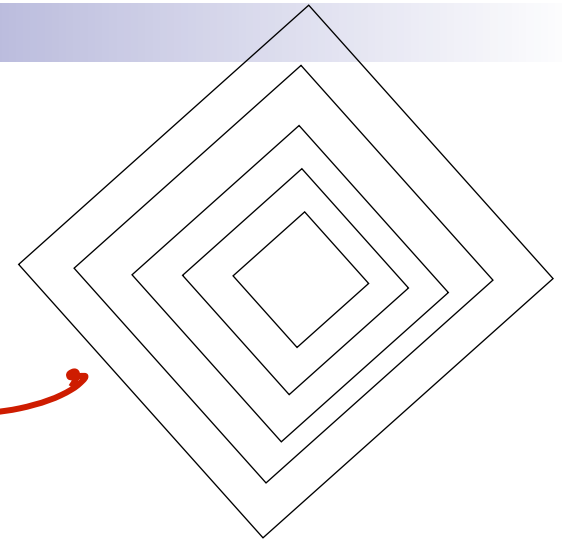
Scaled Euclidian (L_2)



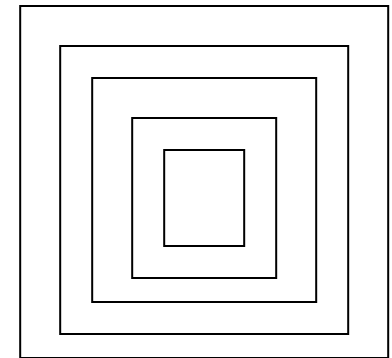
Mahalanobis (here, Σ on the previous slide is not necessarily diagonal, but is symmetric)

x_2 is 3 times more important than x_1

Learning distance metrics from data



L_1 norm (absolute)



L_∞ (max) norm

Consistency of 1-NN

- Consider an estimator f_n trained on n examples
 - e.g., 1-NN, neural nets, regression,...
- Estimator is *consistent* if true error goes to zero as amount of data increases

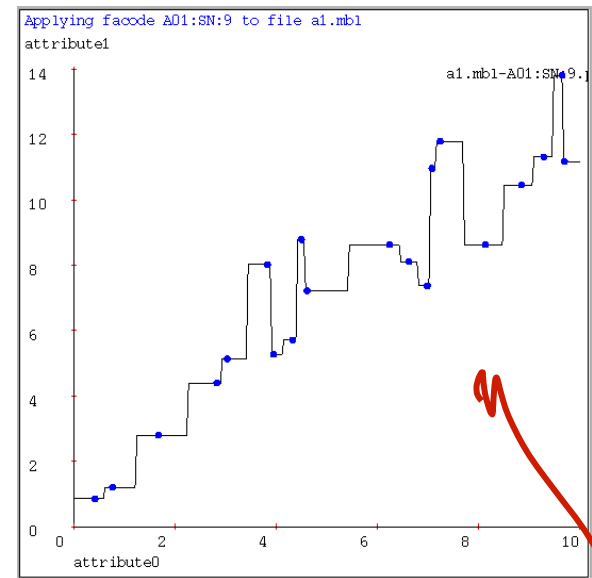
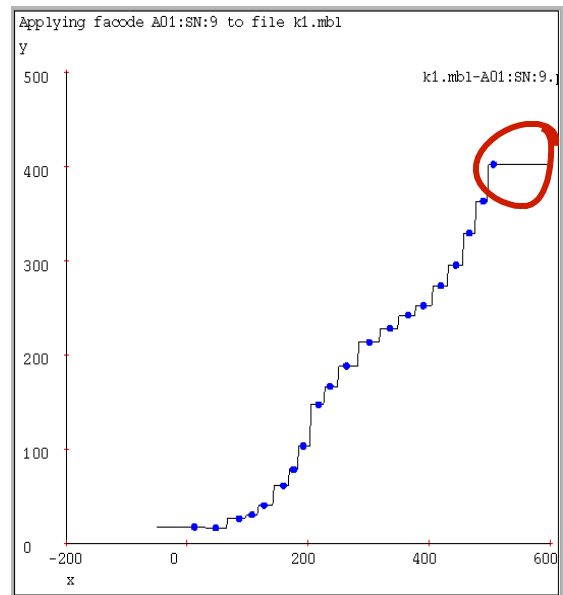
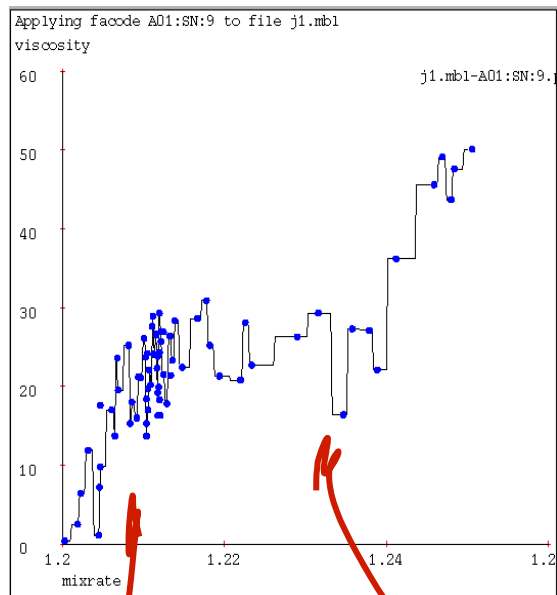
- e.g., for no noise data, consistent if:

$$\lim_{n \rightarrow \infty} MSE(f_n) = 0$$

- Regression is not consistent!
 - Representation bias
- **1-NN is consistent** (under some mild fineprint)

What about variance???

