Overfitting in General

Consider error of hypothesis h over

- training data: $error_{train}(h)$
- entire distribution $\mathcal{D}$ of data: $error_{\mathcal{D}}(h)$

Definition
Hypothesis $h \in H$ overfits training data if there is an alternative hypothesis $h' \in H$ such that

$$error_{train}(h) < error_{train}(h')$$

And

$$error_{\mathcal{D}}(h) > error_{\mathcal{D}}(h')$$

Avoiding Overfitting

How can we avoid overfitting? **Pruning**

- **pre-pruning:** stop growing when data split not statistically significant
- **post-pruning:** grow full tree, then remove sub-trees which are overfitting (based on validation set)

Post-pruning avoids problem of “early stopping”

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, in ID3: chi-squared test plus information gain
  - only statistically significant attributes were allowed to be selected by information gain procedure

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 *sklearn*, this parameter is $\min\_samples\_leaf$
- Stop when Entropy changes is smaller than a lower-round
  - In *sklearn*, the parameter $\min\_impurity\_decrease$ enables stopping when this falls below a lower-bound

Avoiding Overfitting

- Other pre-pruning approaches:
    - Maximum number of leaf nodes
        - in *sklearn*, $\max\_leaf\_nodes$
    - Minimum number of samples to split
        - in *sklearn* $\min\_samples\_split$
    - Maximum depth
        - in *sklearn*, $\max\_depth$

Avoiding Overfitting

Early stopping

- Pre-pruning may suffer from 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

Avoiding Overfitting

Post-pruning

- Builds full tree first and prunes it afterwards
    - Attribute interactions are visible in fully-grown tree
- Problem: identification of subtrees and nodes that are due to chance effects
- Subtree replacement
- Possible strategies: error estimation, significance testing, MDL principle. We examine
    - reduced-error Pruning
    - Minimum error
    - Smallest tree

Reduced-Error Pruning

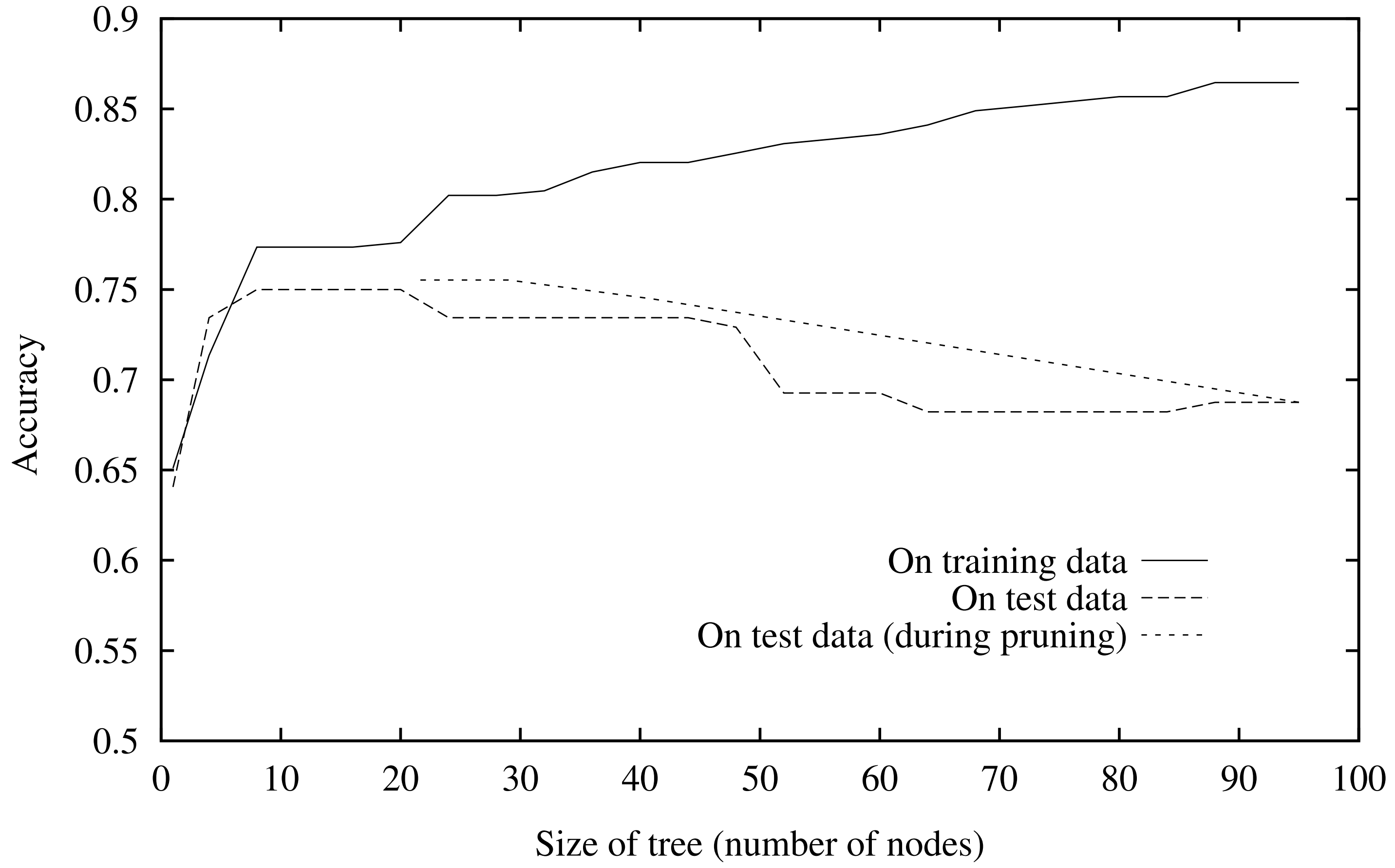
Split data into *training* and *validation* set

Do until further pruning is harmful:

- ○ Evaluate the impact of pruning nodes in the bottom of the tree and replacing them with a leaf node on *validation* set
- ○ Greedily remove the one that most improves *validation* set accuracy

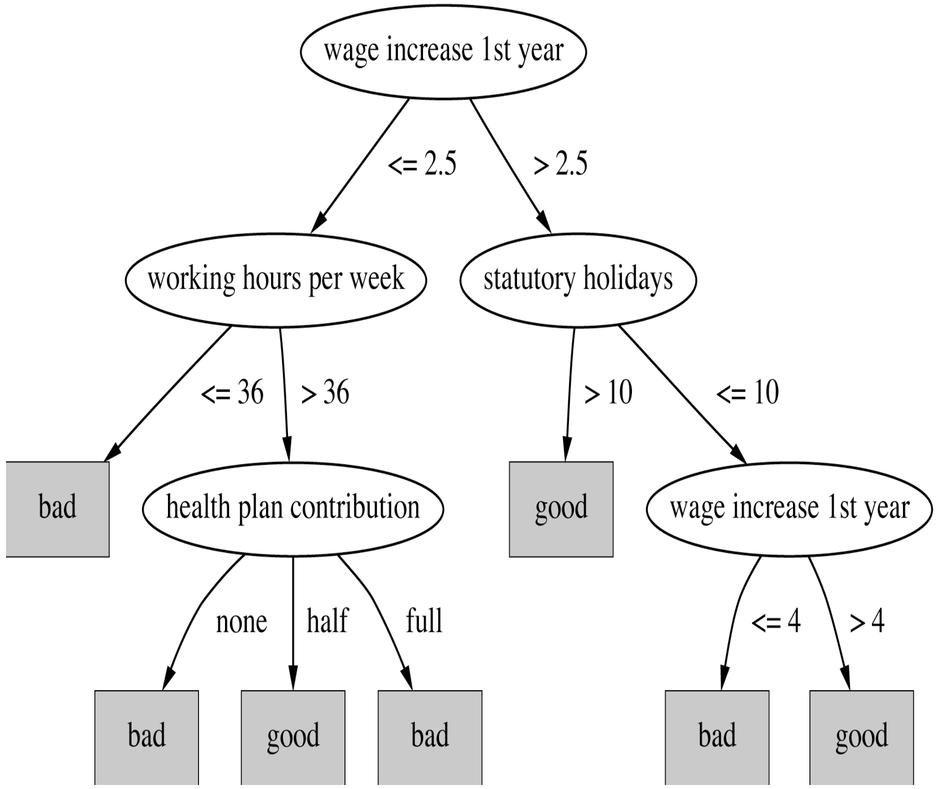
- **Good** it is simple and produces smallest version of most accurate subtree
- **Not so good** reduces effective size of training set

