Decision Trees

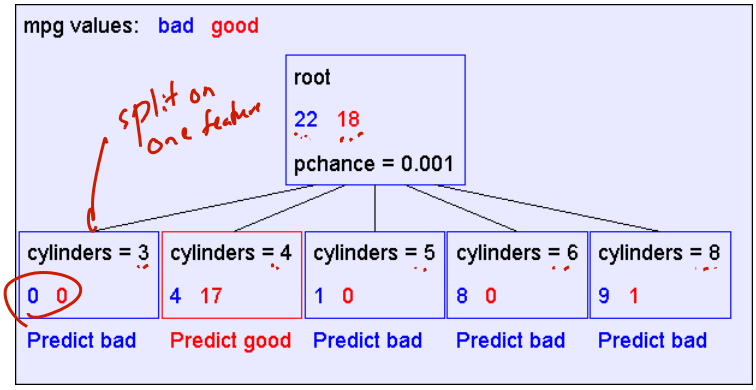
Machine Learning – CSE546

Carlos Guestrin

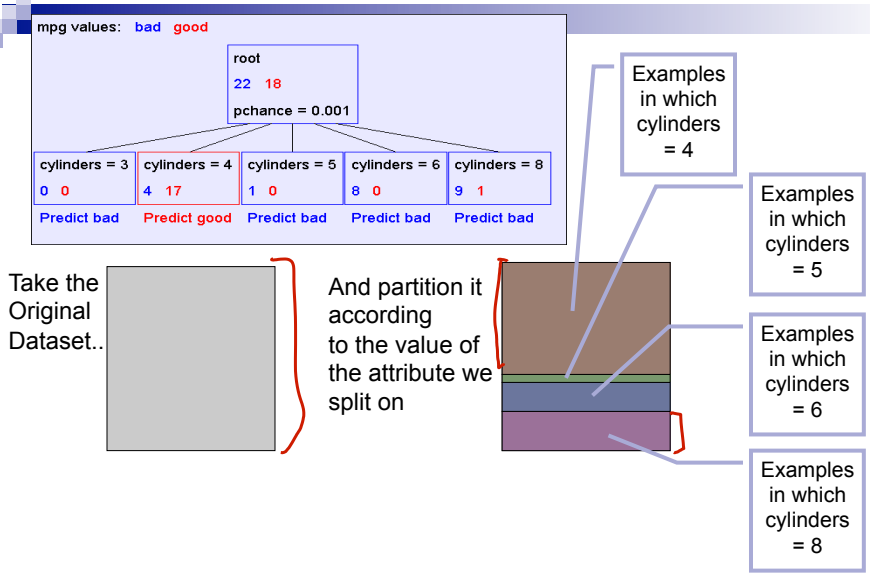
University of Washington

October 16, 2013

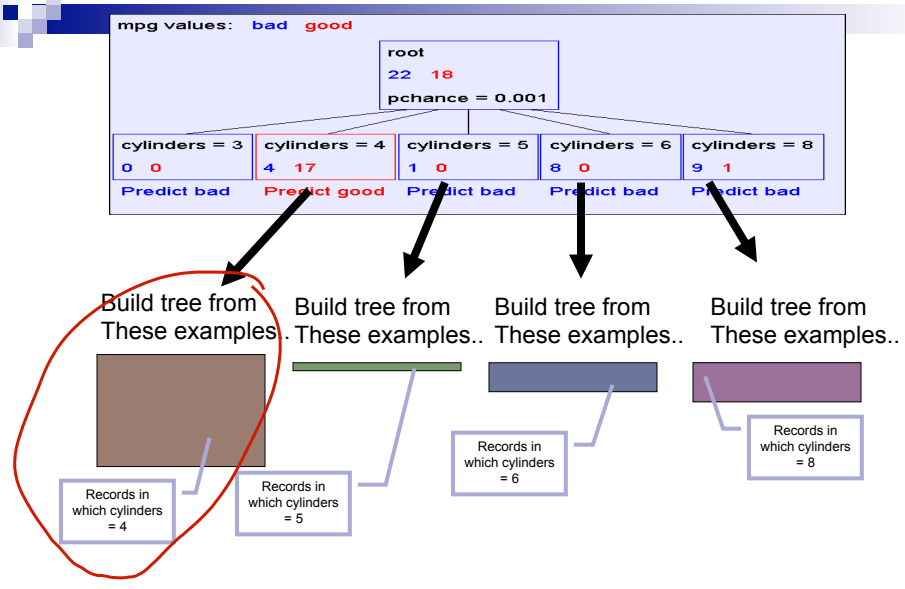
A Decision Stump



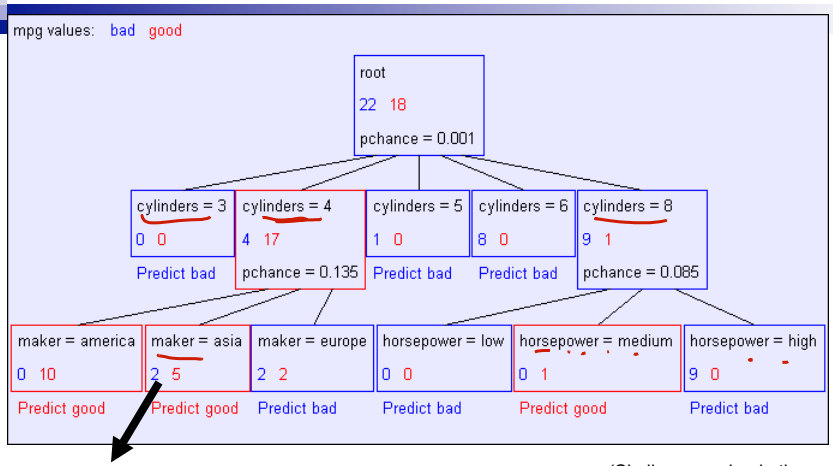
Recursion Step



Recursion Step



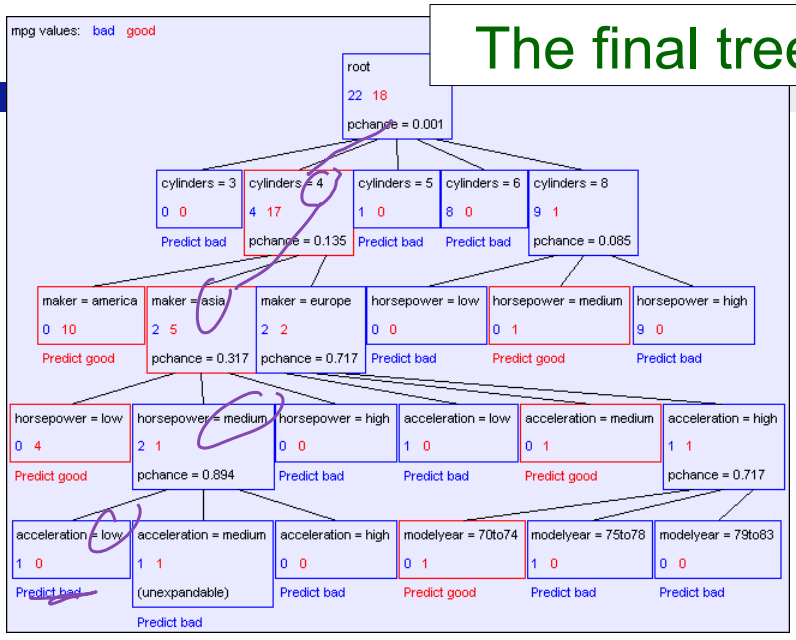
Second level of tree



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

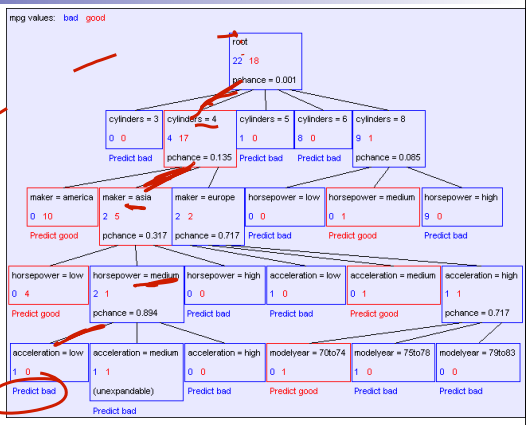
(Similar recursion in the other cases)

The final tree



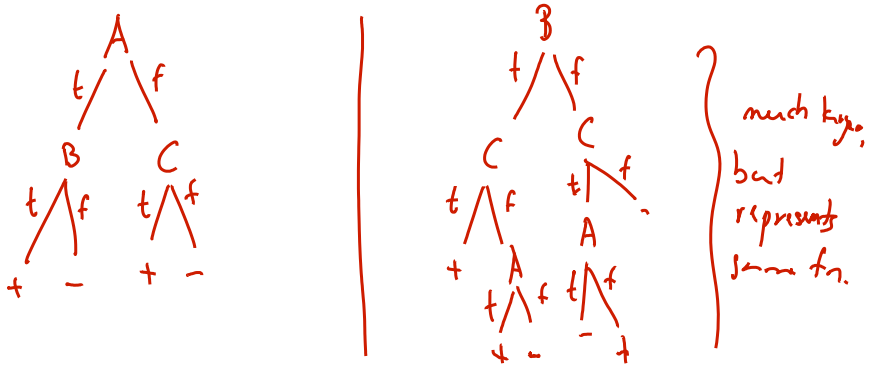
Classification of a new example

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



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))

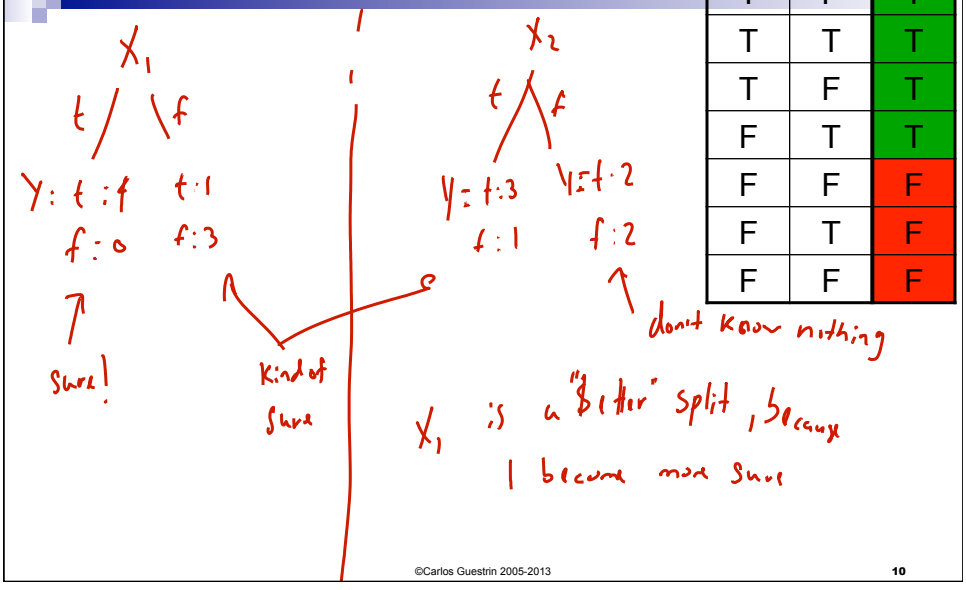


Learning decision trees is hard!!!