1-NN overfits?



noisy

1 NN function has too much flexibility

k-Nearest Neighbor

Four things make a memory based learner:

1. *A distance metric*
Euclidian (and many more)
2. *How many nearby neighbors to look at?*

k

1. *A weighting function (optional)*
Unused

2. *How to fit with the local points?*

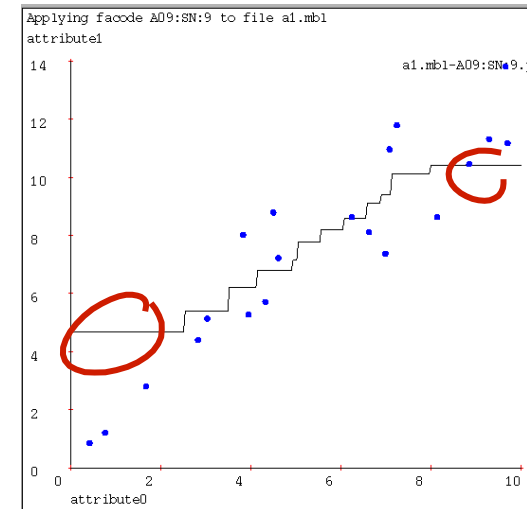
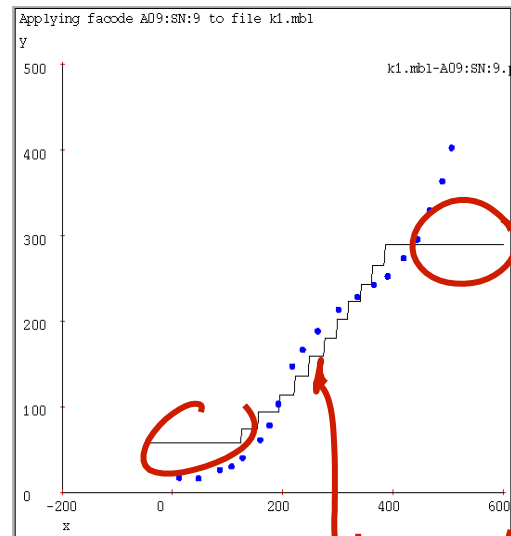
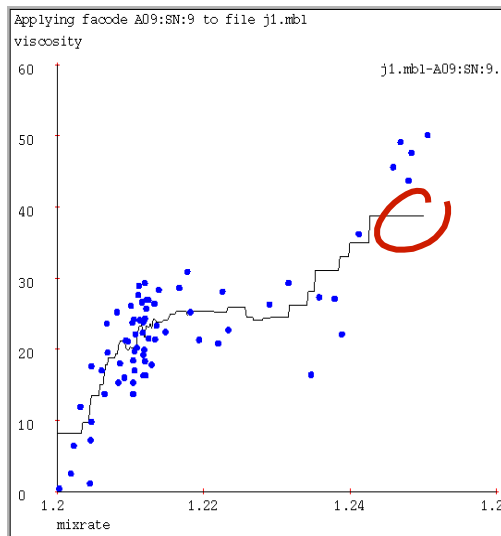
Just predict the average output among the k nearest neighbors.

regression $NN(x^*) \leftarrow k \text{ nearest neighbors}$ *classification*

$$\hat{y} = \frac{1}{k} \sum_{i \in NN(x^*)} y_i$$

majority vote over neighbors

k-Nearest Neighbor (here $k=9$)




Strange steps

K-nearest neighbor for function fitting smoothes away noise, but there are clear deficiencies.

What can we do about all the discontinuities that k-NN gives us?

Weighted k-NNs

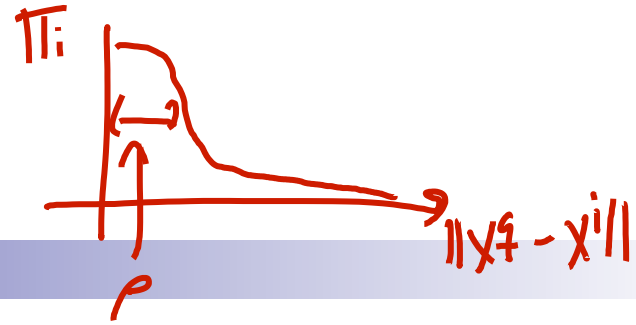
- Neighbors are not all the same


$$\hat{y} = \frac{\pi_1 y^1 + \pi_2 y^2 + \pi_3 y^3}{\pi_1 + \pi_2 + \pi_3}$$

π_i is some weight, e.g.,

$$\pi_i = \frac{1}{\|x^q - x^i\|}$$

Kernel regression



Four things make a memory based learner:

1. *A distance metric*

Euclidian (and many more)

2. *How many nearby neighbors to look at?*

All of them

3. *A weighting function (optional)*

$$\pi^i = \exp(-D(x^i, \text{query})^2 / \rho^2)$$

Nearby points to the query are weighted strongly, far points weakly. The ρ parameter is the **Kernel Width**. Very important.