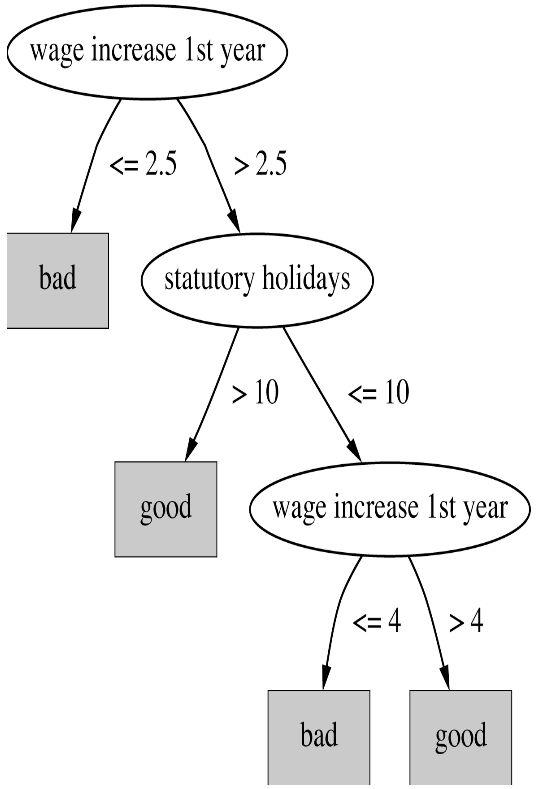
Effect of Reduced-Error Pruning



Pruning operator: Sub-tree replacement

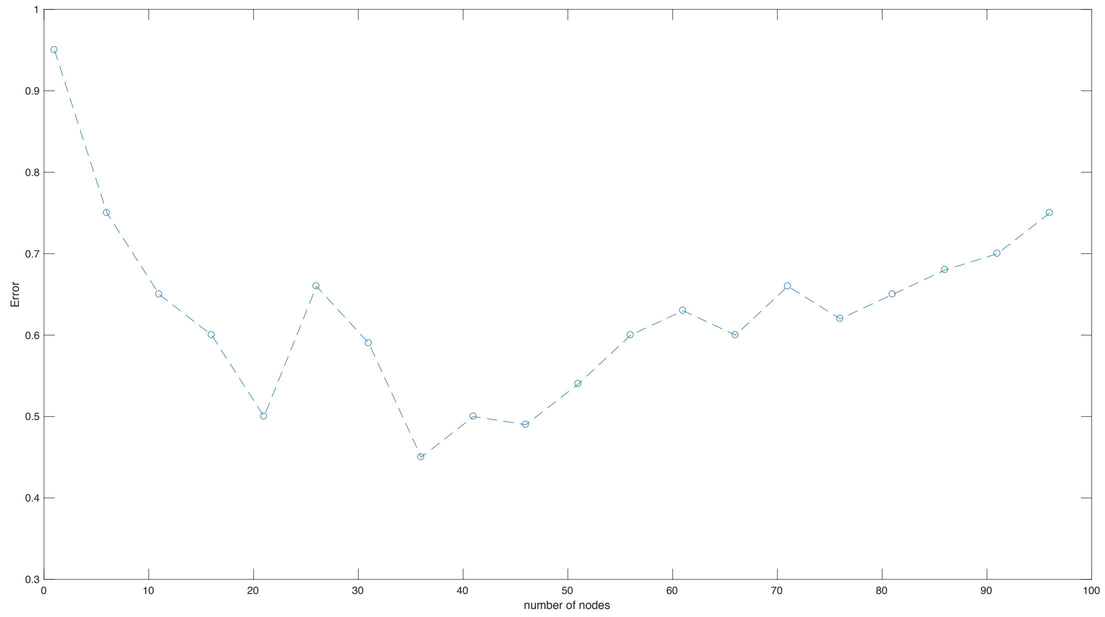
Bottom-up:
Tree is considered for replacement once all its sub-trees have been considered





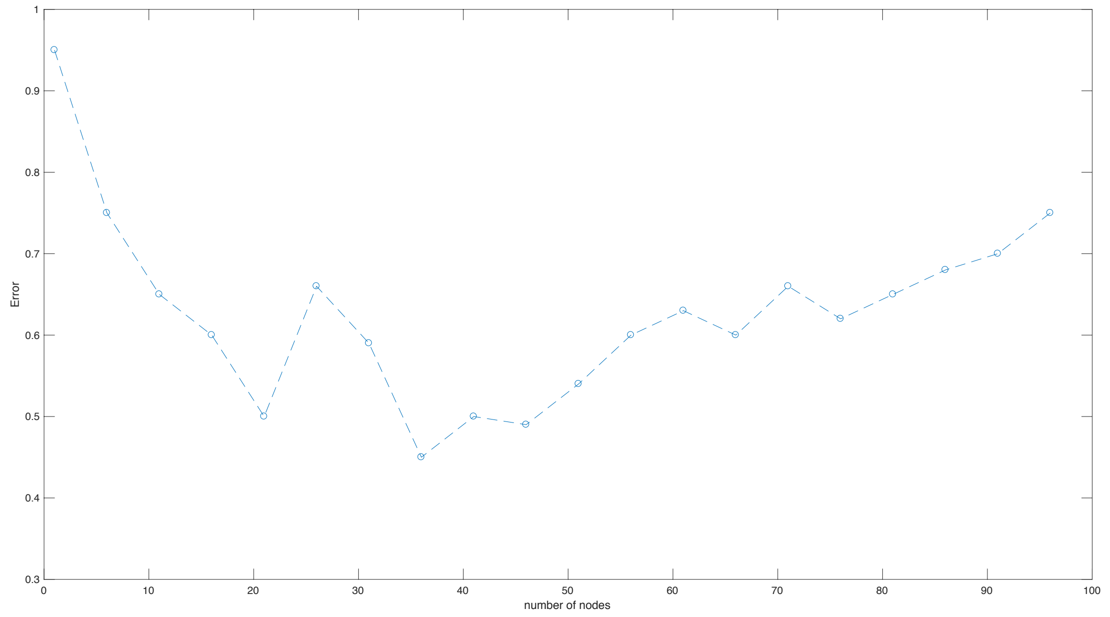
Minimum Error

- Keep the Cross-Validation error during the growing
- Prune back to the point where the cross-validated error is a minimum.
  - In the following example, the first 36 nodes will be kept

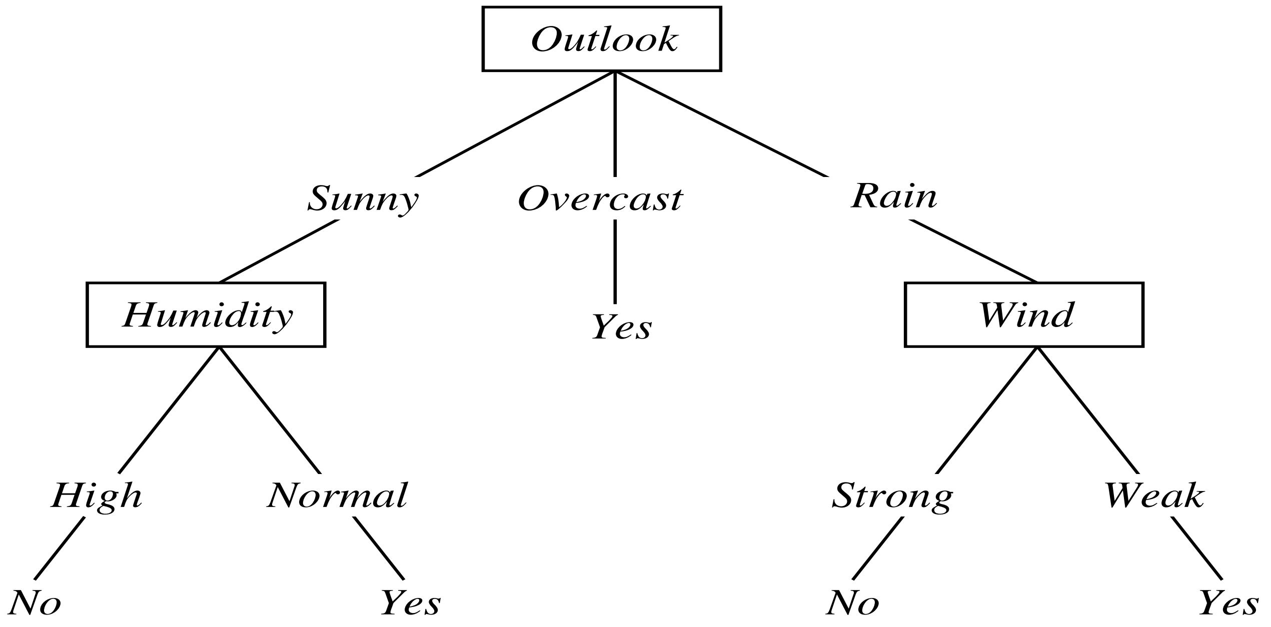


Smallest Tree

- Keep the Cross-Validation error during the growing
- Prune back the tree to the smallest tree within 1 standard error of the minimum error.
  - In the following example, the first 20 nodes will be kept



Converting A Tree to Rules



If $(Outlook = Sunny) \wedge (Humidity = High)$
Then $PlayTennis = No$

If $(Outlook = Sunny) \wedge (Humidity = Normal)$
Then $PlayTennis = Yes$
…

Rules from Trees

Rules can be simpler than trees but just as accurate, e.g.:

- 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 (by pruning)
  - i.e., rules can be generalized while maintaining accuracy

Continuous Valued Attributes

Decision trees originated for discrete attributes only. Now: continuous attributes.