- 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 ← on subset of data consistent with each leaf

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



Measuring uncertainty

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

After the split

more sure, less uncertainty

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

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

uniform bad

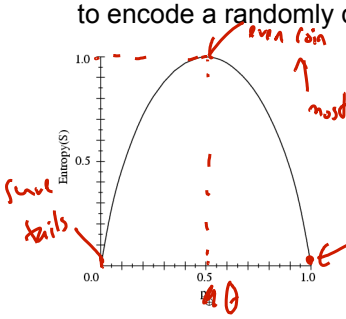
Entropy

Entropy $H(Y)$ 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)



even coin

most uncertain

Y is binary

P(y) = θ

sure tails

sure heads

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

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

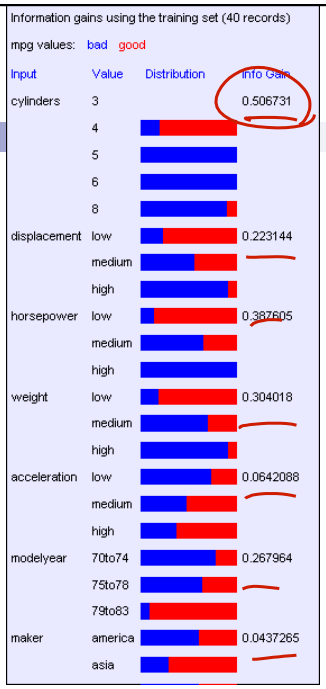
- Information gain is difference $IG(X) = H(Y) - H(Y \mid X)$