4. *How to fit with the local points?*

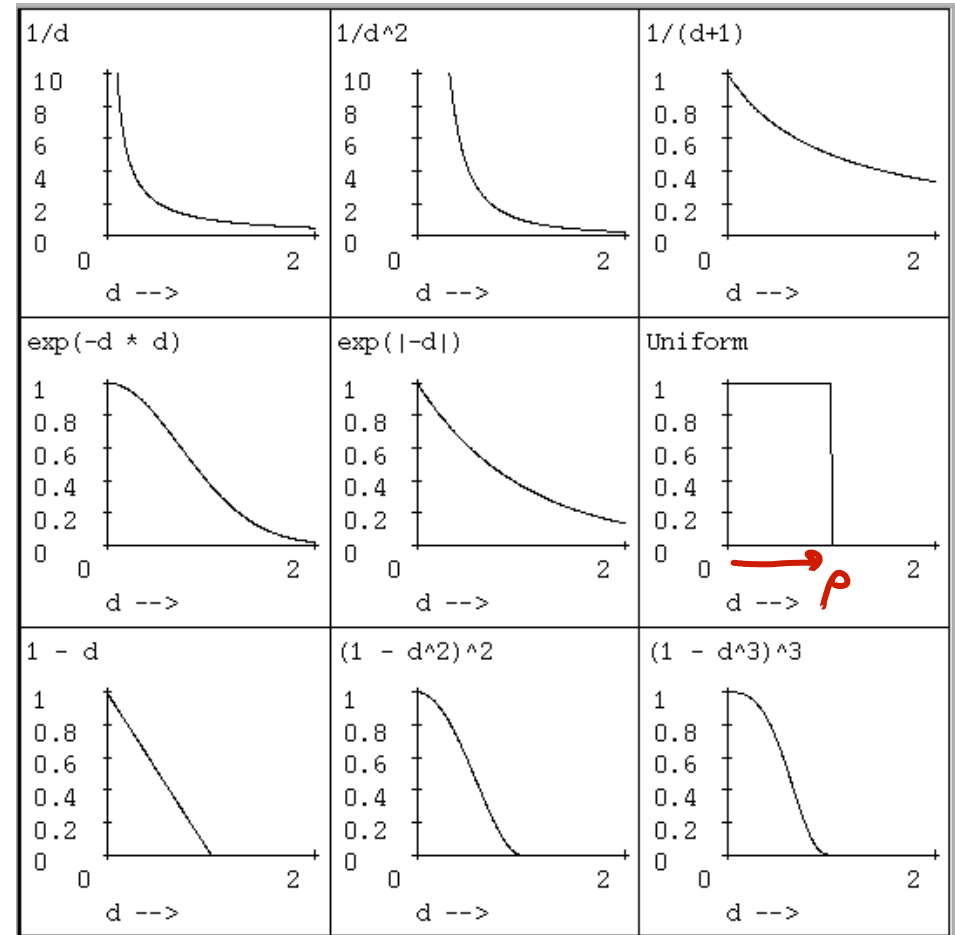
Predict the weighted average of the outputs:

$$\text{predict} = \Sigma \pi^i y^i / \Sigma \pi^i$$

classification
weighted majority

Weighting functions

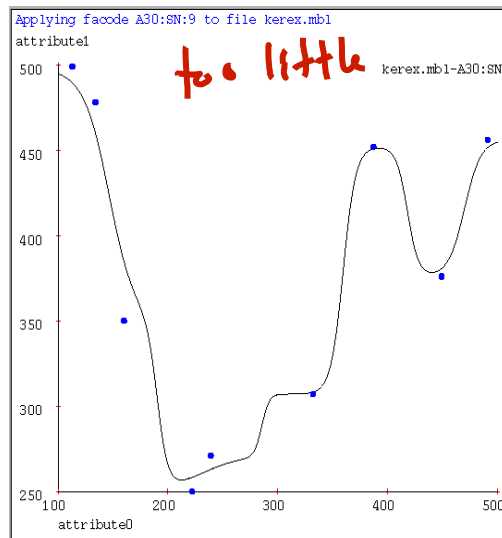
$$\pi^i = \exp(-D(x^i, \text{query})^2 / \rho^2)$$



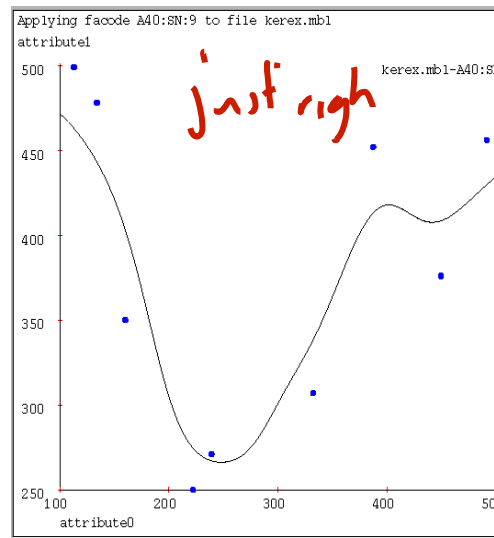
Typically optimize ρ using
gradient descent *or X-validation*