Can create a discrete attribute to test continuous value:

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

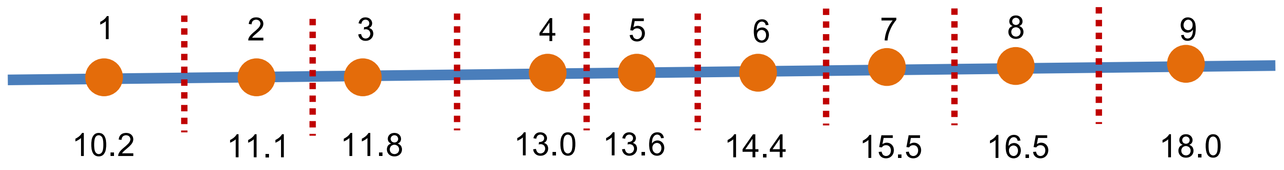
Continuous Valued Attributes

- 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 continuous 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: C4.5 uses actual values in data

Temperature:	40	48	60	72	80	90
PlayTennis:	No	No	Yes	Yes	Yes	No

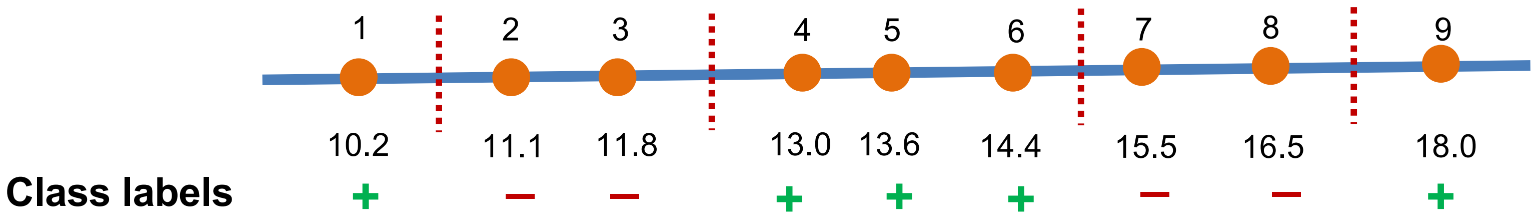
Continuous Valued Attributes

- Example for $n - 1$ splits



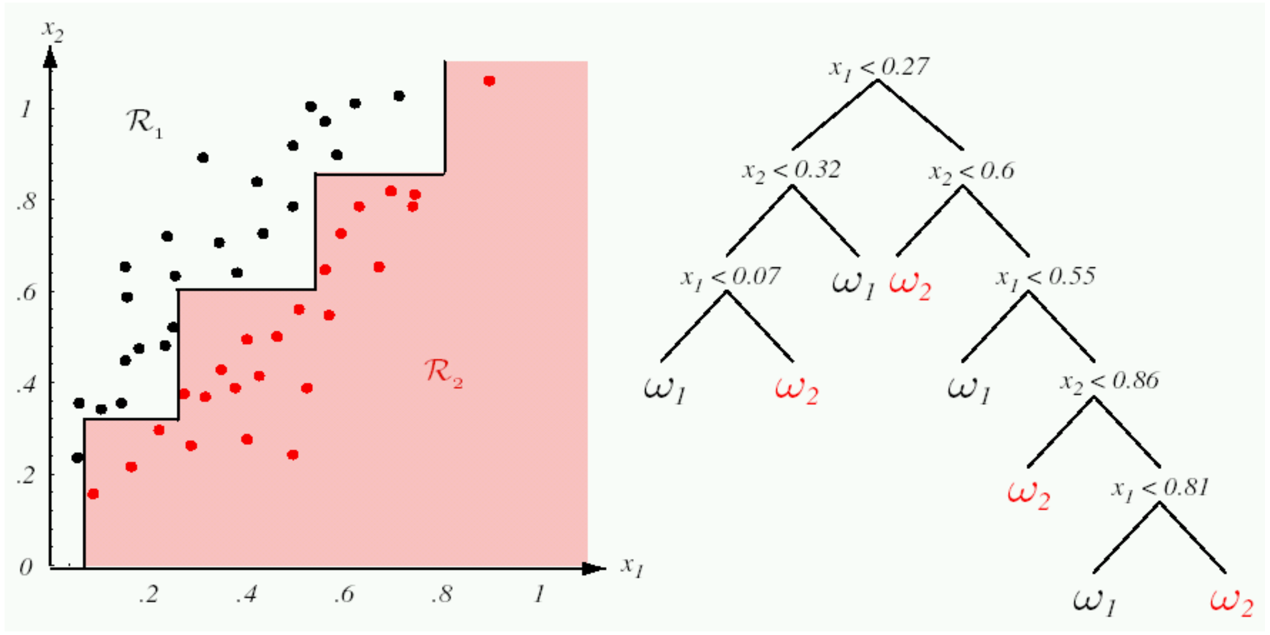
Continuous Valued Attributes

- Example for $n-1$ splits: more efficient number of splits if we know the class labels



- You can use the midway point as your boundary (e.g if $x > 12.4$ between points 3 and 4)
- Or, you can use the value in the training (e.g if $x > 11.8$ between points 3 and 4)

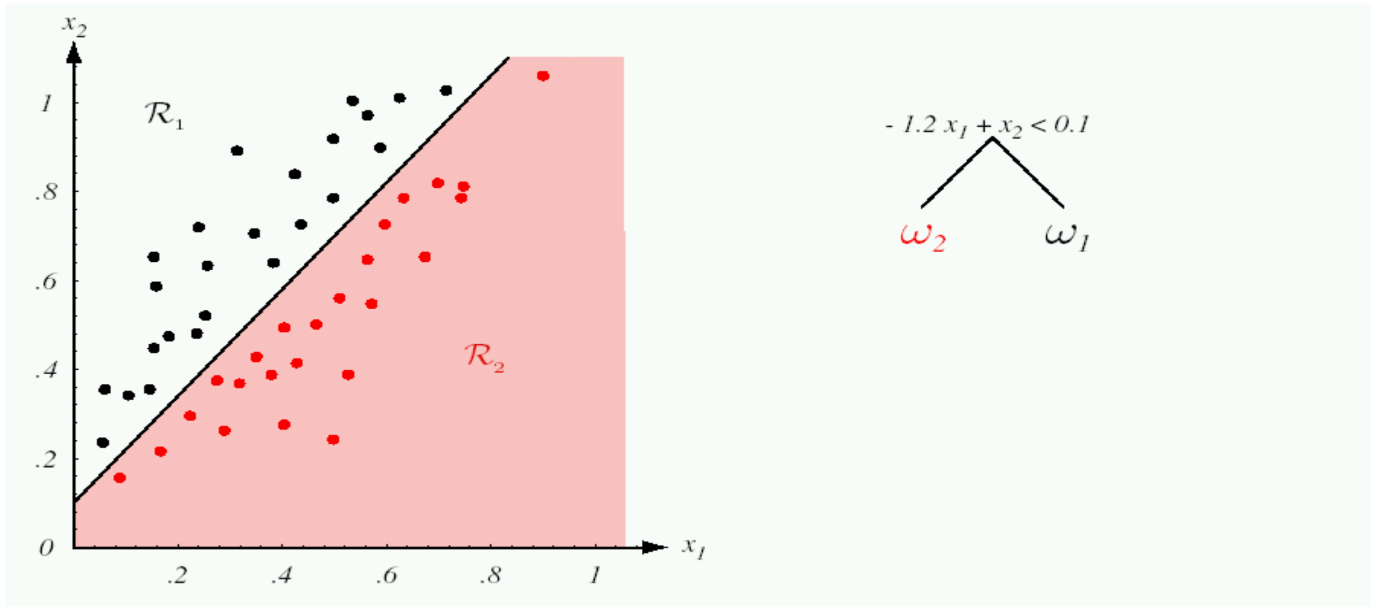
Axis-parallel Splitting



Fitting data that is not a good "match" to the possible splits in a tree.

