# Tree Learning

Based on the ML lecture by Raymond J. Mooney
and Peter Flach book

# Machine learning settings

| | *Predictive model* | *Descriptive model* |
|---|---|---|
| *Supervised learning* | classification, regression | subgroup discovery |
| *Unsupervised learning* | predictive clustering | descriptive clustering, association rule discovery |

**Table 1.1.** An overview of different machine learning settings. The rows refer to whether the training data is labelled with a target variable, while the columns indicate whether the models learned are used to predict a target variable or rather describe the given data.

(from Peter Flach book)

# Machine learning settings

- Logical approach - Decision and regression trees, rules
- Probabilistic methods – Bayesian methods
- Linear methods – Linear discriminant, SVM, perceptron
- Distance-based methods – Lazy Learning (kNN), clustering

# Learning Trees

- Supervised method – data classified into classes, i.e. data contains a target attribute.
- Decision trees. Finite number of classes >= 2
- Regression trees. Class is continuous

# Learning Trees 2

- Classifier is a tree that represents a hypotheses in a disjunctive normal form
- Finding a minimal decision tree (nodes, leaves, or depth) is an NP-hard optimization problem.
- Heuristic algorithm can be used to build a tree
- Want to pick a feature that creates subsets of examples that are – in the case of decision trees - relatively “pure” in a single class so they are “closer” to being leaf nodes.

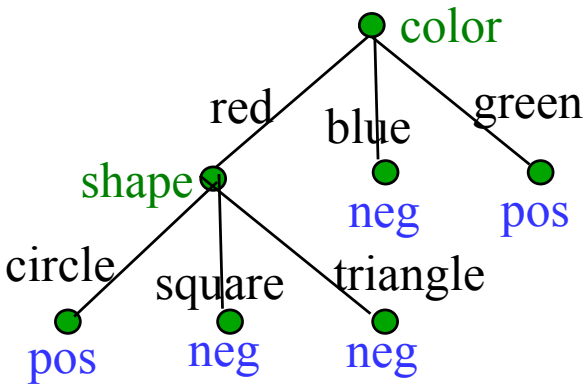
# Learning Decision Trees

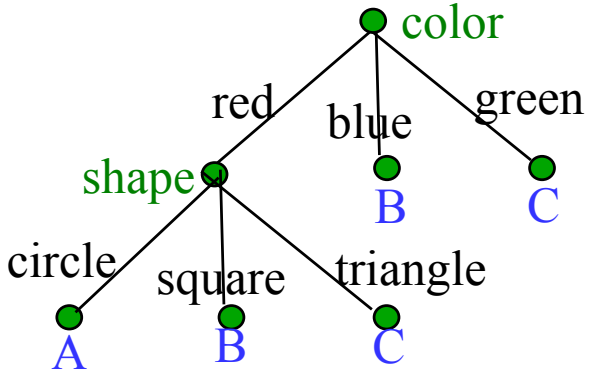
# Data example

| Example | Size | Color | Shape | Category |
|---|---|---|---|---|
| 1 | small | red | circle | positive |
| 2 | large | red | circle | positive |
| 3 | small | red | triangle | negative |
| 4 | large | blue | circle | negative |

# Decision Trees

- Tree-based classifiers for instances represented as feature-vectors. Nodes test features, there is one branch for each value of the feature, and leaves specify the category.




- Can represent arbitrary conjunction and disjunction. Can represent any classification function over discrete feature vectors.
- Can be rewritten as a set of rules, i.e. disjunctive normal form (DNF).
  - red ∧ circle → pos
  - red ∧ circle → A

    blue → B; red ∧ square → B

    green → C; red ∧ triangle → C

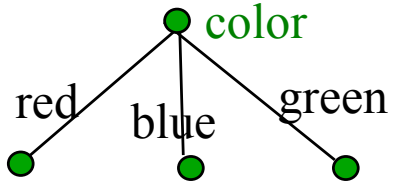
# Properties of Decision Tree Learning

- Continuous (real-valued) features can be handled by allowing nodes to split a real valued feature into two ranges based on a threshold (e.g. length < 3 and length ≥3)
- Classification trees have discrete class labels at the leaves, ***regression trees*** allow real-valued outputs at the leaves.
- Algorithms for finding consistent trees are efficient for processing large amounts of training data for data mining tasks.
- Methods developed for handling noisy training data (both class and feature noise).
- Methods developed for handling missing feature values.

# Top-Down Decision Tree Induction

- Recursively build a tree top-down by divide and conquer.

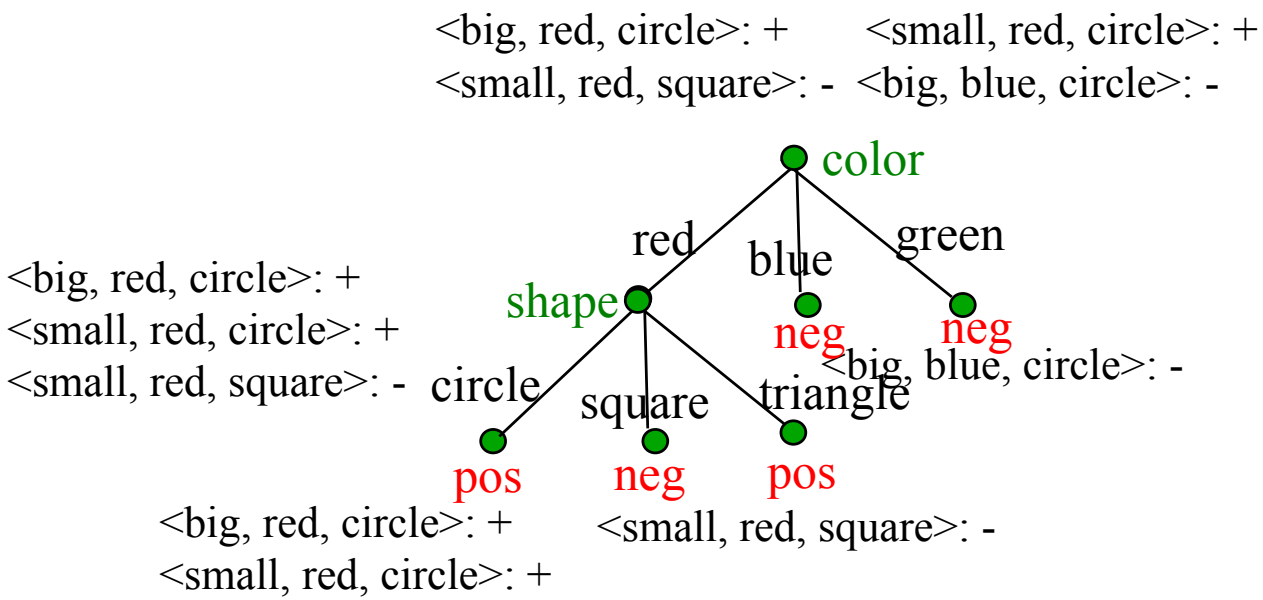
<big, red, circle>: +
<small, red, square>: -
<small, red, circle>: +
<big, blue, circle>: -




<big, red, circle>: +
<small, red, circle>: +
<small, red, square>: -

# Top-Down Decision Tree Induction

- Recursively build a tree top-down by divide and conquer.