(Our examples use Gaussian)

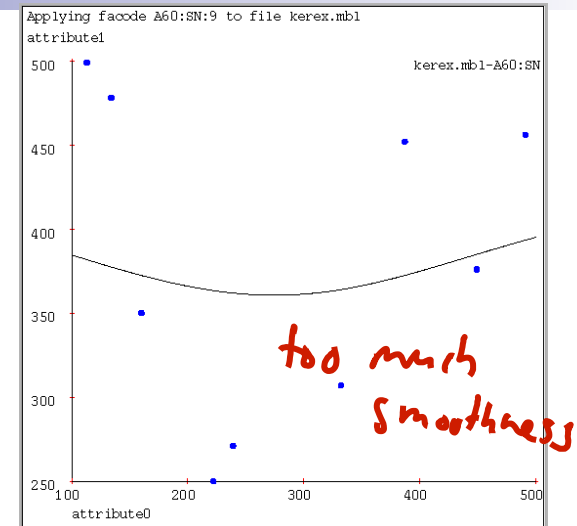
Kernel regression predictions



$\rho=10$



$\rho=20$



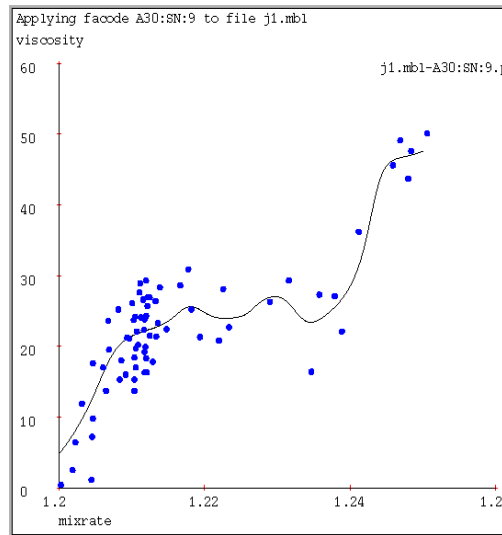
$\rho=80$

smoothness in output

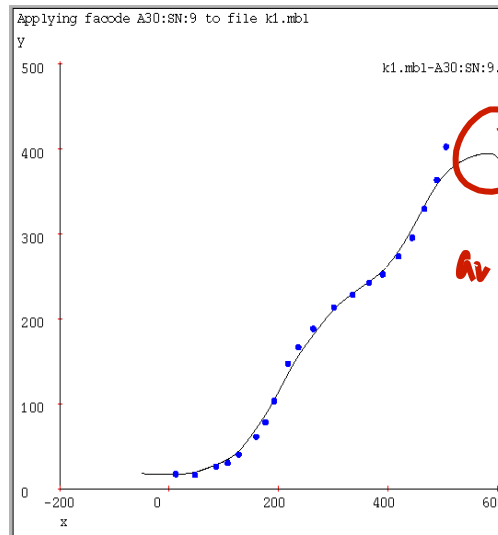
Increasing the kernel width ρ means further away points get an opportunity to influence you.

As $\rho \rightarrow \infty$, the prediction tends to the global average.

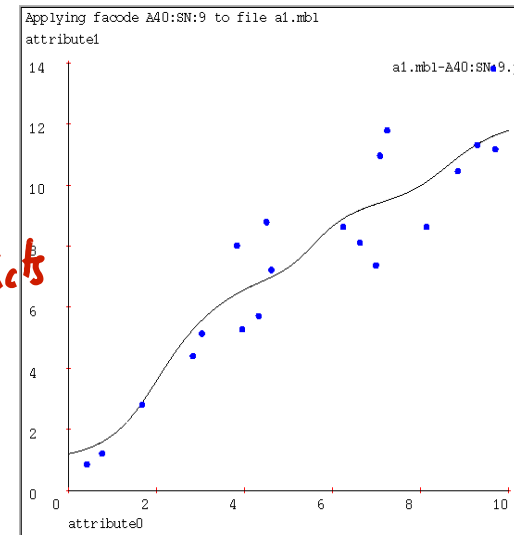
Kernel regression on our test cases



$\rho = 1/32$ of x-axis width.



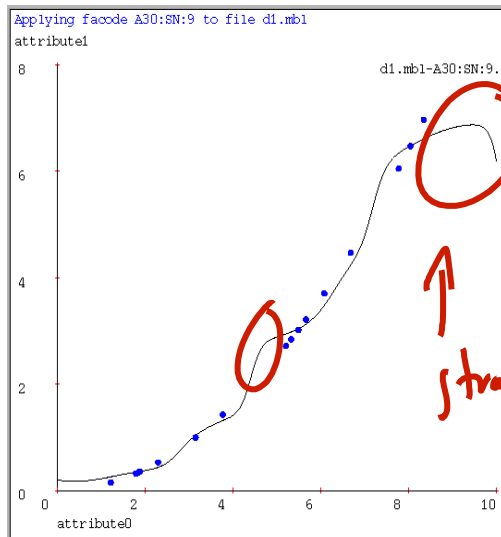
$\rho = 1/32$ of x-axis width.