"Pattern Classification" Duda, Hart, and Stork, (2001)

Splitting on Linear Combinations of Features



Reduced tree size by allowing splits that are a better “match” to the data.

“Pattern Classification” Duda, Hart, and Stork, (2001)

Attributes with Costs

Consider

- medical diagnosis, $Blood\ Test$ has cost \$150
- robotics, computation cost 23 sec.

How to learn a consistent tree with low expected cost?

Attributes with Costs

One approach: evaluate information gain *relative* to cost:

- Example

$$\frac{Gain^2(S, A)}{Cost(A)}$$

Preference for decision trees using lower-cost attributes.

Misclassification with Costs

Also: class (misclassification) costs, instance costs, . . .

Can give *false positives* a different cost to *false negatives.*

Similar to expected loss in"Classification 2" lecture, we can define a loss function $\lambda(\alpha_i|predicted\ class)$ (the loss associated to action α_i) and define the expected loss as:

$$R(\alpha_i|x) = \sum_{h \in H} \lambda(\alpha_i|predicted\ class)\ P(predicted\ class|x)$$

- however, if we use this loss function to split the tree, it has been shown that minimizing R does not necessarily lead to the best long-term growth of the tree.

Misclassification with Costs

So what is usually done instead :

- Grow the tree using gain information to its full depth (or other impurity measures)
- Apply cost minimization when post pruning (or minimize cost-complexity measure which takes the size of the tree into account as well)

Unknown Attribute Values

What if some examples missing values of A?

Use training 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 same fashion

Windowing

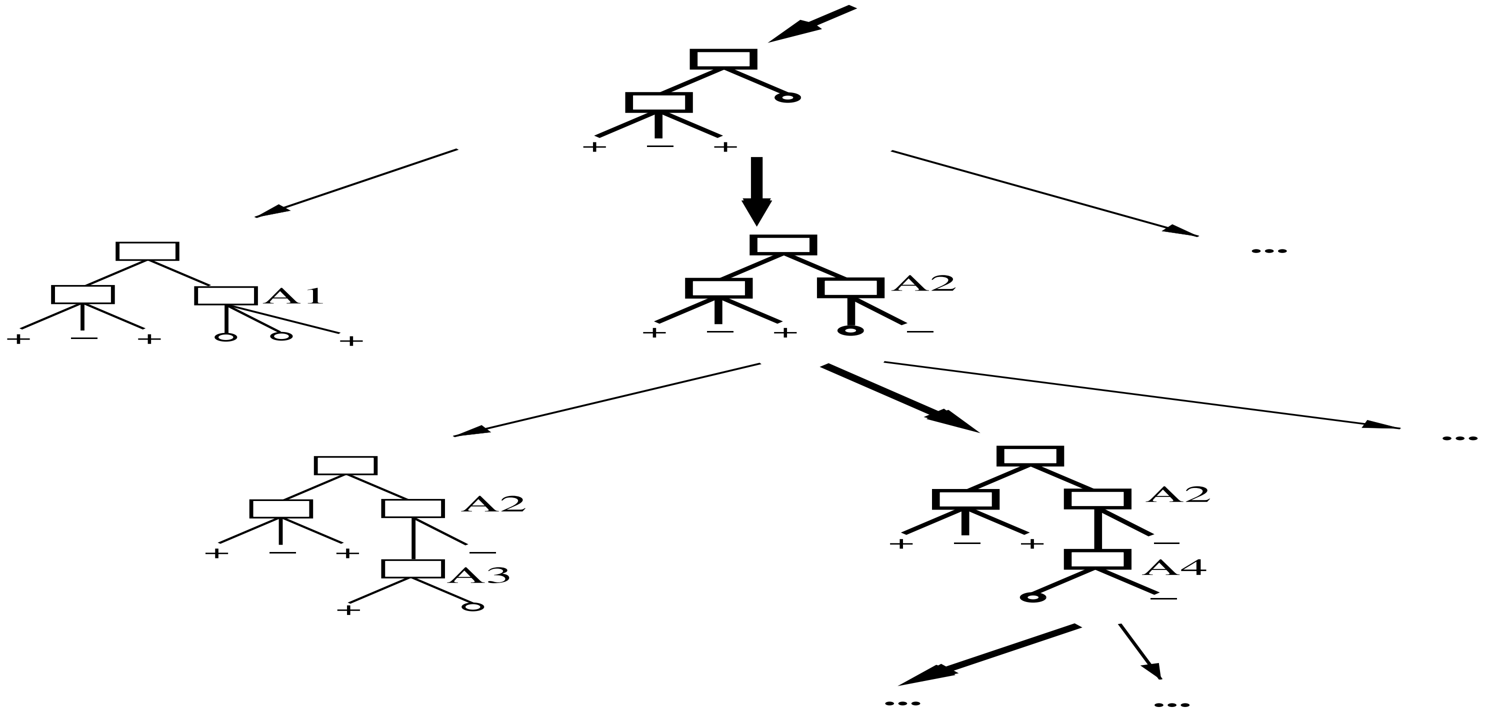
Early implementations – training sets was too large for memory. How to handle the problem?

As a solution ID3 implemented *windowing*:

1. select subset of instances – the *window*
2. construct decision tree from all instances in the window
3. use tree to classify training instances *not* in window
4. if all instances correctly classified then halt, else
5. add selected misclassified instances to the window
6. gotostep2

Windowing retained in C4.5 because it can lead to more accurate trees. Related to *ensemble learning*.

Hypothesis Space Search by ID3



Hypothesis Space Search by ID3

- Usual graph-search technique: greedy or beam search, starting with the node corresponding to the “empty tree” (single leaf node)
- Greedy choice: which one to select? The one that results in the greatest increase in posterior probability
- RESULT: set of trees with (reasonably) high posterior probabilities given D: we can now use these to answer questions like $P\ (y' = C_1 | \ldots)$? or even make a *decision* or a *classification* that $y' = C_1$, given input data x

Hypothesis Space Search by ID3

- ID3 searches the space of possible decision trees: doing hill-climbing on information gain.
- Hypothesis space is complete! (contains all finite discrete-valued functions w.r.t. attributes)
- Outputs a single hypothesis.
    - It cannot tell us how many other viable ones there are.
- No back tracking
    - Local minima...
- Uses all training examples at each step.
    - Results are less sensitive to errors
- Inductive bias

Inductive Bias in ID3

Inductive bias: set of assumptions needed in addition to training data to justify deductively learner's classifications

Restriction bias:

- The set of hypothesis that can be modelled by decision trees

Preference biases:

- Prefers trees with good splits near the top (splitting on features with the most information gain)
- Prefers shorter trees (comes naturally from good splits at the top and minimum description length)

Decision Tree

Pros:

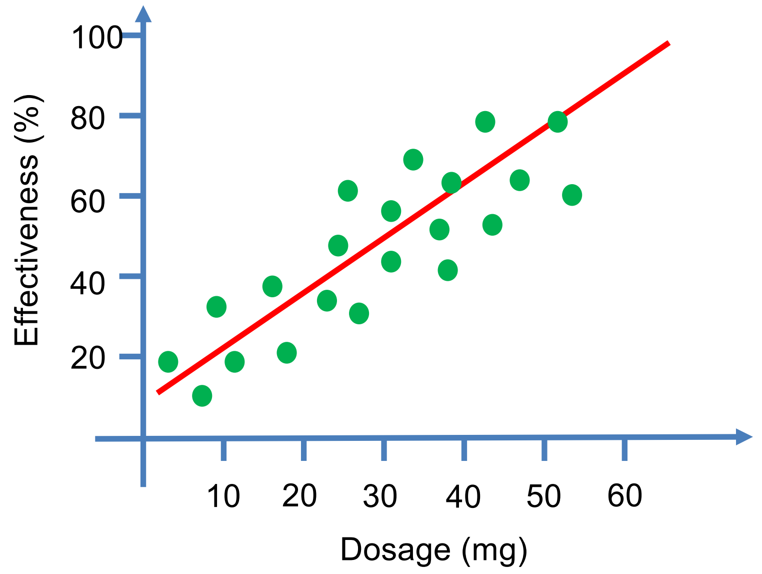
- Interpretability
- Easily handle irrelevant attributes (Gain = 0)
- Can handle both categorical and numerical data
- Can handle missing data
- Very compact (number of nodes << number of examples)
- Very fast at testing