# Decision Tree Induction Pseudocode

DTree(*examples*, *features*) returns a tree
  If all *examples* are in one category, return a leaf node with that category label.
  Else if the set of *features* is empty, return a leaf node with the category label that
      is the most common in examples.
  Else pick a feature $F$ and create a node $R$ for it
      For each possible value $v_i$ of $F$:
          Let $examples_i$ be the subset of examples that have value $v_i$ for $F$
          Add an out-going edge $E$ to node $R$ labeled with the value $v_i$.
          If $examples_i$ is empty
              then attach a leaf node to edge $E$ labeled with the category that
                  is the most common in *examples*.
              else call DTree($examples_i$ , $features - \{F\}$) and attach the resulting
                  tree as the subtree under edge $E$.
      Return the subtree rooted at $R$.

# Decision Tree Induction Pseudocode

DTree(*examples*, *features*) returns a tree
  If all *examples* are in one category, return a leaf node with that category label.
  Else if the set of *features* is empty, return a leaf node with the category label that
      is the most common in examples.
  Else pick a feature $F$ and create a node $R$ for it
      For each possible value $v_i$ of $F$:
          Let $examples_i$ be the subset of examples that have value $v_i$ for $F$
          Add an out-going edge $E$ to node $R$ labeled with the value $v_i$.
          If $examples_i$ is empty
              then attach a leaf node to edge $E$ labeled with the category that
                  is the most common in *examples*.
              else call DTree($examples_i$ , $features - \{F\}$) and attach the resulting
                  tree as the subtree under edge $E$.
      Return the subtree rooted at $R$.

# Picking a Good Split Feature

- Goal is to have the resulting tree be as small as possible, per Occam's razor.
- Finding a minimal decision tree (nodes, leaves, or depth) is an NP-hard optimization problem.
- Top-down divide-and-conquer method does a greedy search for a simple tree but does not guarantee to find the smallest.
  - General lesson in ML: "Greed is good."
- Want to pick a feature that creates subsets of examples that are relatively "pure" in a single class so they are "closer" to being leaf nodes.
- There are a variety of heuristics for picking a good test, a popular one is based on information gain that originated with the ID3 system of Quinlan (1979).

# Entropy

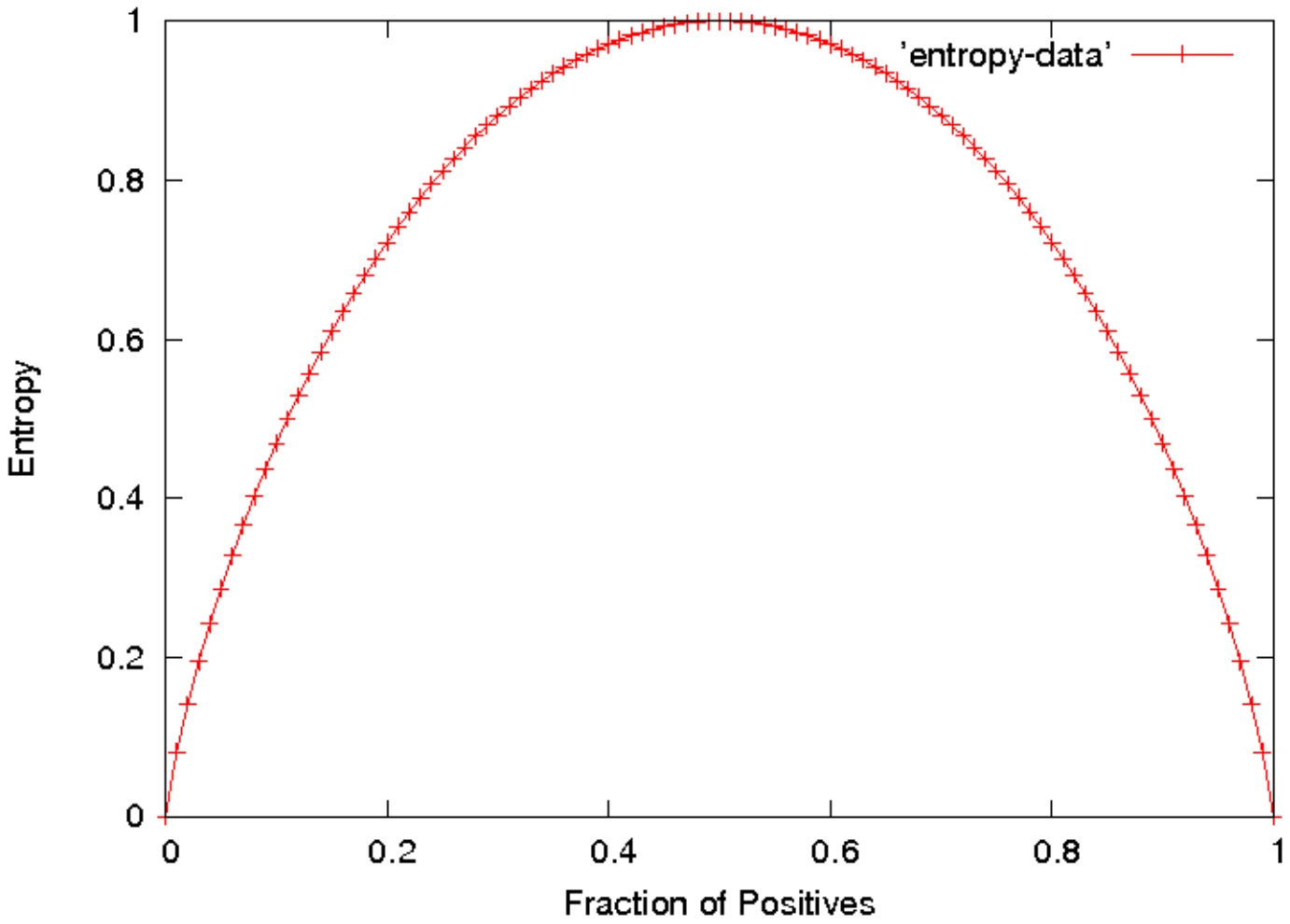
- Entropy (disorder, impurity) of a set of examples, S, relative to a binary classification is:

$$Entropy(S) = -p_1 \log_2(p_1) - p_0 \log_2(p_0)$$

  where $p_1$ is the fraction of positive examples in S and $p_0$ is the fraction of negatives.
- If all examples are in one category, entropy is zero (we define $0 \cdot \log(0)=0$)
- If examples are equally mixed ($p_1=p_0=0.5$), entropy is a maximum of 1.
- Entropy can be viewed as the number of bits required on average to encode the class of an example in *S* where data compression (e.g. Huffman coding) is used to give shorter codes to more likely cases.
- For multi-class problems with c categories, entropy generalizes to:

$$Entropy(S) = \sum_{i=1}^{c} -p_i \log_2(p_i)$$

# Entropy Plot for Binary Classification



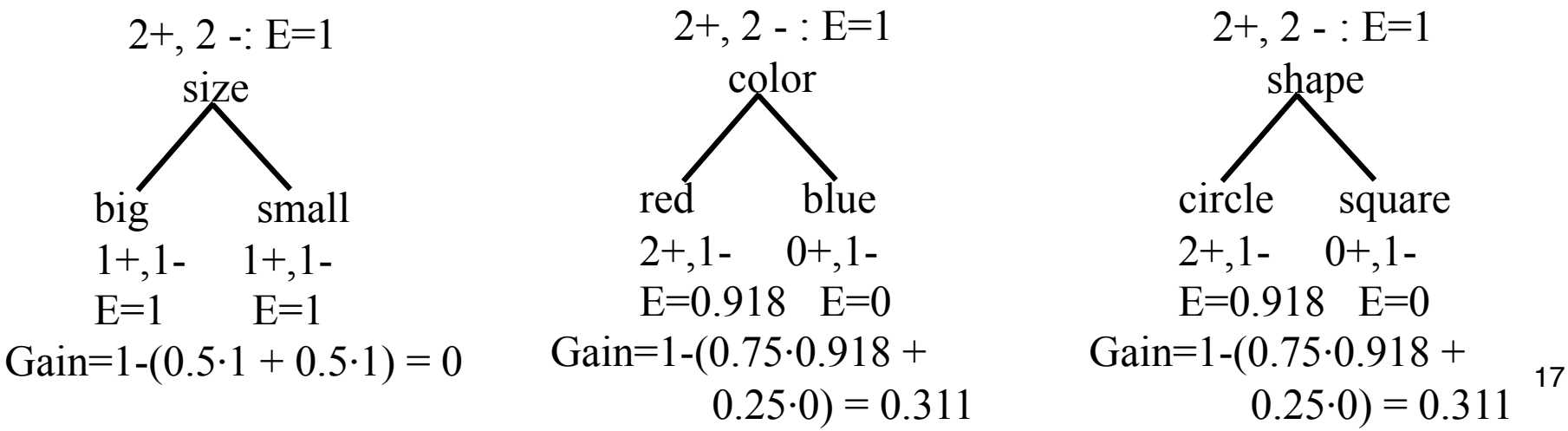
# Information Gain

- The information gain of a feature $F$ is the expected reduction in entropy resulting from splitting on this feature.

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

where $S_v$ is the subset of $S$ having value $v$ for feature $F$.

- Entropy of each resulting subset weighted by its relative size.
- Example:
  - <big, red, circle>: +  <small, red, circle>: +
  - <small, red, square>: -  <big, blue, circle>: -



# Hypothesis Space Search

- Performs ***batch learning*** that processes all training instances at once rather than ***incremental learning*** that updates a hypothesis after each example.
- Performs hill-climbing (greedy search) that may only find a locally-optimal solution. Guaranteed to find a tree consistent with any conflict-free training set (i.e. identical feature vectors always assigned the same class), but not necessarily the simplest tree.
- Finds a single discrete hypothesis, so there is no way to provide confidences or create useful queries.

# Continuous features

Use binary split of the current interval using the same impurity measure as for discrete attributes

| 64 | 65 | 68 | 69 | 70 | 71 | 72 | 75 | 80 | 81 | 83 | 85 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| yes | no | yes | yes | yes | no | no yes | yes yes | no | yes | yes | no |

(Repeated values have been collapsed together.) There are only 11 possible positions for the breakpoint—8 if the breakpoint is not allowed to separate items of the same class. The information gain for each can be calculated in the usual way. For example, the test *temperature* < 71.5 produces four *yes*'s and two *no*'s, whereas *temperature* > 71.5 produces five *yes*'s and three *no*'s, and so the information value of this test is

$$\text{info}([4,2],[5,3]) = (6/14) \times \text{info}([4,2]) + (8/14) \times \text{info}([5,3]) = 0.939 \text{ bits}.$$

# Missing feature values

- Remove the instance
- Replace with the most common (mode, mean) value
- Replace with the most common (mode, mean) value w.r.t. a class
- Decision trees: use weighted Impurity measure (add relative increment to each atribute value

# Bias in Decision-Tree Induction

- Information-gain gives a bias for trees with minimal depth.
- Implements a search (preference) bias instead of a language (restriction) bias.

# Learning Regression Trees

# Learning Regression Trees

**Example 5.4 (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 |

Goal: to construct a regression tree that will help you determine a reasonable price for your next purchase.

# Regression Trees: Impurity measure

- Need for another impurity measure

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

- Weighted average variance

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

# Regression tree growing

- To minimize the square error on the learning sample, the prediction at a leaf is the average output of the learning cases reaching that leaf
- Impurity of a sample is defined by the variance of the output in that sample:

$$I(LS)=\mathrm{var}_{y|LS}\{y\}=\mathrm{E}_{y|LS}\{(y-\mathrm{E}_{y|LS}\{y\})^2\}$$

- The best split is the one that reduces the most variance:

$$\Delta I(LS,A) = \mathrm{var}_{y|LS}\{y\} - \sum_a \frac{|LS_a|}{|LS|}\mathrm{var}_{y|LS_a}\{y\}$$

# Learning Regression Trees

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 Regression Trees

There are three features, hence three possible splits:

Model [A100,B3,E112,M102,T202]
[1051,1770,1900][4513][77][870][99,270,625]
means : 1574, 4513, 77, 870 and 331,
weighted average of squared means $3.21*10^6$

Condition [excellent,good, fair]
[1770,4513][270,870,1051,1900][77,99,625]
means : 3142, 1023 and 267, weighted average $2.68*10^6$

Leslie [yes,no]
[625,870,1900][77,99,270,1051,1770,4513]
means : 1132 and 1297, weighted average $1.55*10^6$

# Learning Regression Trees

There are three features, hence three possible splits:

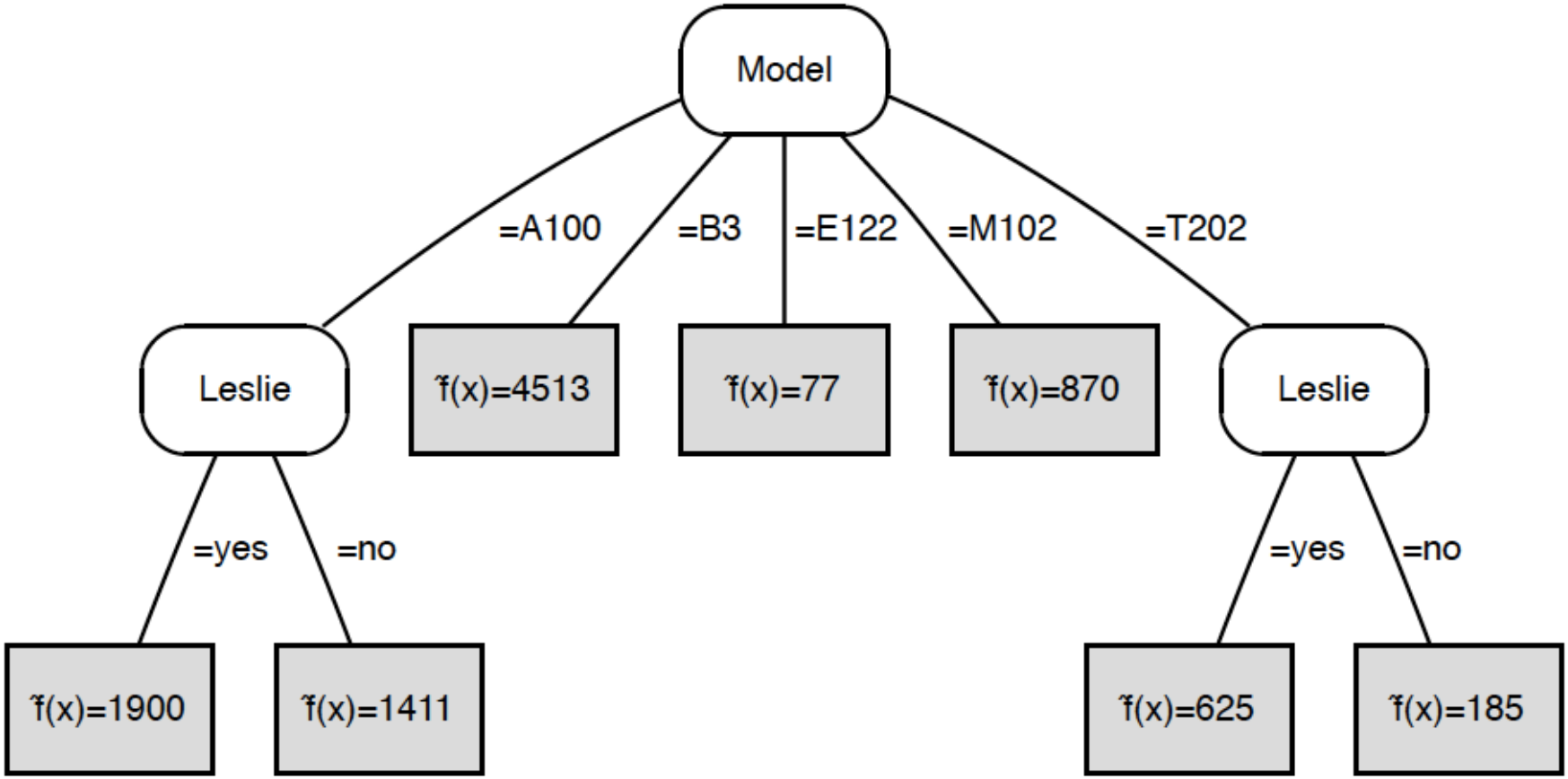
Model [A100,B3,E112,M102,T202]
[1051,1770,1900][4513][77][870][99,270,625]
means : 1574, 4513, 77, 870 and 331,
weighted average of squared means $3.21*10^6$ Model is a winner

Condition [excellent,good, fair]
[1770,4513][270,870,1051,1900][77,99,625]
means : 3142, 1023 and 267, weighted average $2.68*10^6$

Leslie [yes,no]
[625,870,1900][77,99,270,1051,1770,4513]
means : 1132 and 1297, weighted average $1.55*10^6$