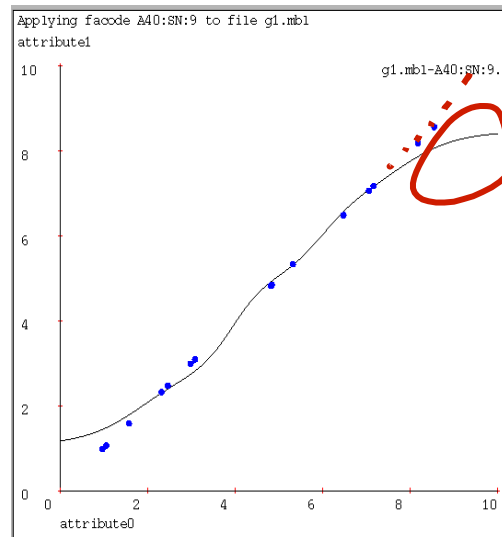
$\rho = 1/16$ axis width.

Choosing a good ρ is important. Not just for Kernel Regression, but for all the locally weighted learners we're about to see.

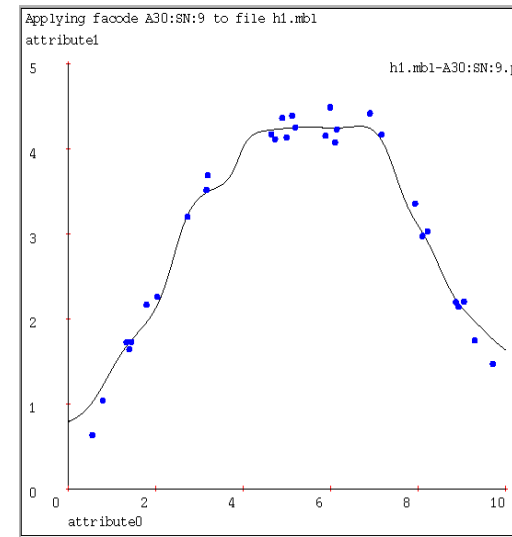
Kernel regression can look bad



$\rho = \text{Best.}$



$\rho = \text{Best.}$



$\rho = \text{Best.}$

Time to try something more powerful...

Locally weighted regression



Kernel regression:

Take a very very conservative function approximator called AVERAGING. Locally weight it.

Locally weighted regression:

Take a conservative function approximator called LINEAR REGRESSION. Locally weight it.

Locally weighted regression

- Four things make a memory based learner:

- A distance metric

Any

- How many nearby neighbors to look at?

All of them

- A weighting function (optional)

Kernels

- $\pi^i = \exp(-D(x^i, \text{query})^2 / \rho^2)$

- How to fit with the local points?

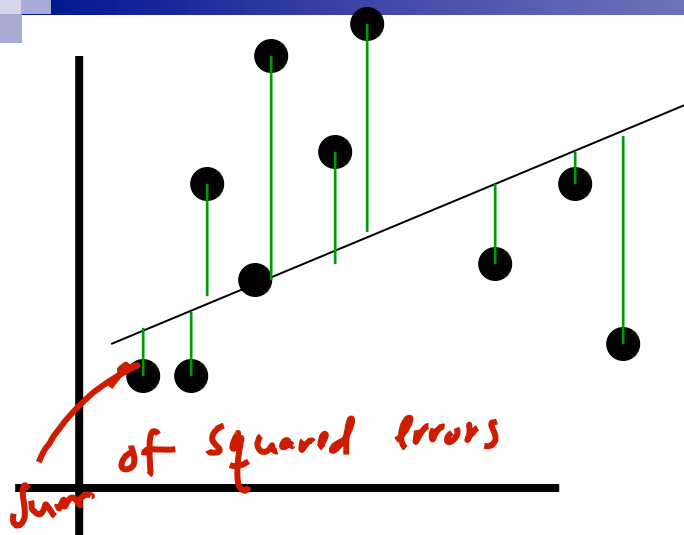
General weighted regression:

$$\hat{w}^q = \underset{w}{\operatorname{argmin}} \sum_{k=1}^N \pi_q^k \left(y^k - w^T x^k \right)^2$$

locally weighted (pointing to π_q^k)

regression function (pointing to $y^k - w^T x^k$)