Learning decision trees

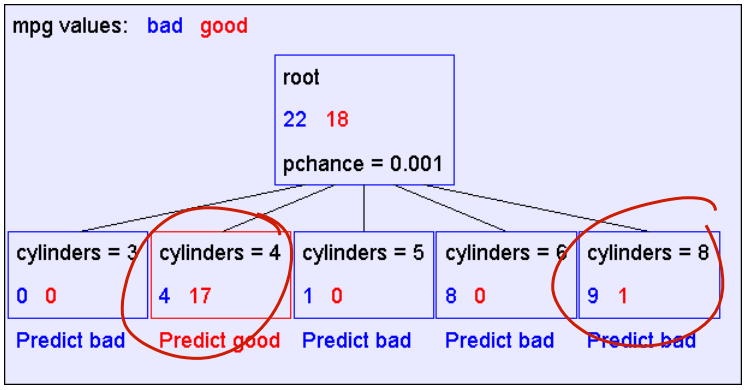
- Start from empty decision tree
- Split on **next best attribute (feature)**
  - Use, for example, information gain to select attribute
  - Split on $\arg\max_i IG(X_i) = \arg\max_i H(Y) - H(Y \mid X_i)$
- Recurse

Suppose we want to predict MPG

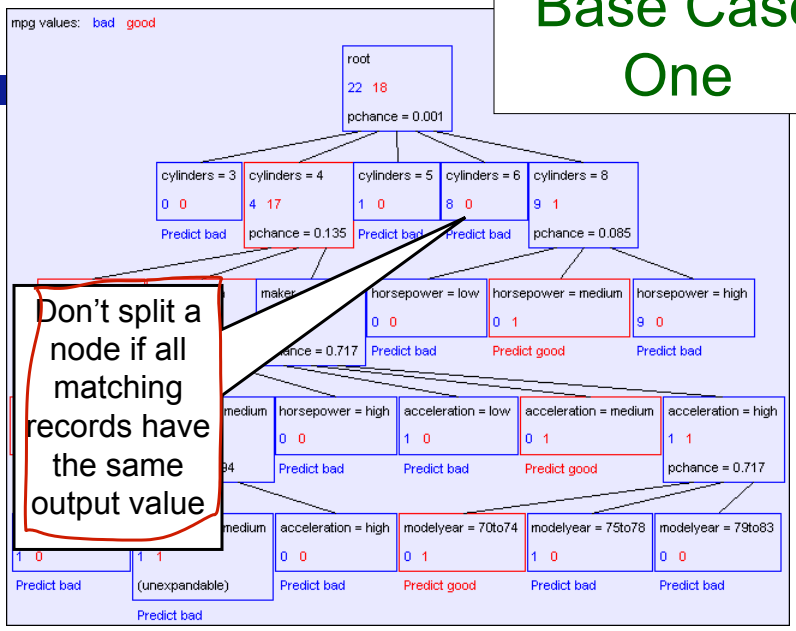
Look at all the information gains…



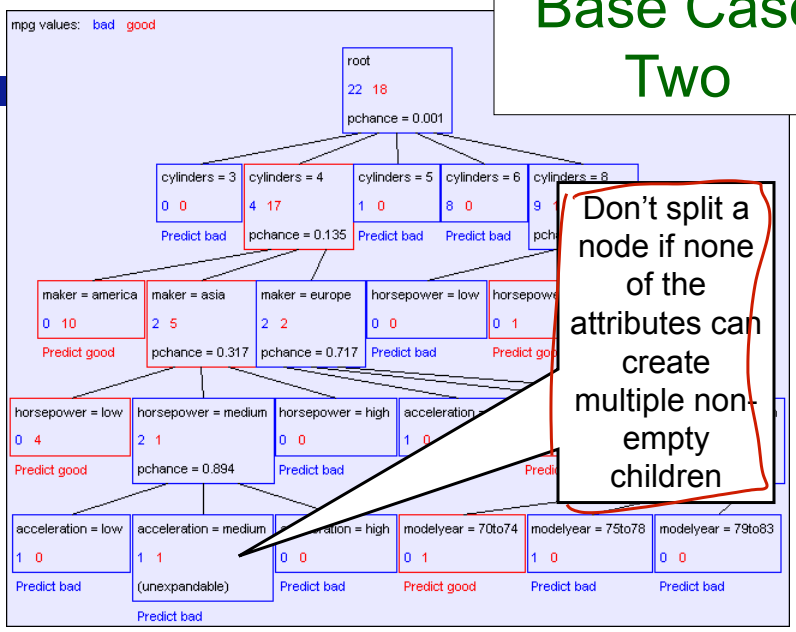
A Decision Stump



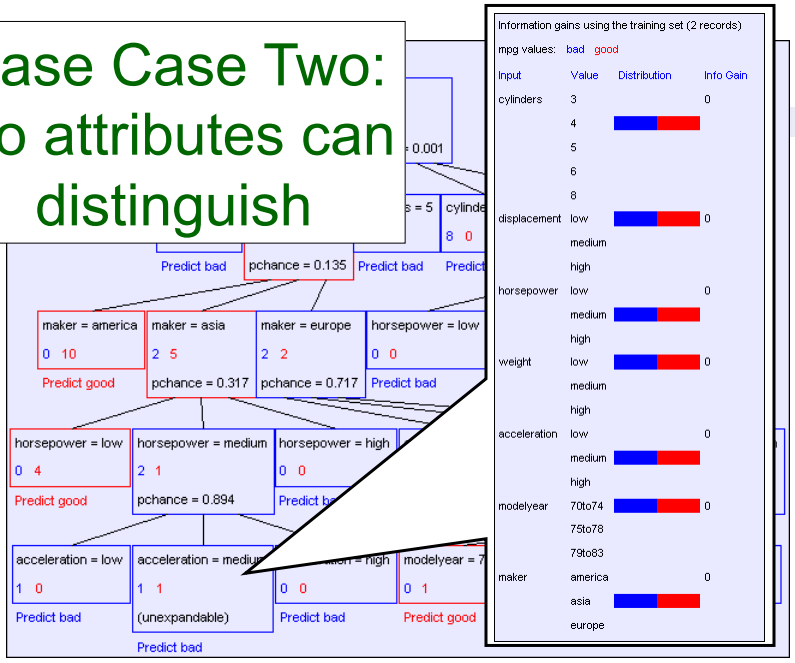
Base Case One



Base Case Two



Base Case Two: No attributes can distinguish



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

Proposed Base Case 3:

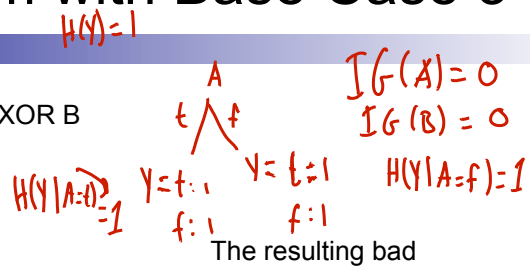
If all attributes have zero information gain then don't recurse

•*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 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		