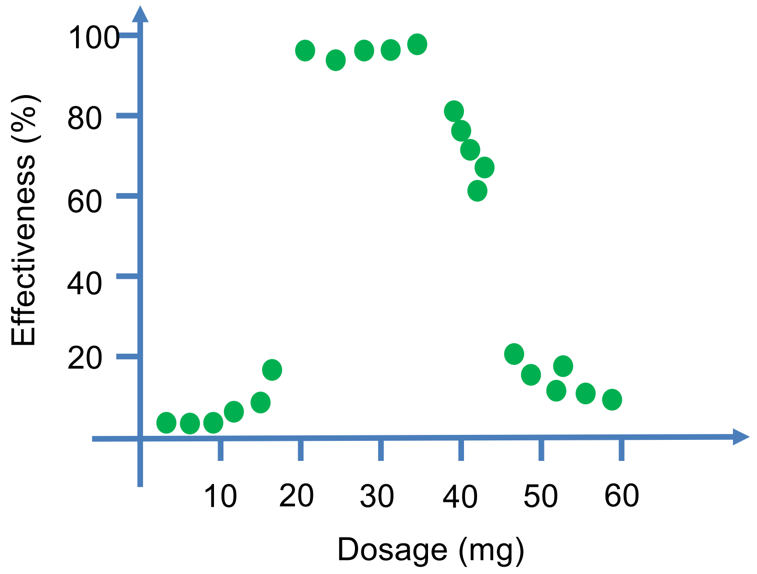
Cons:

- Only axis-aligned splits of the data
- Tend to overfit
- Greedy (may not find the best tree)
    - Exponentially many possible trees

Regression Tree

Example: A clinical trial has been done to evaluate the effect of a drug dosage.

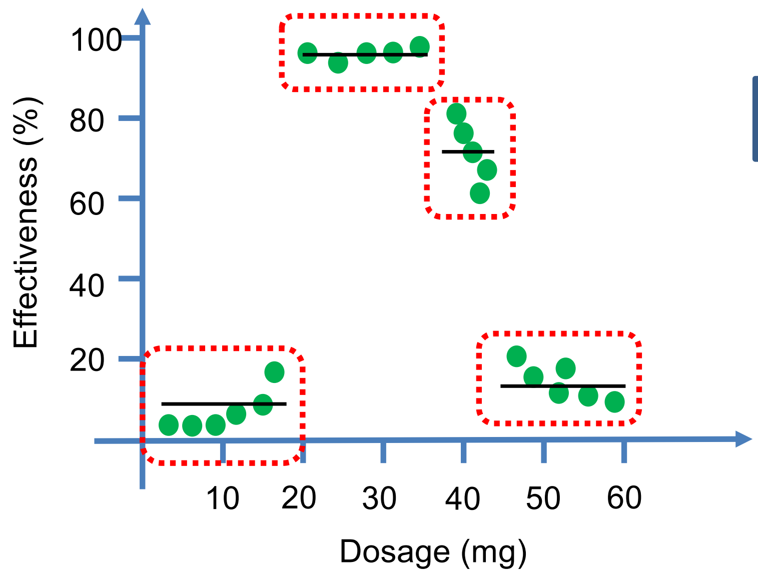


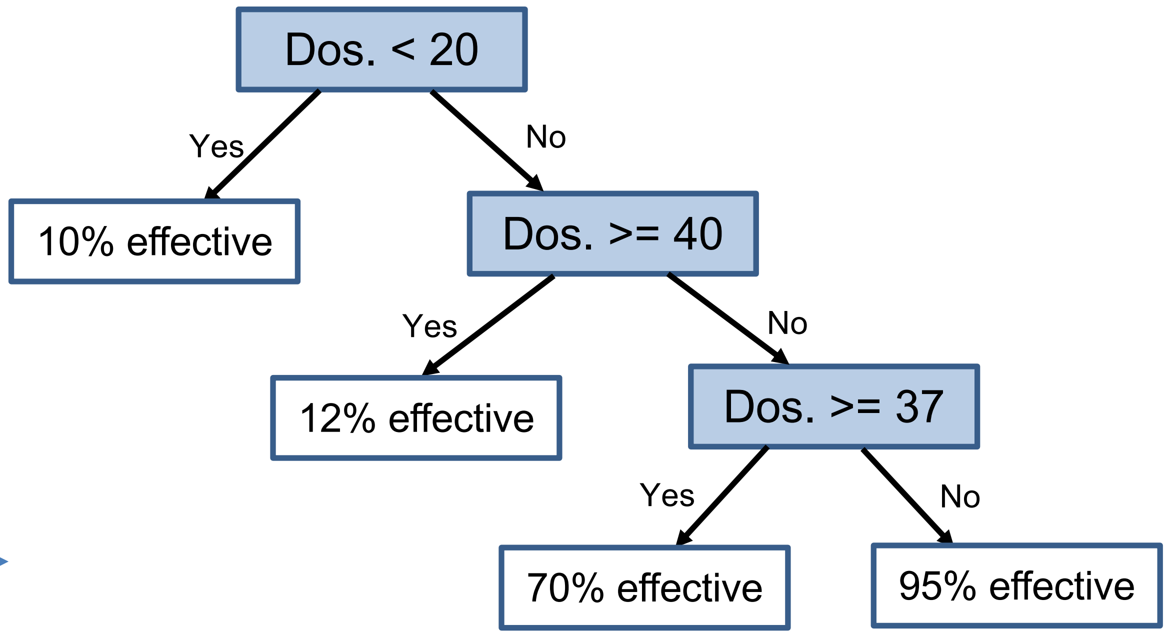


How to model the data on the right?

Regression Tree (CaRT algorithm)

One option would be to use regression trees





Each leaf corresponds to average drug effectiveness in a different cluster of examples, the tree does a better job than *Linear Regression*

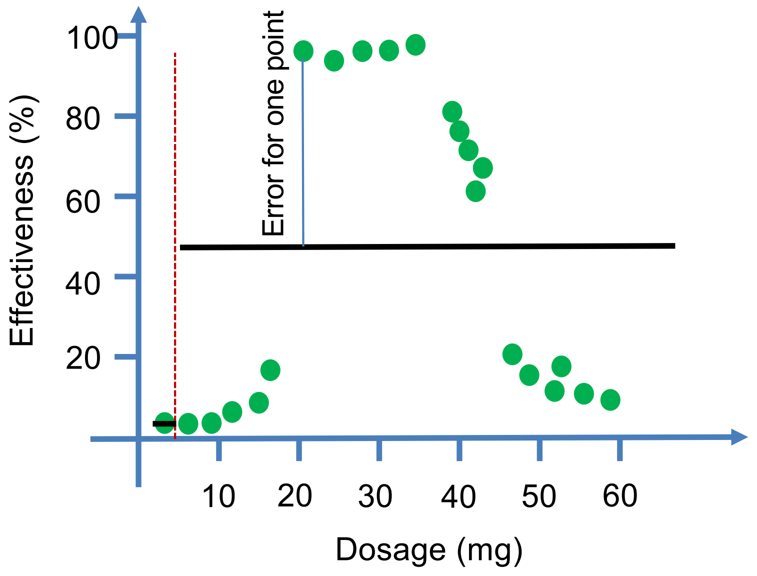
Regression Tree

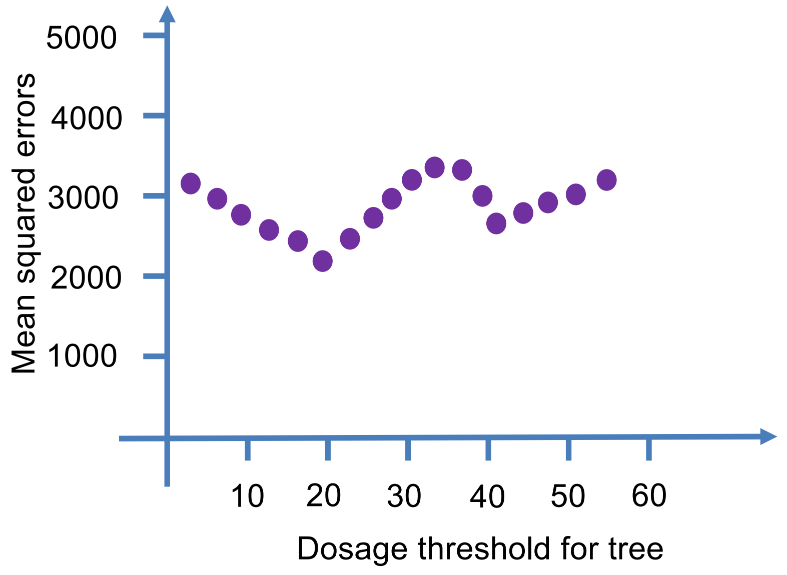
Although in this example, we can use other methods like *kNN* or *local regression*, but the advantage of *Regression Tree* will be more clear if you have multiple variables, specially if some of them are categorical and some are real-valued.