How LWR works

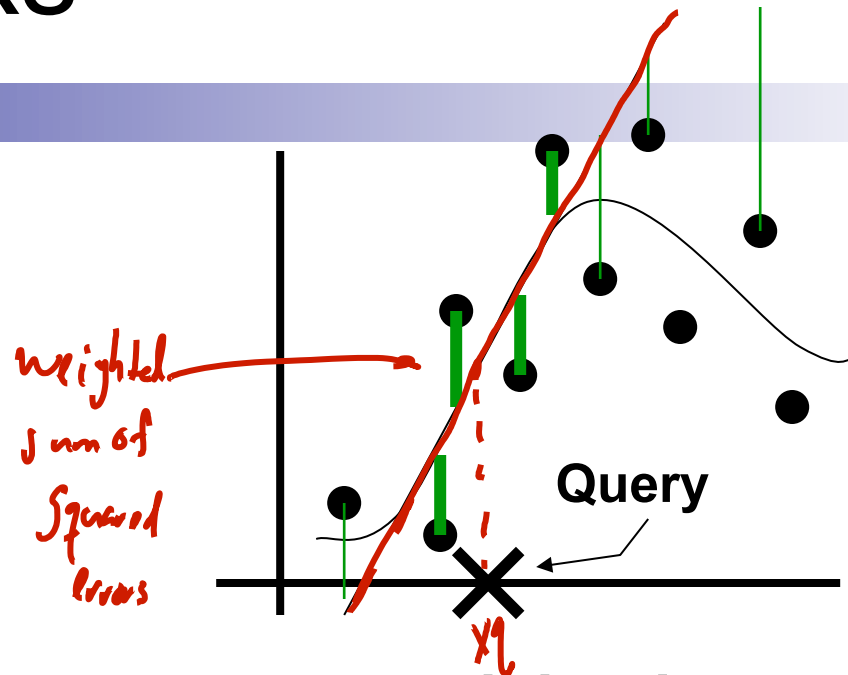


Linear regression

- Same parameters for all queries

$$\hat{w} = (X^T X)^{-1} X^T Y$$

Some matrix inversion



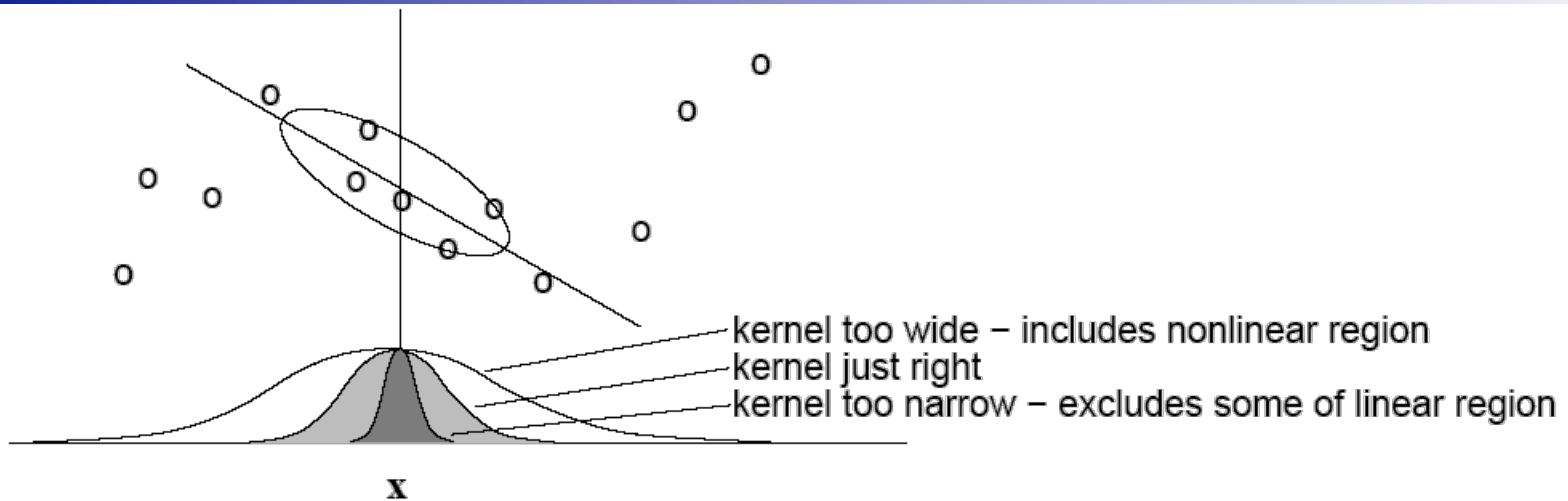
Locally weighted regression

- Solve weighted linear regression for each query

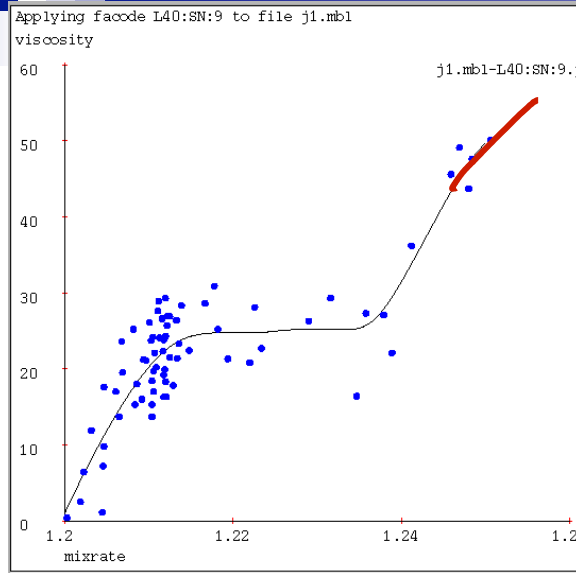
$$w^q = \left((\Pi X)^T \Pi X \right)^{-1} (\Pi X)^T \Pi Y$$

$$\Pi = \begin{pmatrix} \pi_1 & 0 & 0 & 0 \\ 0 & \pi_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \pi_n \end{pmatrix}$$