The resulting bad decision tree:

y values: 0 1

root

2 2

Predict 0

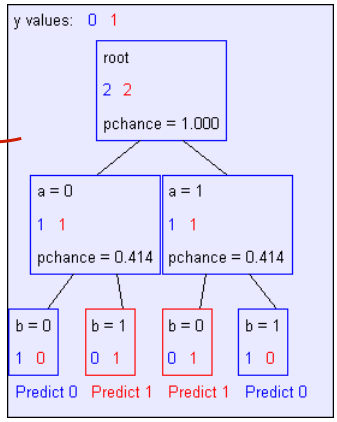
If we omit Base Case 3:

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

y = a XOR b

The resulting decision tree:

low info gain is not a good stopping criteria



perfect classification

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

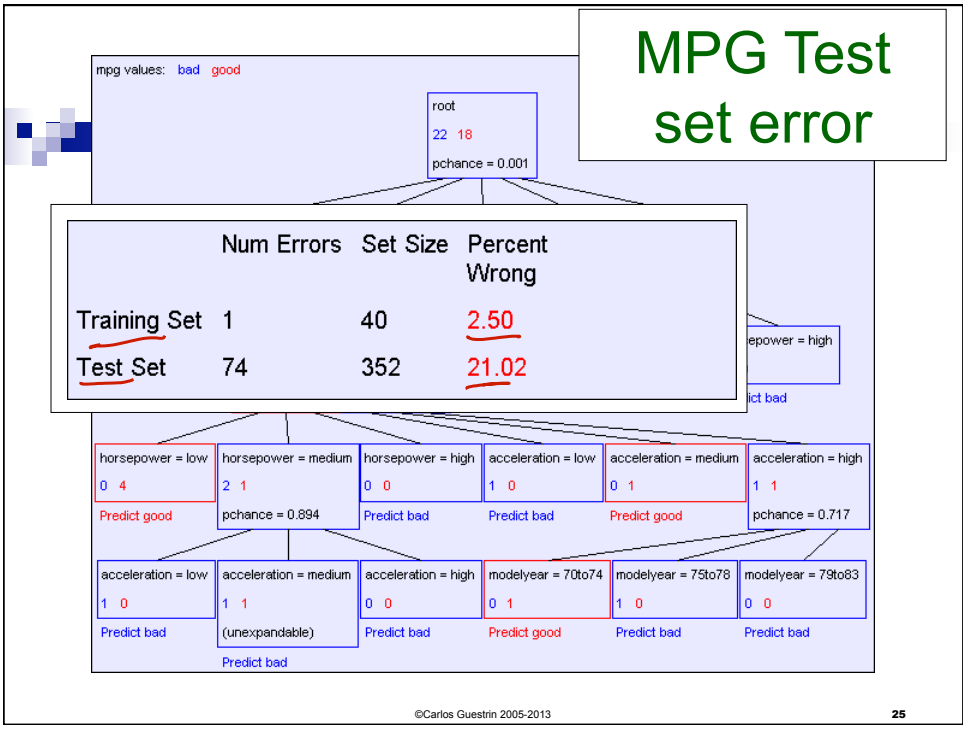
    BuildTree(DS_i,*Output*)

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

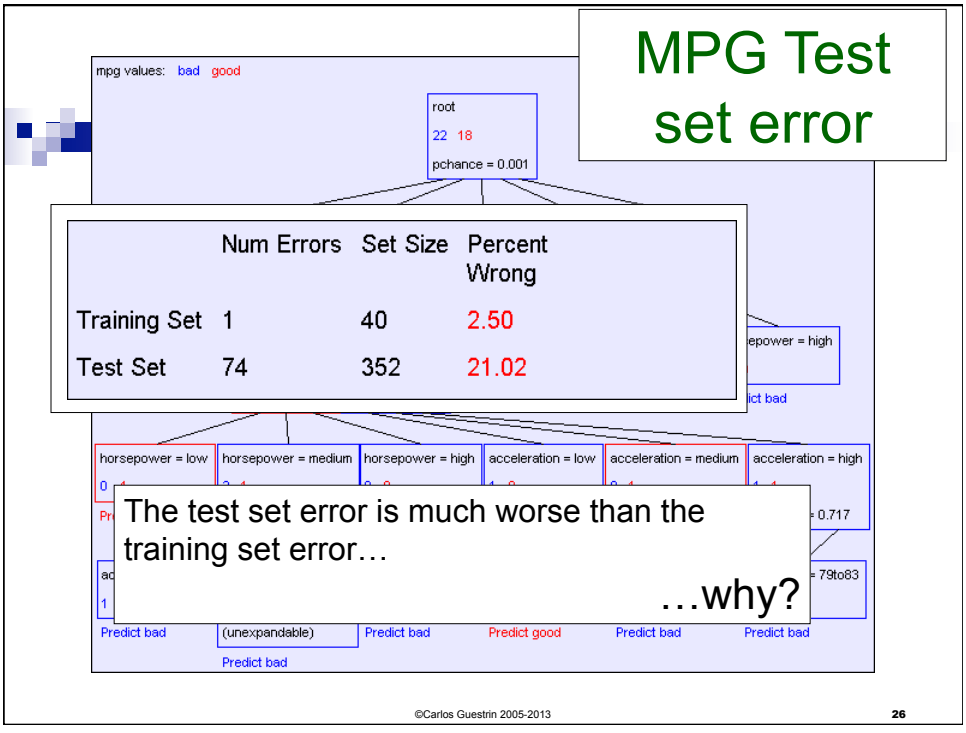
go for max

prune to simplify

MPG Test set error



MPG Test set error



Decision trees & Learning Bias

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	europe
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	europe
bad	5	medium	medium	medium	medium	75to78	europe

Suppose "no label noise":