How to build the regression tree and find the thresholds?

Regression Tree

We will use mean squared errors to find the best threshold. We start from setting the threshold to the value of the first point to the last point, successively, and examine the mean of squared errors.





Regression Tree

- Mean squared error after setting the threshold, for each subset with m examples is:

$$MSE(Y_i) = \frac{1}{m}\sum_{j=1}^{m}(y_j - \bar{y})^2$$

  - The MSE here is equal to the variance of the examples in that subset
- Compute the weighted average of variance

$$weighted\ average\ variance = \sum_{i}^{l}\frac{|Y_i|}{|Y|} MSE(Y_i)$$

- We pick a threshold which minimizes the weighted-average/average of MSE/variance

Regression Tree

Multiple variables:

Imagine that in the drug experiment we also have other attributes like “age”, “sex”,…

To choose an attribute for each node:

- For each attribute separately find the best split using the minimum weighted average variance
- Pick the attribute with minimum weighted average variance

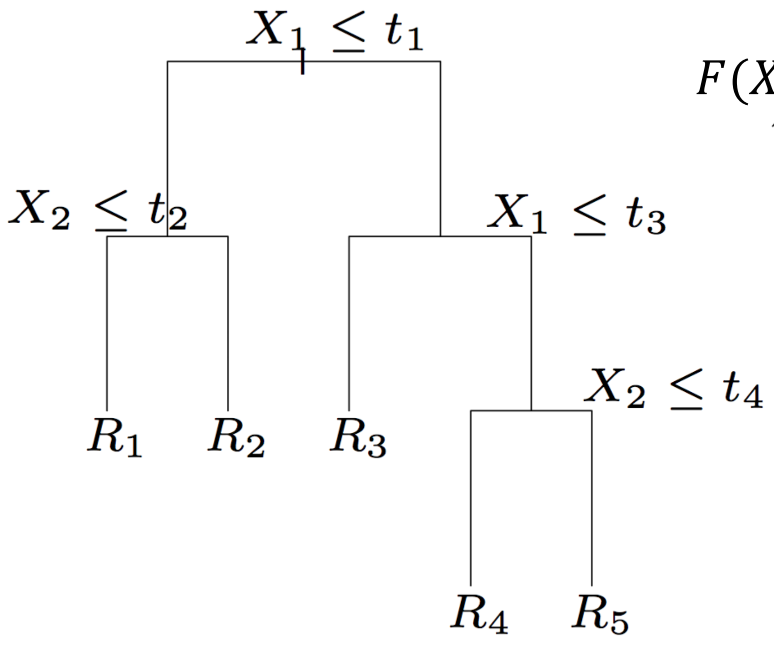
Regression Tree

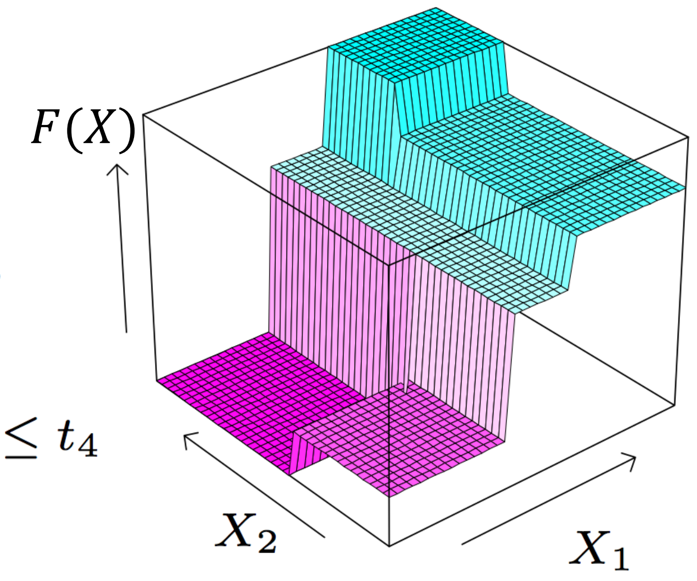
Similar to decision tree, if we continue splitting all nodes that have have a *mean squared error* bigger than zero, we will get a tree which fits the training data perfectly, but will not generalize well to the test data.

How to avoid overfitting:

- The simplest to avoid overfitting, similar to, *decision tree* is to only split examples, when there are more than some minimum number (usually a default value for minimum number of examples to split is 20).
- Or using maximum depth. Allow the tree to grow to the specified depth
- Or maximum number of leaves
- …

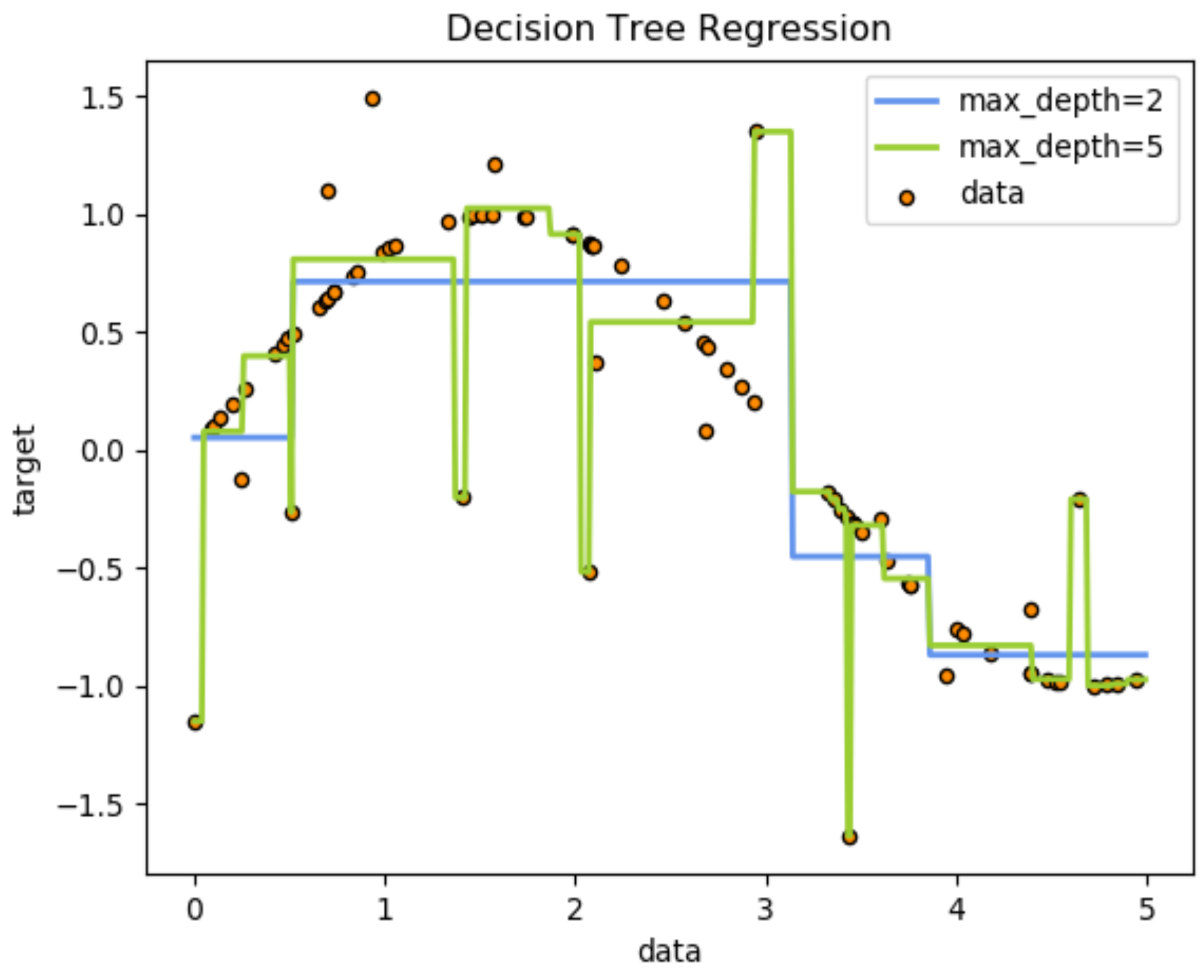
A Regression Tree and its Prediction Surface