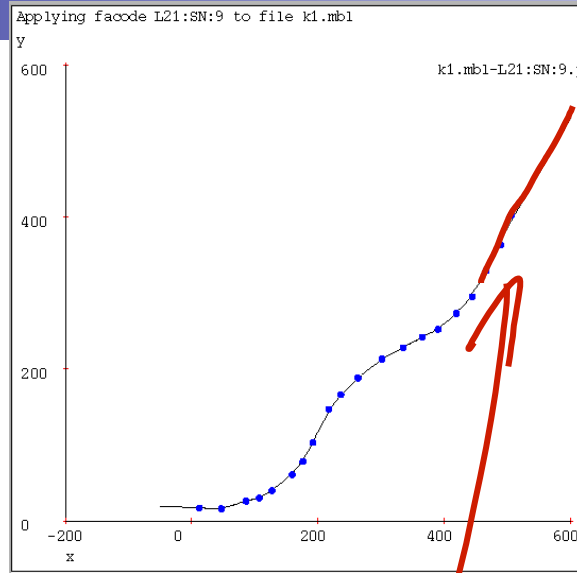
Another view of LWR



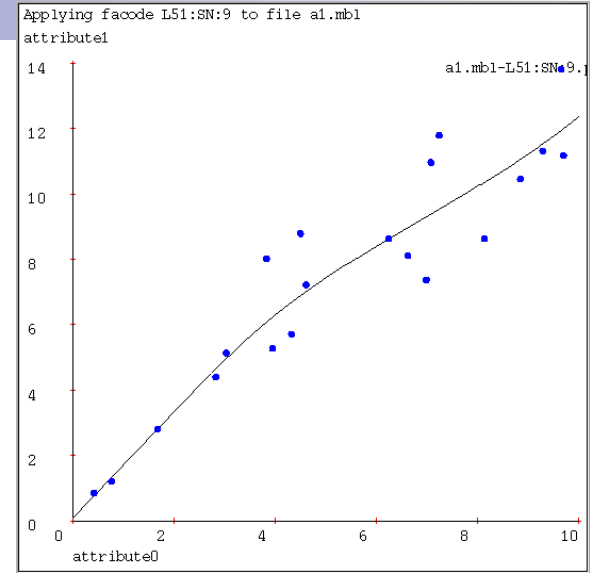
LWR on our test cases



$\rho = 1/16$ of x-axis width.



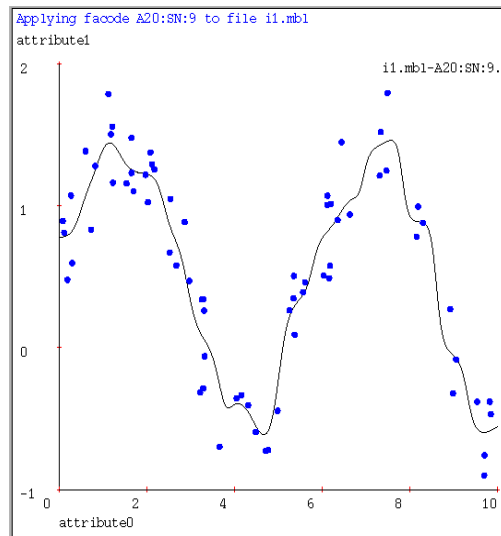
$\rho = 1/32$ of x-axis width.



$\rho = 1/8$ of x-axis width.

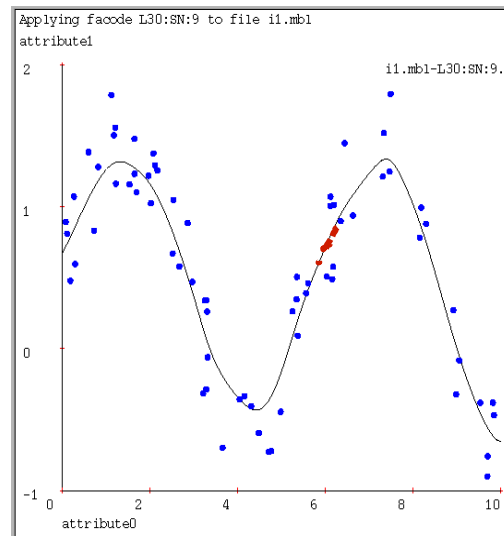
*much better fit,
but more expensive computationally*

Locally weighted polynomial regression



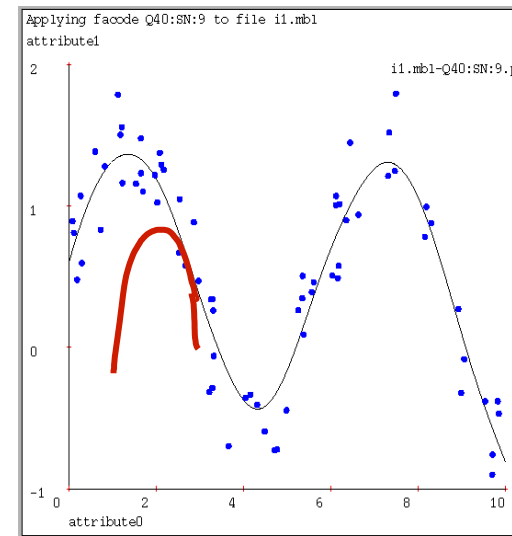
Kernel Regression
Kernel width ρ at optimal level.

$\rho = 1/100$ x-axis



LW Linear Regression
Kernel width ρ at optimal level.