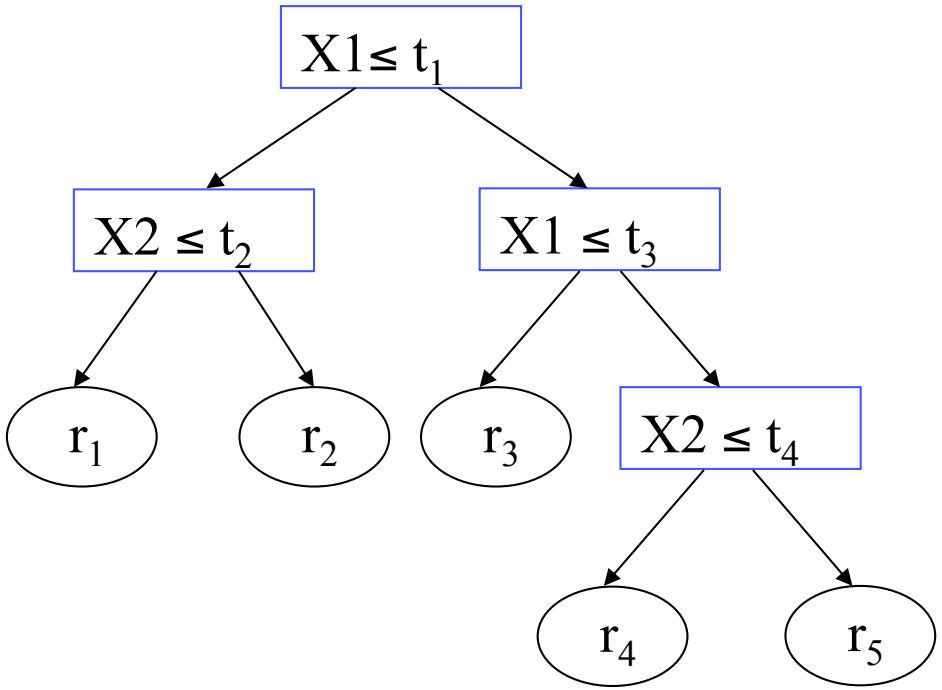
Continue in the same manner and receive

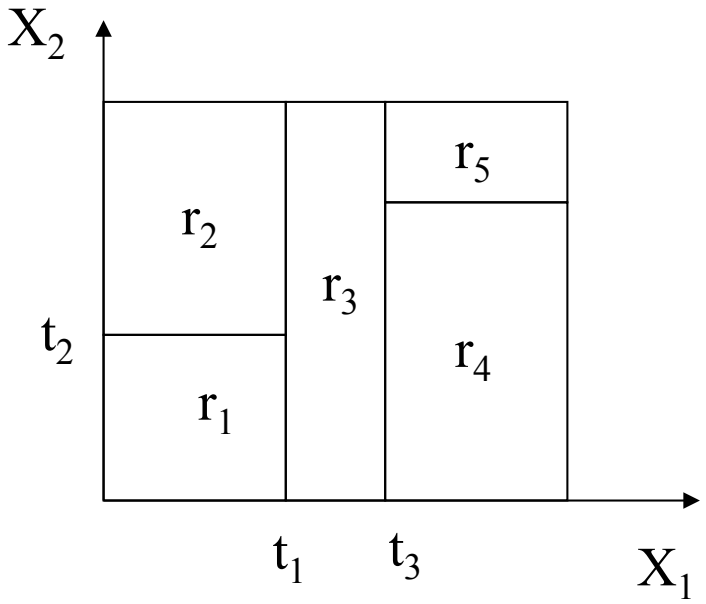
# Result


Model
=A100
=B3
=E122
=M102
=T202
Leslie
f(x)=4513
f(x)=77
f(x)=870
Leslie
=yes
=no
=yes
=no
f(x)=1900
f(x)=1411
f(x)=625
f(x)=185

# Regression trees (2)

- A regression tree is a piecewise constant function of the input attributes



# Regression Trees: Evaluation

- descend the path from root to a leaf
- how accurate the prediction is?
- measure a difference between a correct value and the value in the leaf

**Table 5.8 Performance measures for numeric prediction*.**

| Performance measure | Formula |
|---|---|
| mean-squared error | $\frac{(p_1-a_1)^2+\ldots+(p_n-a_n)^2}{n}$ |
| root mean-squared error | $\sqrt{\frac{(p_1-a_1)^2+\ldots+(p_n-a_n)^2}{n}}$ |
| mean absolute error | $\frac{\lvert p_1-a_1\rvert+\ldots+\lvert p_n-a_n\rvert}{n}$ |
| relative squared error | $\frac{(p_1-a_1)^2+\ldots+(p_n-a_n)^2}{(a_1-\bar{a})^2+\ldots+(a_n-\bar{a})^2}$, where $\bar{a}=\frac{1}{n}\sum_i a_i$ |
| root relative squared error | $\sqrt{\frac{(p_1-a_1)^2+\ldots+(p_n-a_n)^2}{(a_1-\bar{a})^2+\ldots+(a_n-\bar{a})^2}}$ |
| relative absolute error | $\frac{\lvert p_1-a_1\rvert+\ldots+\lvert p_n-a_n\rvert}{\lvert a_1-\bar{a}\rvert+\ldots+\lvert a_n-\bar{a}\rvert}$ |
| correlation coefficient | $\frac{S_{PA}}{\sqrt{S_P S_A}}$, where $S_{PA}=\frac{\sum_i (p_i-\bar{p})(a_i-\bar{a})}{n-1}$, $S_p=\frac{\sum_i (p_i-\bar{p})^2}{n-1}$, and $S_A=\frac{\sum_i (a_i-\bar{a})^2}{n-1}$ |

* $p$ are predicted values and $a$ are actual values.

# Differences from classification trees

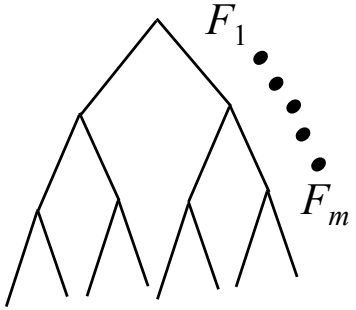
- Prediction is computed as the **average** of numerical target variable in the rectangle (in CT it is majority vote)
- Impurity measured by **sum of squared deviations** from leaf mean
- Performance measured by RMSE (root mean squared error)

# History of Decision-Tree Research

- Hunt and colleagues use exhaustive search decision-tree methods (CLS) to model human concept learning in the 1960's.
- In the late 70's, Quinlan developed ID3 with the information gain heuristic to learn expert systems from examples.
- Simulataneously, Breiman and Friedman and colleagues develop CART (Classification and Regression Trees), similar to ID3.
- In the 1980's a variety of improvements are introduced to handle noise, continuous features, missing features, and improved splitting criteria. Various expert-system development tools results.
- Quinlan's updated decision-tree package (C4.5) released in 1993.
- Weka includes Java version of C4.5 called J48.

# Computational Complexity

- Worst case builds a complete tree where every path test every feature. Assume *n* examples and *m* features.




Maximum of *n* examples spread across all nodes at each of the *m* levels

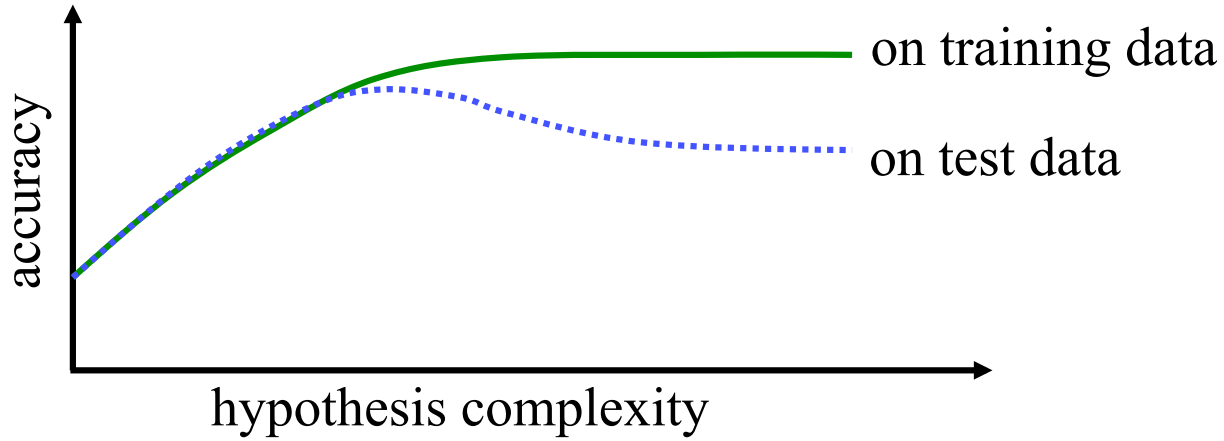
- At each level, *i*, in the tree, must examine the remaining *m-i* features for each instance at the level to calculate info gains.

$$\sum_{i=1}^{m} i \cdot n = O(nm^2)$$

- However, learned tree is rarely complete (number of leaves is ≤ *n*). In practice, complexity is linear in both number of features (*m*) and number of training examples (*n*).