$\nexists$ x^1, y^1 label $\&$ x^2, y^2

$x^1 = x^2$, but $y^1 \neq y^2$

What can we say about simple decision trees

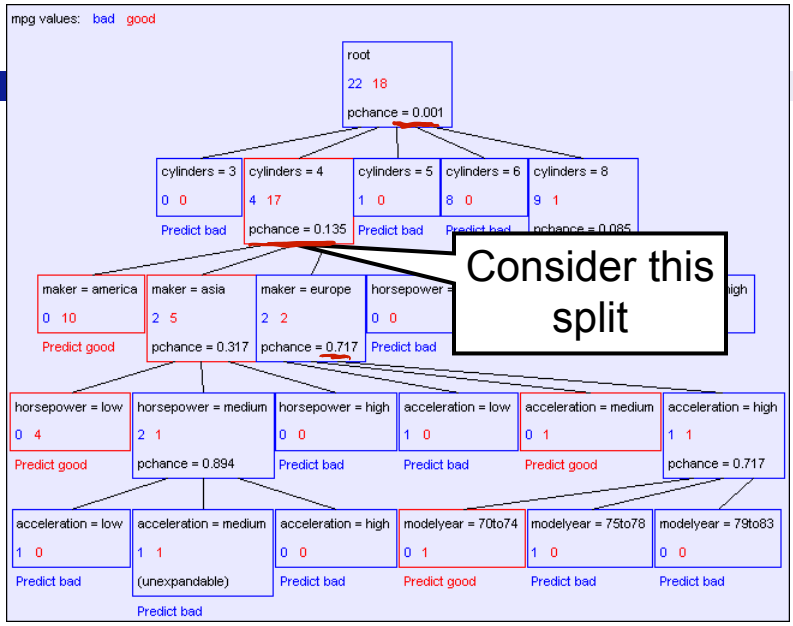
$\Rightarrow$ $\emptyset$ train error

$\Rightarrow$ can lead to overfitting

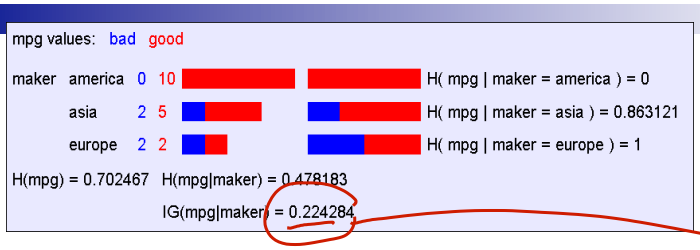
$\Rightarrow$ "bias" $\sim \emptyset$ $\Rightarrow$ variance high

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 : 1, 2, 3, … D penalty term on size
  - Fixed number of leaves
  - Or something smarter…

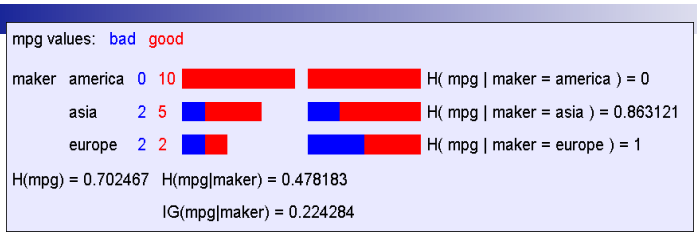


A chi-square test



- 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?

A chi-square test



- 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…)

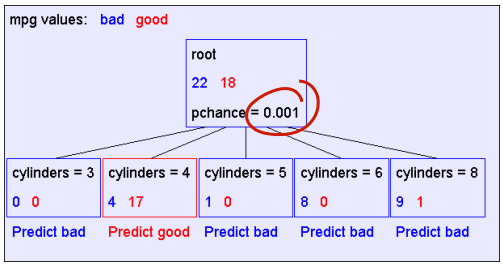
Using Chi-squared to avoid overfitting

- 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 $p_{chance} > MaxPchance$
  - Continue working you 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:

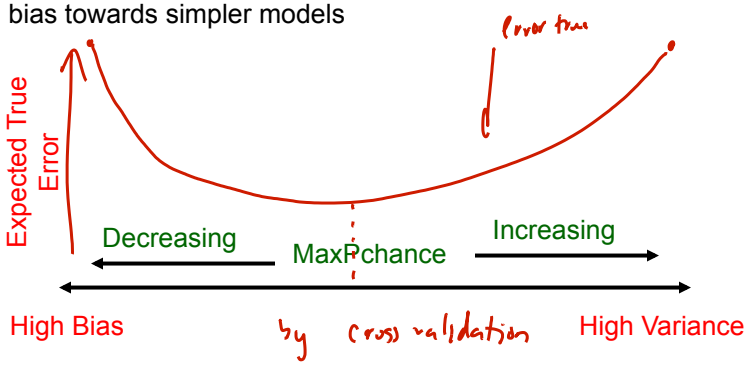


Note the improved test set accuracy compared with the unpruned tree

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

MaxPchance

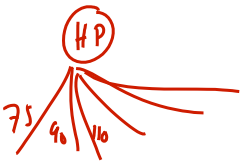
- Technical note MaxPchance is a regularization parameter that helps us bias towards simpler models

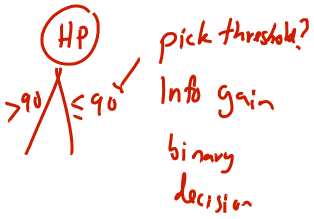


Real-Valued inputs

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

mpg	cylinders	displacemen	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	europe
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	europe
bad	5	131	103	2830	15.9	78	europe



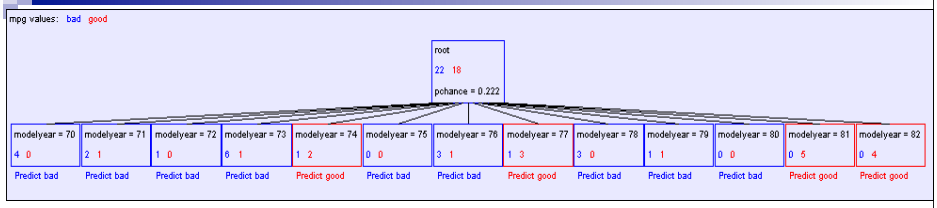


Infinite number of possible split values!!!

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_j
  - One branch: $X_j < t$
  - Other branch: $X_j \geq t$