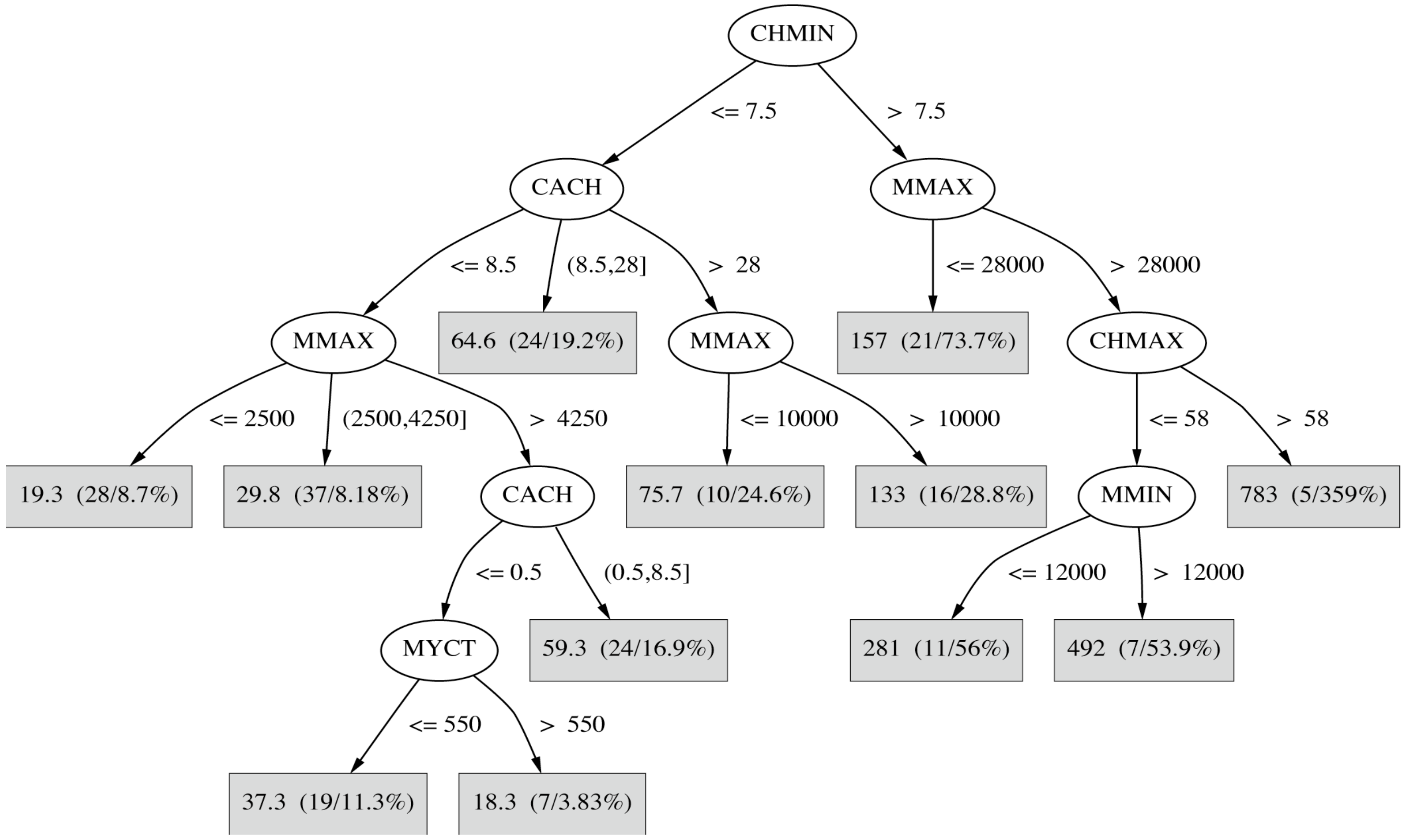


"Elements of Statistical Learning" Hastie, Tibshirani & Friedman (2001)

Regression Tree on sine dataset



Regression Tree on CPU dataset



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:

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

$$Condition = [excellent, good, fair]$$
$$[1770, 4513][270, 870, 1051, 1900][77, 99, 625]$$

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

Learning a regression tree

- The means of the first split are 1574, 4513, 77, 870 and 331, and the weighted average of mean squared errors is 3.21×10^5.
- The means of the second split are 3142, 1023 and 267, with weighted average of mean squared errors 2.68×10^6;
- for the third split the means are 1132 and 1297, with weighted average of mean squared errors 1.55×10^6.

We therefore branch on $Model$ at the top level. This gives us three single-instance leaves, as well as three $A100$s and three $T202$s.

Learning a regression tree

For the A100s we obtain the following splits:

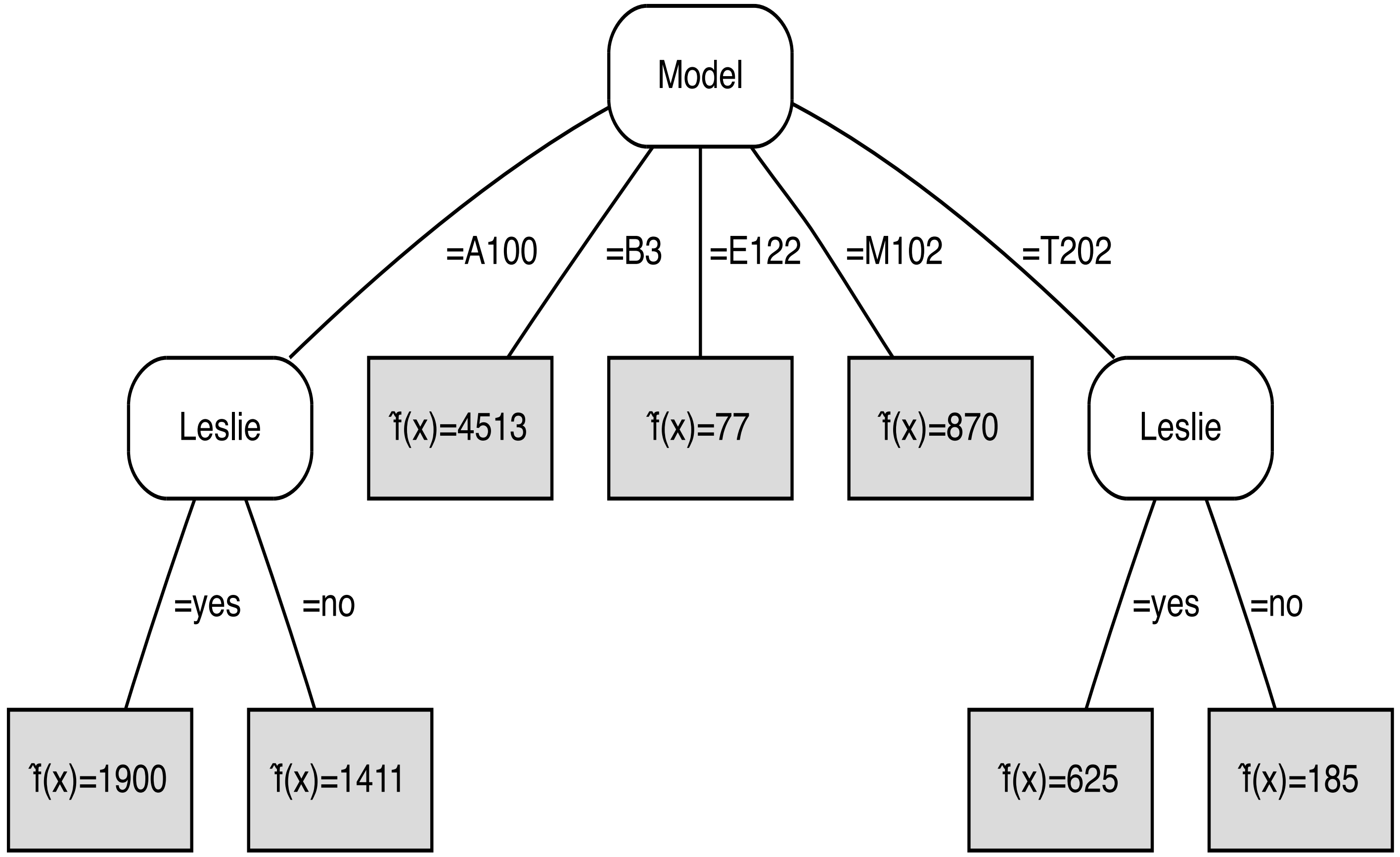
$$Condition = [excellent, good, fair]$$
$$[1770][1051{,}1900][]$$
$$Leslie = [yes, no]\ [1900]$$
$$[1051{,}1770]$$

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 $T202$s 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.