# Overfitting

- Learning a tree that classifies the training data perfectly may not lead to the tree with the best generalization to unseen data.
  - There may be noise in the training data that the tree is erroneously fitting.
  - The algorithm may be making poor decisions towards the leaves of the tree that are based on very little data and may not reflect reliable trends.
- A hypothesis, $h$, is said to overfit the training data is there exists another hypothesis which, $h'$, such that $h$ has less error than $h'$ on the training data but greater error on independent test data.

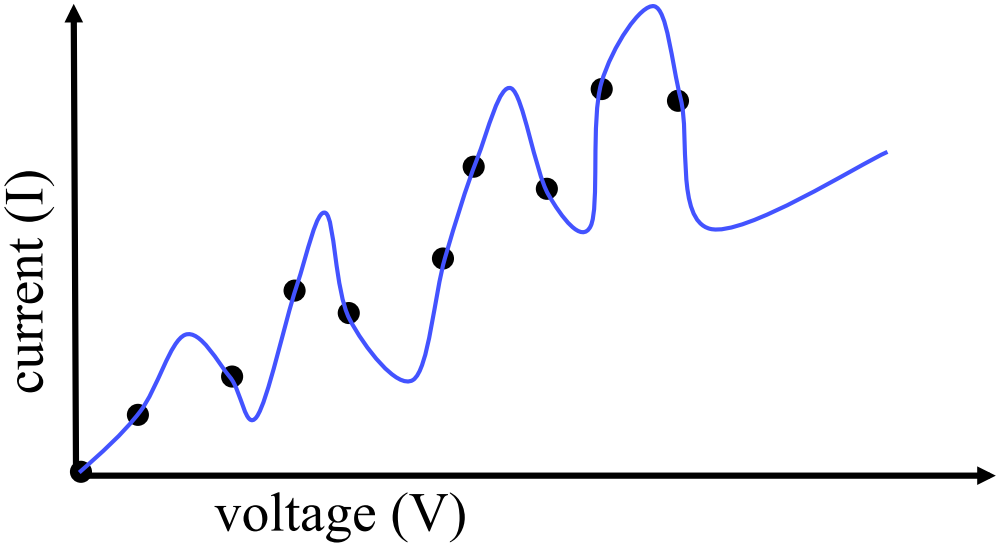


# Overfitting Example

**Testing Ohms Law: V = IR (I = (1/R)V)**

Experimentally measure 10 points

Fit a curve to the Resulting data.

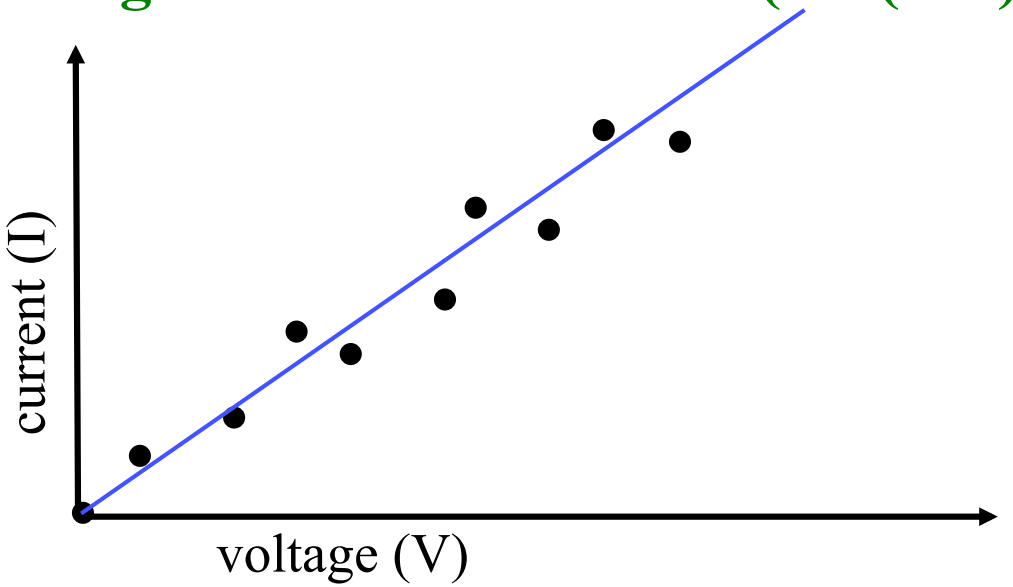



Perfect fit to training data with an 9th degree polynomial
(can fit *n* points exactly with an *n*-1 degree polynomial)

**Ohm was wrong, we have found a more accurate function!**

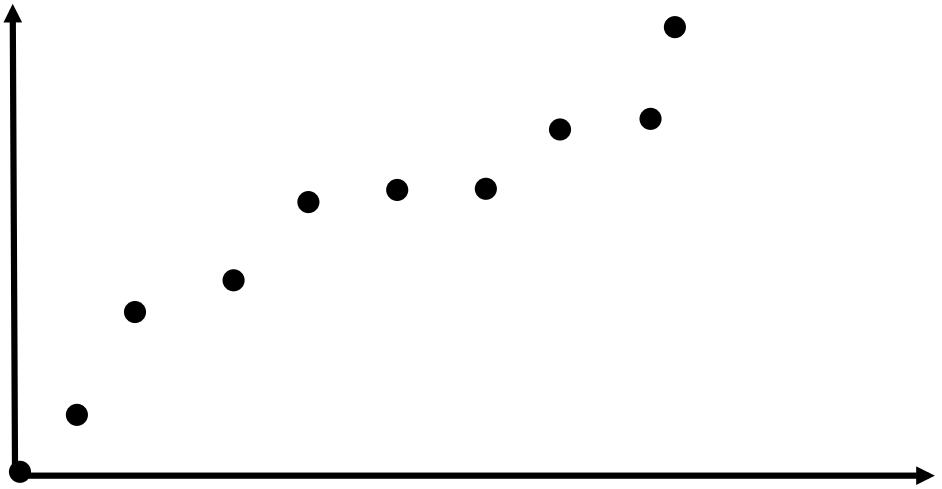
# Overfitting Example

**Testing Ohms Law: V = IR (I = (1/R)V)**



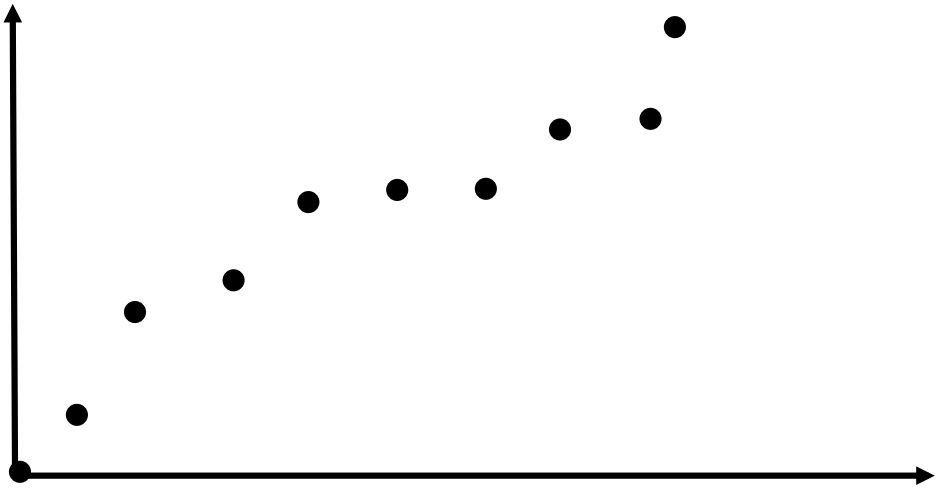

Better generalization with a linear function that fits training data less accurately.