Choosing threshold split

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

100 : 105 : 125

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)* =

$$H(Y|X < t)\, P(X < t) + H(Y|X >= t)\, P(X >= t)$$

- *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

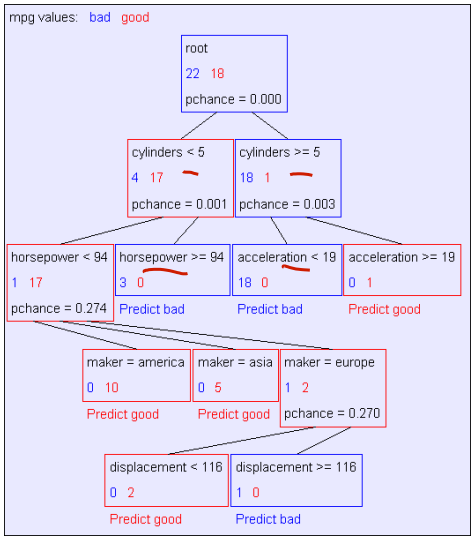
Example with MPG

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		
	europe		

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

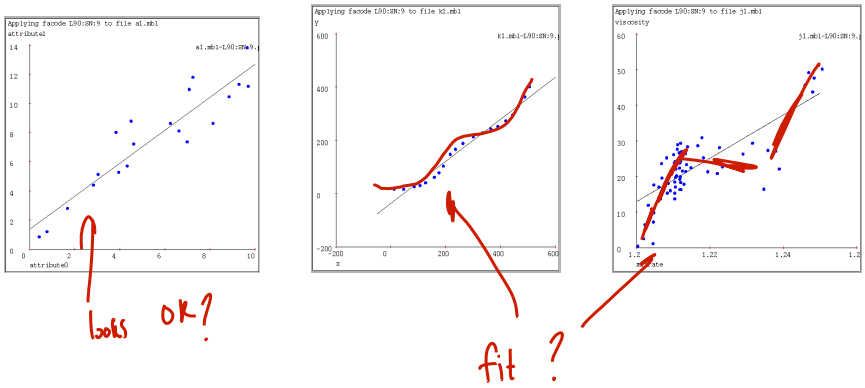
Machine Learning – CSE546

Carlos Guestrin

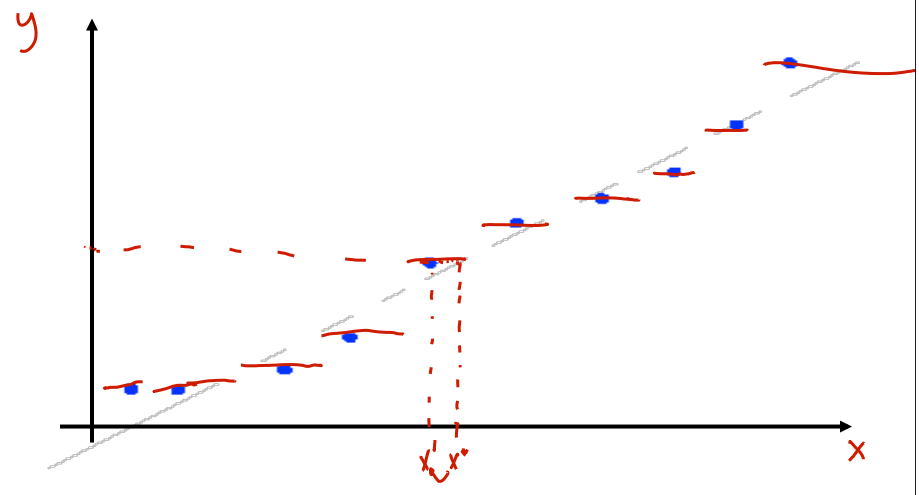
University of Washington

October 21, 2013

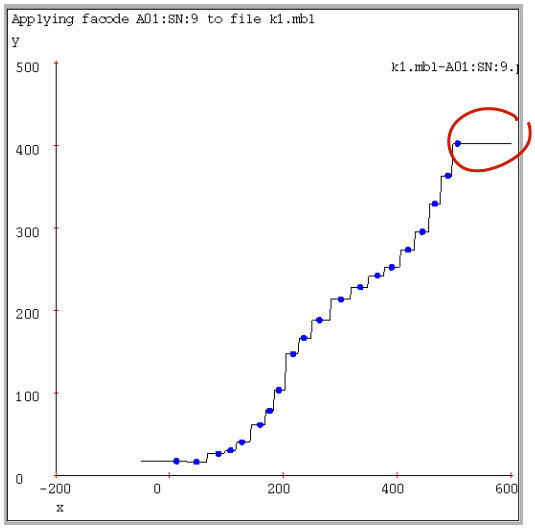
Why not just use Linear Regression?



Using data to predict new data



Nearest neighbor



Univariate 1-Nearest Neighbor

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

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

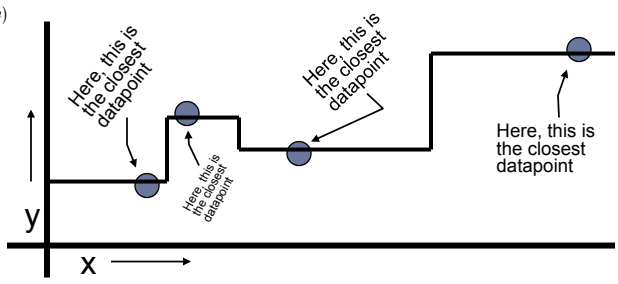
Nearest Neighbor:

1. Find the closest x_i in our set of datapoints

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

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

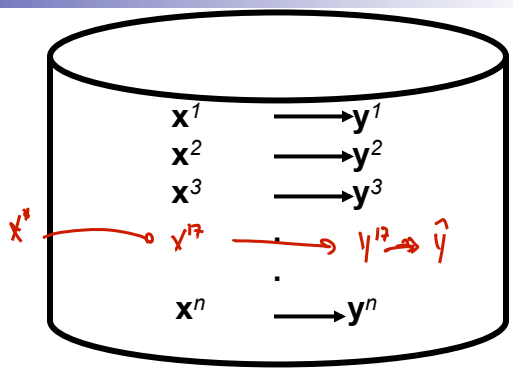
Here's a dataset with one input, one output and four datapoints.



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:

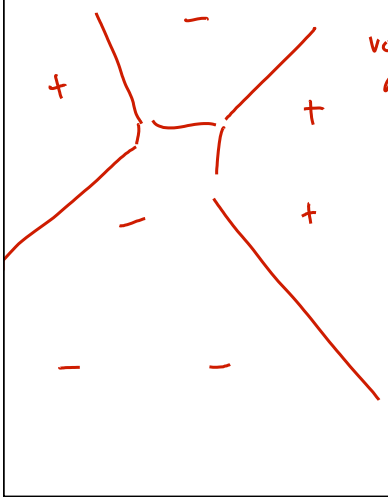
1. *A distance metric*
   Euclidian (and many more)
2. *How many nearby neighbors to look at?*
   One
3. *A weighting function (optional)*
   Unused
4. *How to fit with the local points?*
   Just predict the same output as the nearest neighbor.

$i = \arg\min_j \|x^j - x^q\|$

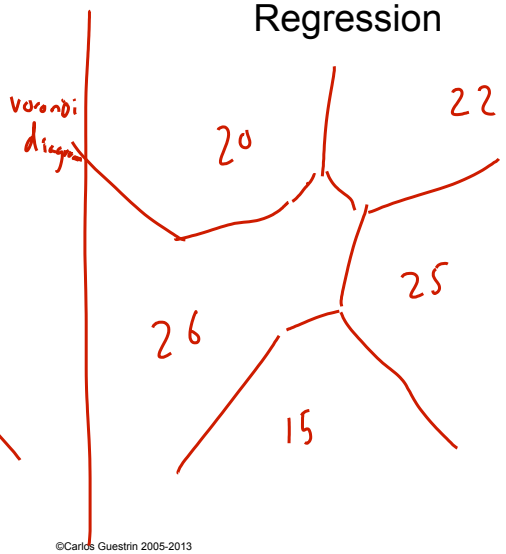
predict $\hat{y} = y^i$

Multivariate 1-NN examples

Classification



Regression

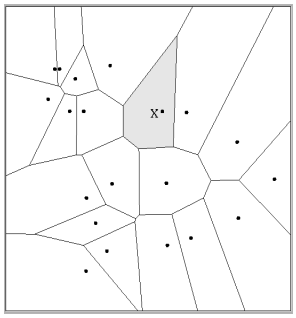


Multivariate distance metrics

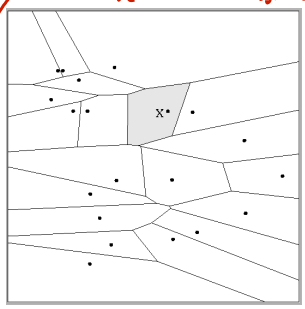
Suppose the input vectors $x^1, x^2, \ldots x^N$ are two dimensional:

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

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



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



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

The relative scalings in the distance metric affect region shapes

Euclidean distance metric

Or equivalently,

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

where

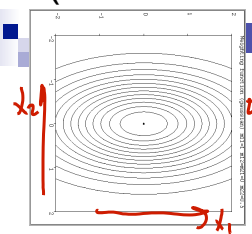
$$D(x, x') = \sqrt{(x - x')^T \sum (x - x')}$$

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

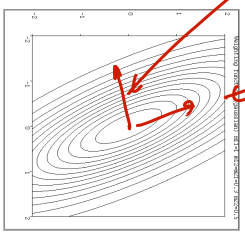
Other Metrics…

- Mahalanobis, Rank-based, Correlation-based,…

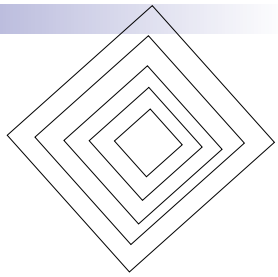
Notable distance metrics (and their level sets)



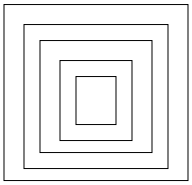
Scaled Euclidian (L_2)



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



L_1 norm (absolute)



L1 *(max)* norm

Consistency of 1-NN

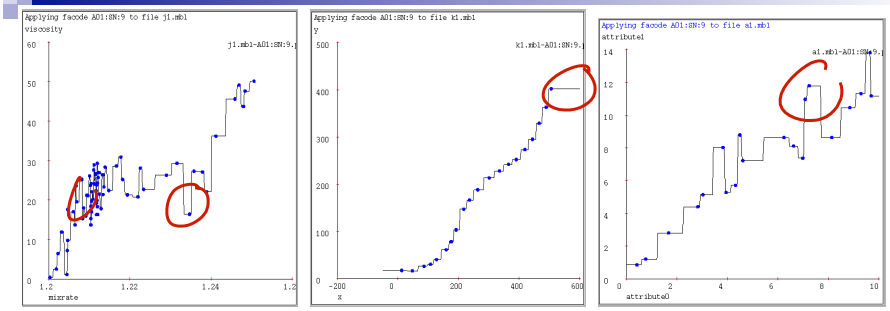
- Consider an estimator f_n trained on n examples
  - e.g., 1-NN, neural nets, regression,...
- Estimator is *consistent* if true error goes to zero as amount of data increases
  - e.g., for no noise data, consistent if:

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

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

What about variance???

1-NN overfits?



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.

$NN(x_q) \leftarrow$ k nearest neighbors

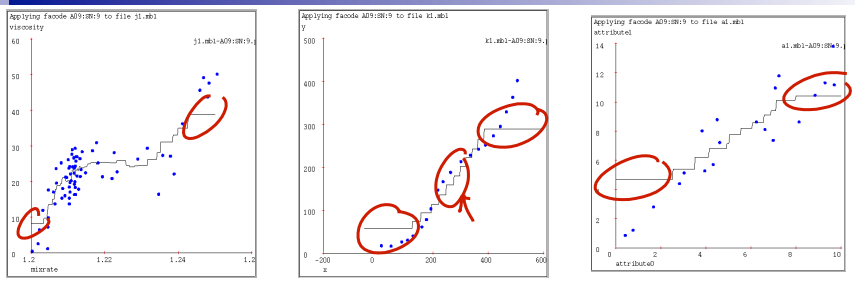
in regression

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

in classification:

majority class in $NN(x_q)$

k-Nearest Neighbor (here k=9)



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

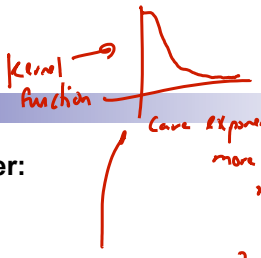
- Neighbors are not all the same

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

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)*
   e.g.) $\pi^i = exp(-D(x^i, query)^2 / \rho^2)$ ↙ Squared exponential function

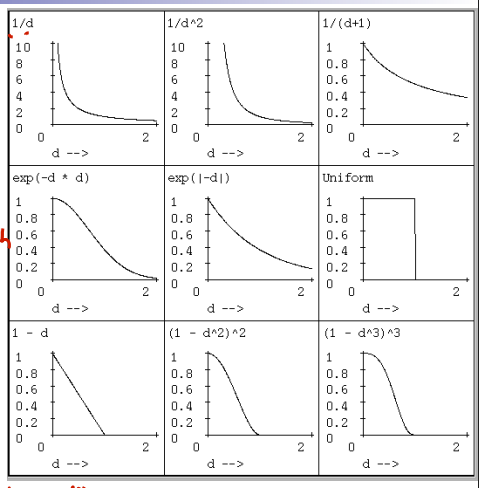
   $\pi_i = e^{\frac{-\|x^q - x_i\|^2}{\rho^2}}$

   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:
   predict = $\Sigma \pi^i y^i / \Sigma \pi^i$
   ↗ regression

classification: weighted majority

Weighting functions

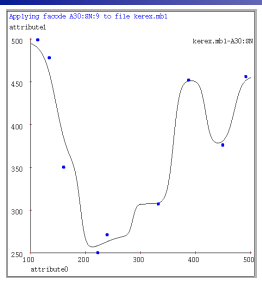
$\pi^i = exp(-D(x^i, query)^2 / \rho^2)$



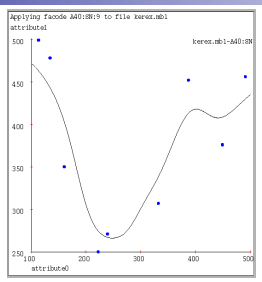
Typically optimize ρ using gradient descent

(Our examples use Gaussian)

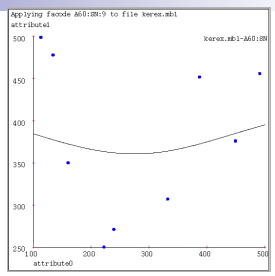
Kernel regression predictions



ρ=10



ρ=20

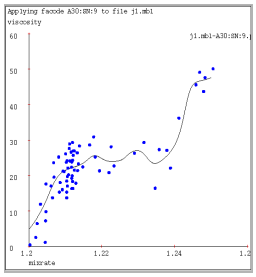


ρ=80

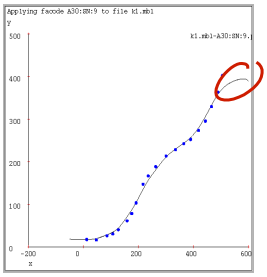
Increasing the kernel width ρ means further away points get an opportunity to influence you.