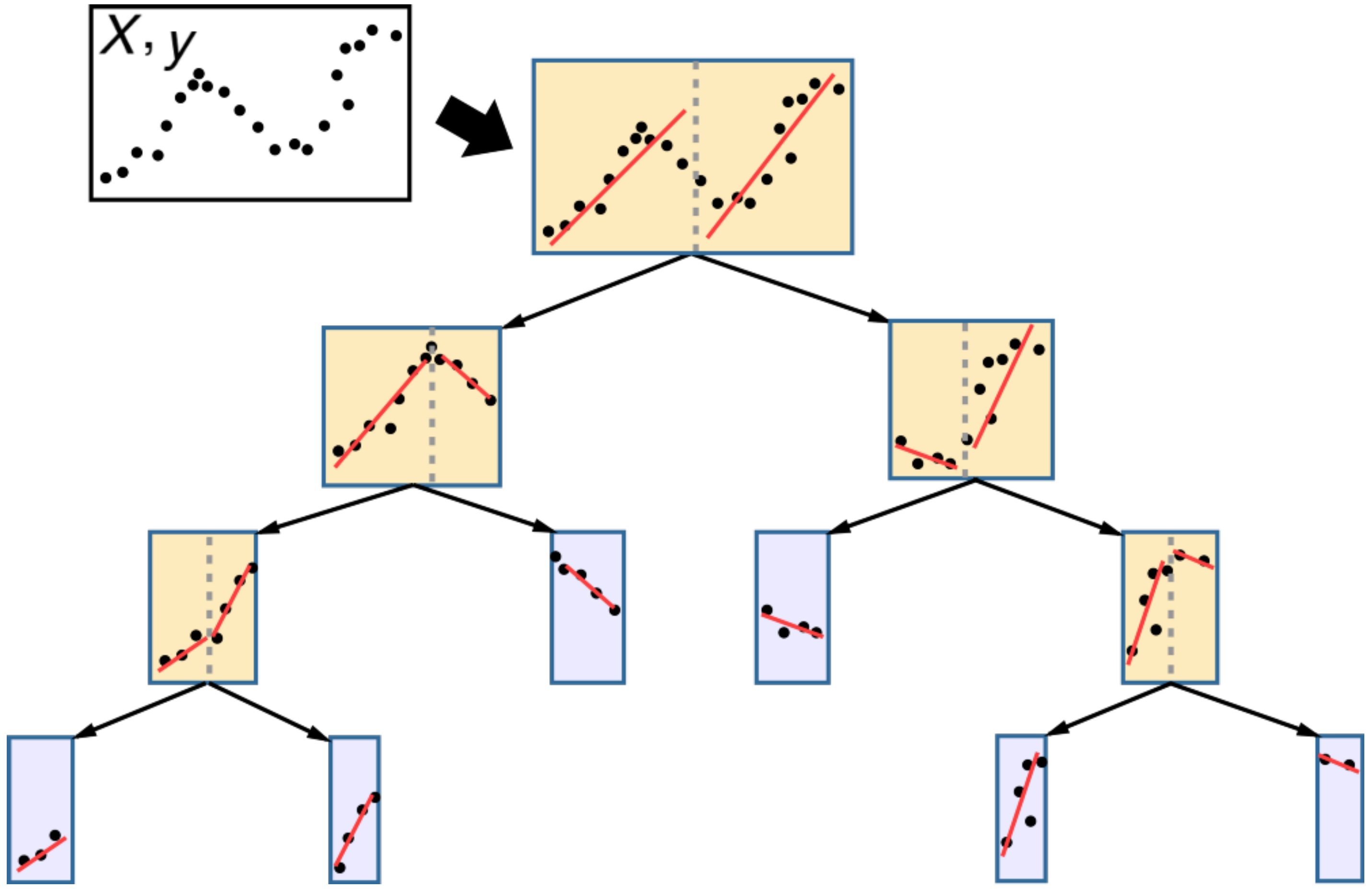
Regression trees

- 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

Model trees

- Like regression trees but with linear regression functions (or any model of your choice) 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 Linear Regression
  - Attributes occurring in subtree (+maybe attributes occurring in path to the root)
- Fast: overhead for Linear Regression not large because usually only a small subset of attributes is used in tree

Model trees

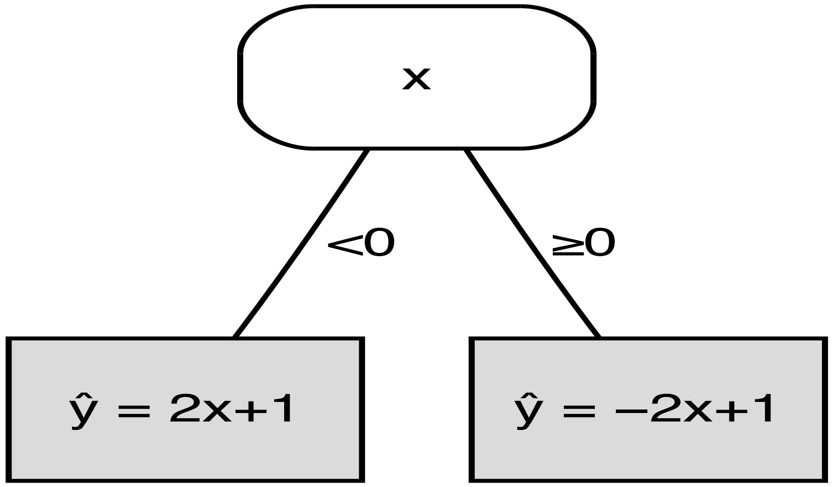


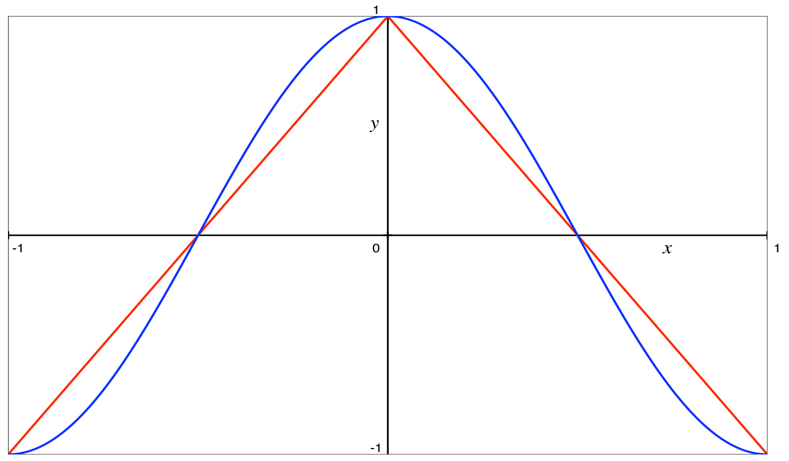
Model Tree Example

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).

Model Tree Example





Building the tree

- Splitting criterion: *standard deviation reduction*

$$SDR = sd(T) - \sum_i \frac{|T_i|}{|T|} \times sd(T_i)$$

where $T_1,\ T_2, \ldots$ are the sets from splits of data at node.

- Termination criteria (important when building trees for numeric prediction):
  - Standard deviation becomes smaller than certain fraction of sd for full training set (e.g. 5%)
  - Too few instances remain (e.g. less than four)

Pruning the tree

Model trees suffer the same deficits as decision tree, which is the tendency to overfit the train data when using complex models

- Pruning is based on estimated absolute error of LR models
- Linear Regression (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

Discrete (nominal) attributes

- Nominal attributes converted to binary attributes and treated as numeric
    - Nominal values sorted using average class value for each one
    - For k-values, $k - 1$ binary attributes are generated
        - the ith binary attribute is 0 if an instance's value is one of the first i in the ordering, 1 otherwise
- Best binary split with original attribute provably equivalent to a split on one of the new attributes

This is basically "one-hot-encoding"

Summary – decision trees

- Decision tree learning is a practical method for many classifier learning tasks – still a “Top 10” data mining algorithm
- TDIDT family descended from ID3 searches complete hypothesis space - the hypothesis is there, somewhere...
- 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

Summary – regression and model trees

- Often the “try-first” machine learning method in applications, illustrates many general issues
- Performance can be improved with use of “ensemble” methods
- Regression trees were introduced in CART – R’s implementation is close to CART, but see sklearn.tree.DecisionTreeRegressor for a basic version

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
- Material derived from slides by Josh Starmer (StatQuest Channel)