# Bias-variance tradeoff



Another example

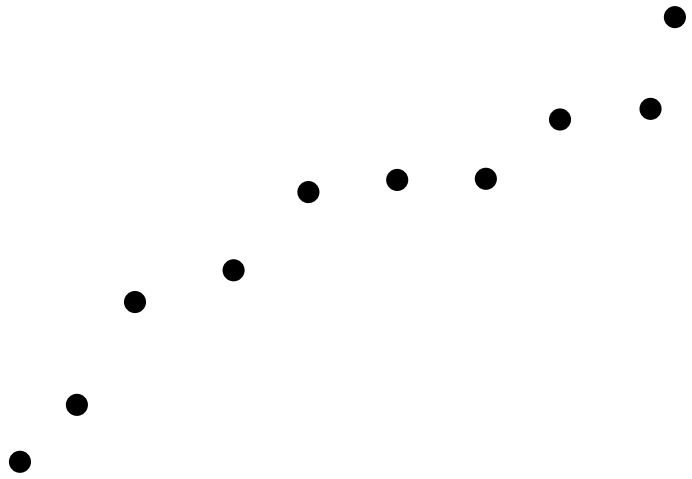
# Bias-variance tradeoff



Linear function works well but
Cubic seems better

Is there any general view to the problem,
preferably with a theoretical background?