$\rho = 1/40$ x-axis



LW Quadratic Regression
Kernel width ρ at optimal level.

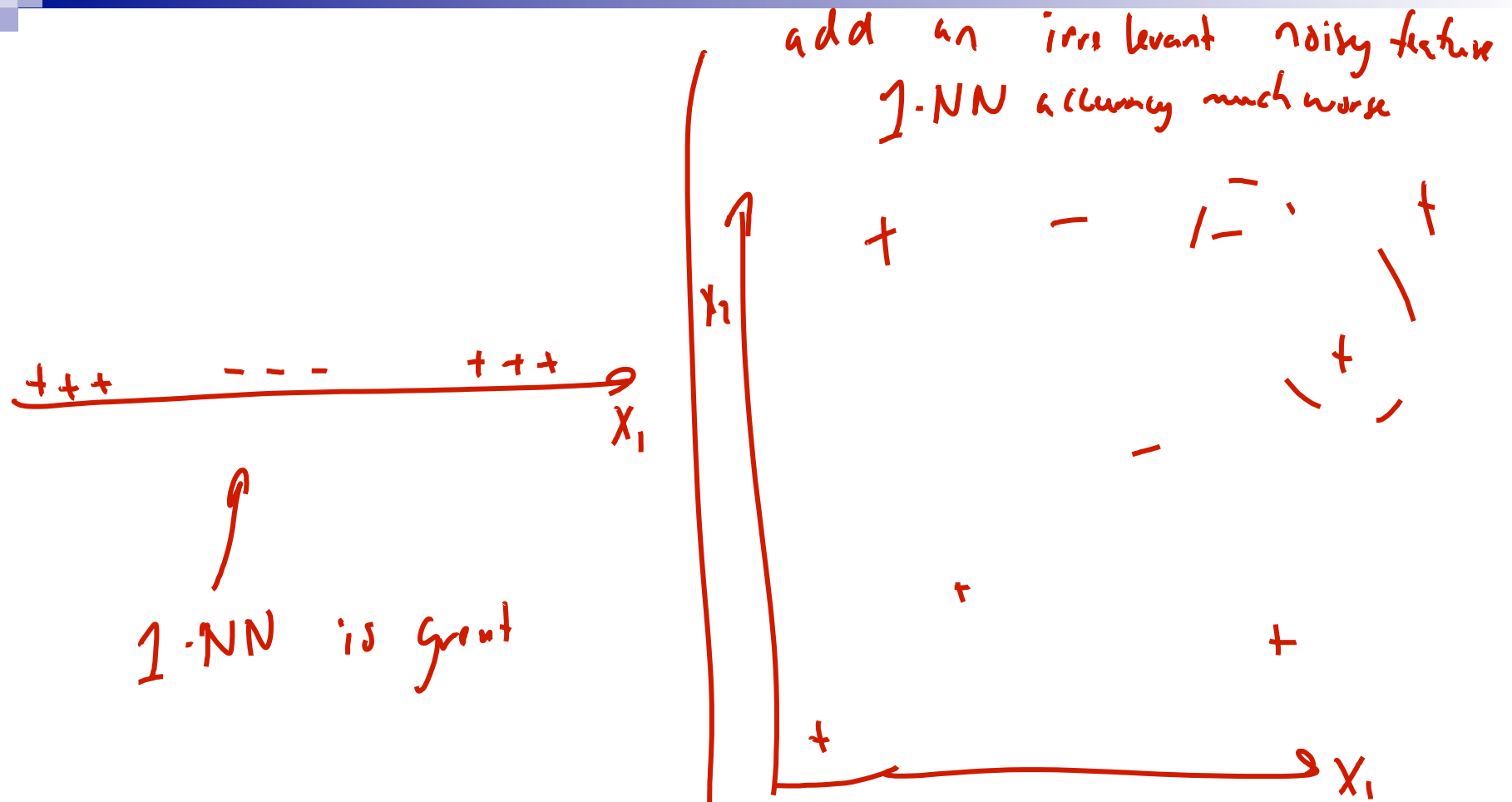
$\rho = 1/15$ x-axis

Local quadratic regression is easy: just add quadratic terms to the X matrix. As the regression degree increases, the kernel width can increase without introducing bias.

Curse of dimensionality for instance-based learning

- Must store and retrieve all data!
 - Most real work done during testing
 - For every test sample, must search through all dataset – very slow!
 - There are (sometimes) fast methods for dealing with large datasets
- Instance-based learning often poor with noisy or irrelevant features

Curse of the irrelevant feature



What you need to know about instance-based learning

■ k-NN

- Simplest learning algorithm
- With sufficient data, very hard to beat “strawman” approach
- Picking k ?

■ Kernel regression

- Set k to n (number of data points) and optimize weights by gradient descent
- Smoother than k-NN

■ Locally weighted regression

- Generalizes kernel regression, not just local average

■ Curse of dimensionality

- Must remember (very large) dataset for prediction
- Irrelevant features often killers for instance-based approaches

Decision boundary
1 NN
Voronoi diagram

decision tree boundary
 x_1

axis aligned splits

Acknowledgment



- This lecture contains some material from Andrew Moore's excellent collection of ML tutorials:
 - <http://www.cs.cmu.edu/~awm/tutorials>