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

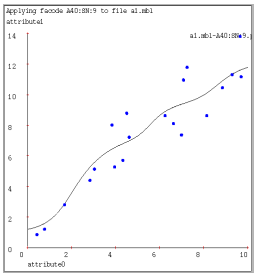
Kernel regression on our test cases



ρ=1/32 of x-axis width.



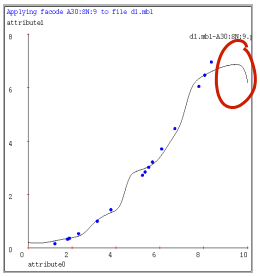
ρ=1/32 of x-axis width.



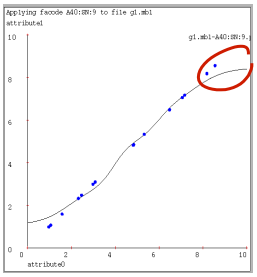
ρ=1/16 axis width.

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

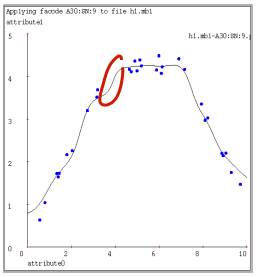
Kernel regression can look bad



ρ = Best.



ρ = Best.



ρ = 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:

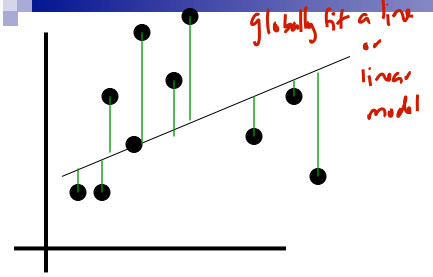
$$\hat{w}^q = \underset{w}{\operatorname{argmin}} \sum_{k=1}^{N} \pi_q^k \left(y^k - w^T x^k\right)^2$$

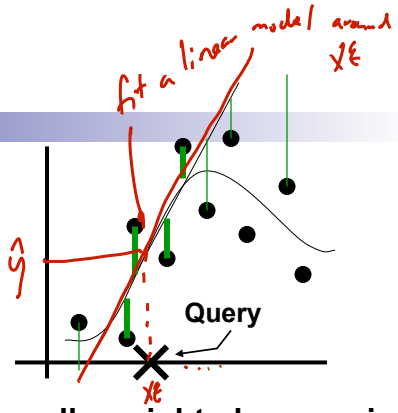
$\hat{y}_t = \hat{w}_t \cdot x_t$

fit a regression on weighted data

weights

How LWR works





Linear regression

- Same parameters for all queries

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

matrix inverse

Locally weighted regression

- Solve weighted linear regression for each query

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

using matrix inversion?

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

Another view of LWR

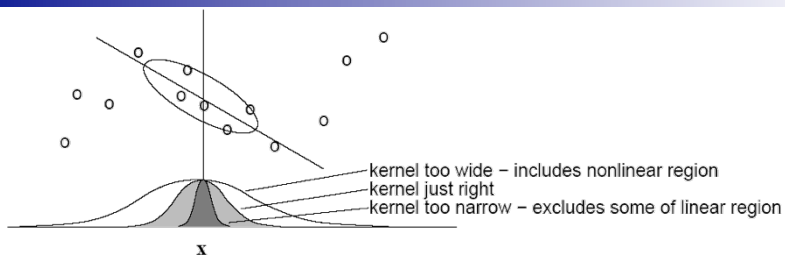
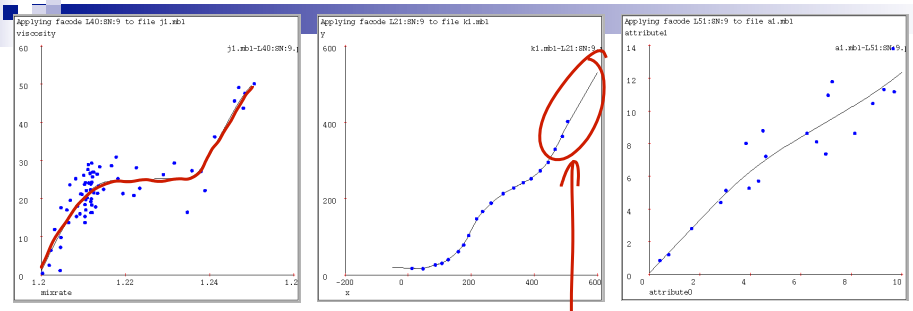


Image from Cohn, D.A., Ghahramani, Z., and Jordan, M.I. (1996) "Active Learning with Statistical Models", JAIR Volume 4, pages 129-145.

LWR on our test cases



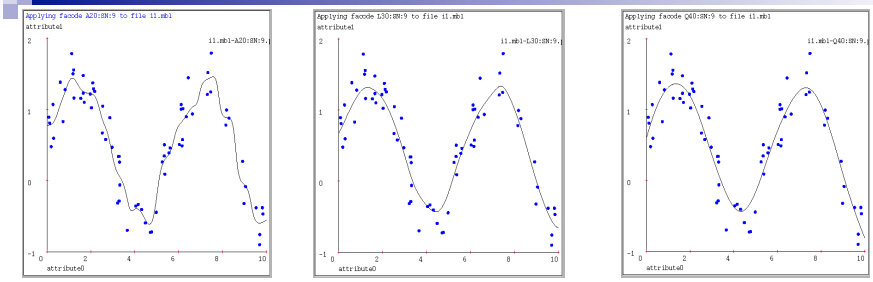
ρ = 1/16 of x-axis width.

ρ = 1/32 of x-axis width.

ρ = 1/8 of x-axis width.

more localized fits

Locally weighted polynomial regression



Kernel Regression
Kernel width ρ at optimal level.

ρ = 1/100 x-axis

LW Linear Regression
Kernel width ρ at optimal level.

ρ = 1/40 x-axis

LW Quadratic Regression
Kernel width ρ at optimal level.

ρ = 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 retreve 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

Acknowledgment

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