# Bias-variance tradeoff

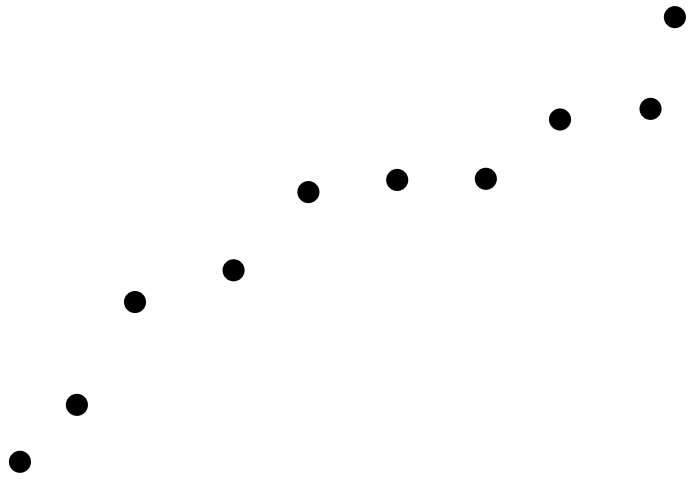


$y = f(x) + e, \quad E[e] = 0, \mathrm{Var}[e] = s^2$

f'() … estimate of f() learned by a classifier

$E[\,(y-f'(x))^2\,] = \mathrm{Bias}[\,f'(x)\,] + \mathrm{Var}[\,f'(x)\,] + s^2$

# Bias-variance tradeoff

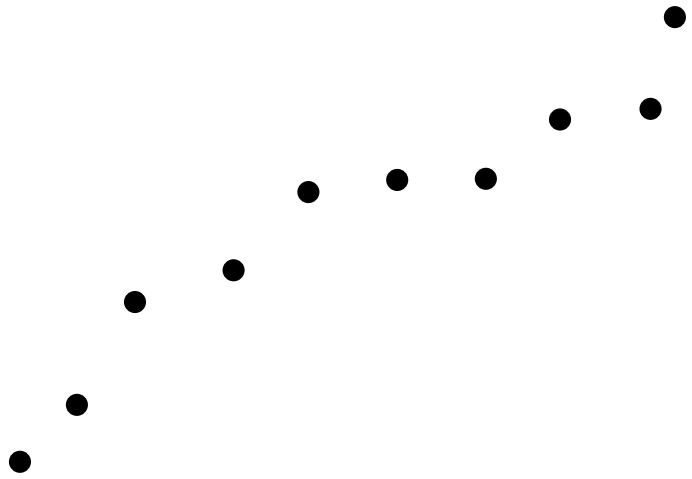


$y = f(x) + e, \quad E[e] = 0, \mathrm{Var}[e] = s^2$

f'() … estimate of f() learned by a classifier

$E[\,(y-f'(x))^2\,] = \mathrm{Bias}[\,f'(x)\,] + \mathrm{Var}[\,f'(x)\,] + s^2$

# Bias-variance tradeoff

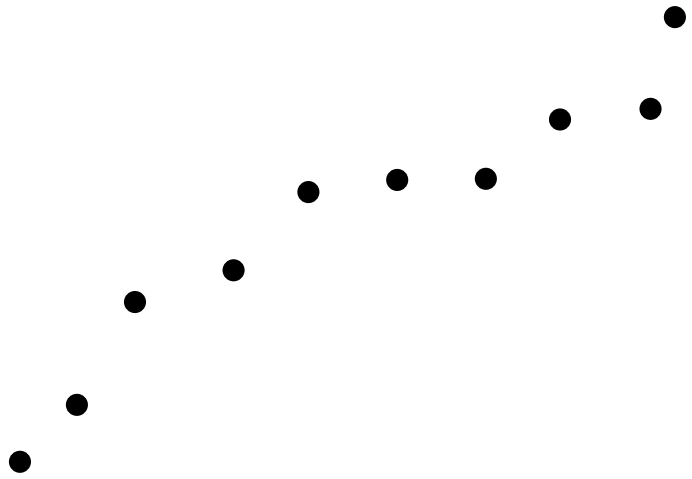


$y = f(x) + e, \quad E[e] = 0, \mathrm{Var}[e] = s^2$

f'() … estimate of f() learned by a classifier

$E[\,(y-f'(x))^2\,] = \mathrm{Bias}[\,f'(x)\,] + \mathrm{Var}[\,f'(x)\,] + s^2$

# Bias-variance tradeoff

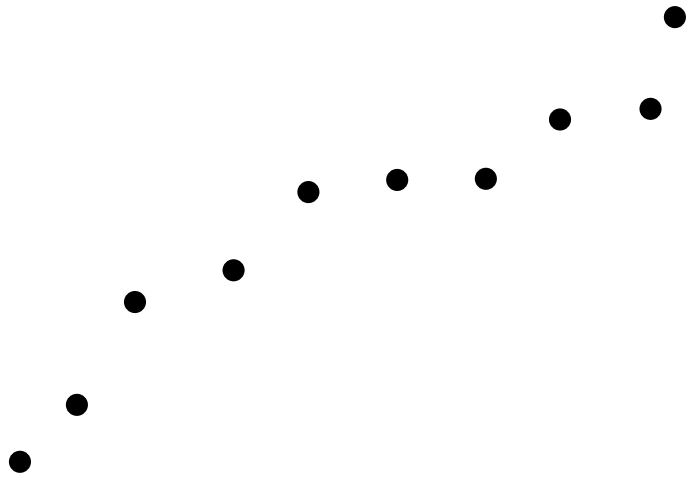


$y = f(x) + e, \quad E[e] = 0, \mathrm{Var}[e] = s^2$

f'() … estimate of f() learned by a classifier

$E[\,(y-f'(x))^2\,] = \mathrm{Bias}[\,f'(x)\,] + \mathrm{Var}[\,f'(x)\,] + s^2$

# Bias-variance tradeoff

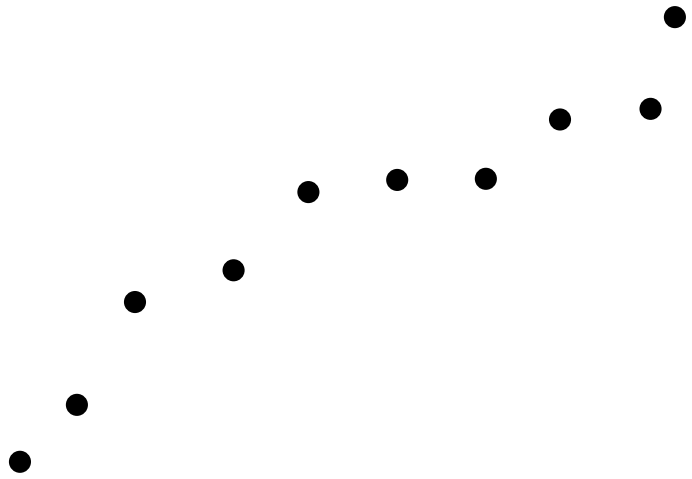


$y = f(x) + e, \quad E[e] = 0, \mathrm{Var}[e] = s^2$

f'() … estimate of f() learned by a classifier

$E[\,(y-f'(x))^2\,] = \mathrm{Bias}[\,f'(x)\,] + \mathrm{Var}[\,f'(x)\,] + s^2$

# Bias-variance tradeoff



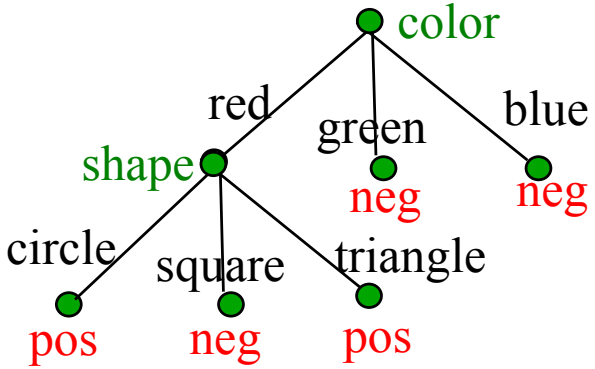
$y = f(x) + e, \quad E[e] = 0, \mathrm{Var}[e] = s^2$

f'() … estimate of f() learned by a classifier

$E[\,(y-f'(x))^2\,] = \mathrm{Bias}[\,f'(x)\,] + \mathrm{Var}[\,f'(x)\,] + s^2$

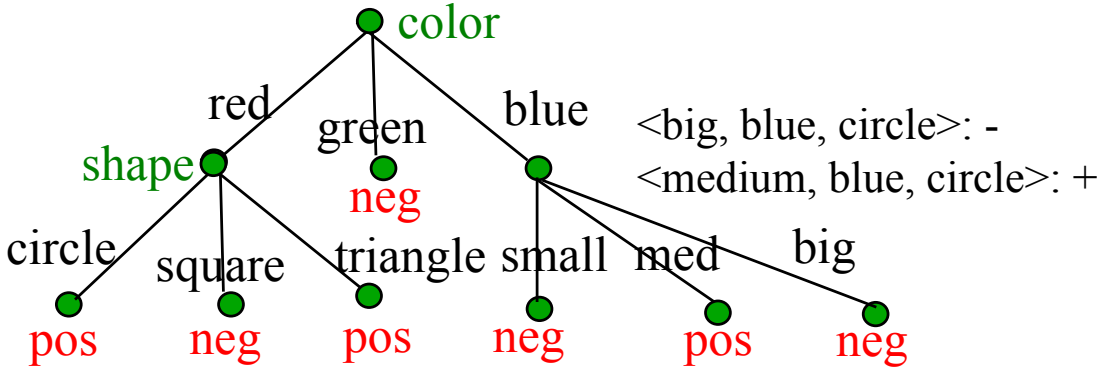
# Overfitting Noise in Decision Trees

- Category or feature noise can easily cause overfitting.
  - Add noisy instance <medium, blue, circle>: pos (but really neg)



# Overfitting Noise in Decision Trees

- Category or feature noise can easily cause overfitting.
  - Add noisy instance <medium, blue, circle>: pos (but really neg)




- Noise can also cause different instances of the same feature vector to have different classes. Impossible to fit this data and must label leaf with the majority class.
  - <big, red, circle>: neg (but really pos)
- Conflicting examples can also arise if the features are incomplete and inadequate to determine the class or if the target concept is non-deterministic.

# Overfitting Prevention (Pruning) Methods

- Two basic approaches for decision trees
  - Prepruning: Stop growing tree as some point during top-down construction when there is no longer sufficient data to make reliable decisions.
  - Postpruning: Grow the full tree, then remove subtrees that do not have sufficient evidence.
- Label leaf resulting from pruning with the majority class of the remaining data, or a class probability distribution.
- Method for determining which subtrees to prune:
  - Cross-validation: Reserve some training data as a hold-out set (***validation set***, ***tuning set***) to evaluate utility of subtrees.
  - Statistical test: Use a statistical test on the training data to determine if any observed regularity can be dismisses as likely due to random chance.
  - Minimum description length (MDL): Determine if the additional complexity of the hypothesis is less complex than just explicitly remembering any exceptions resulting from pruning.

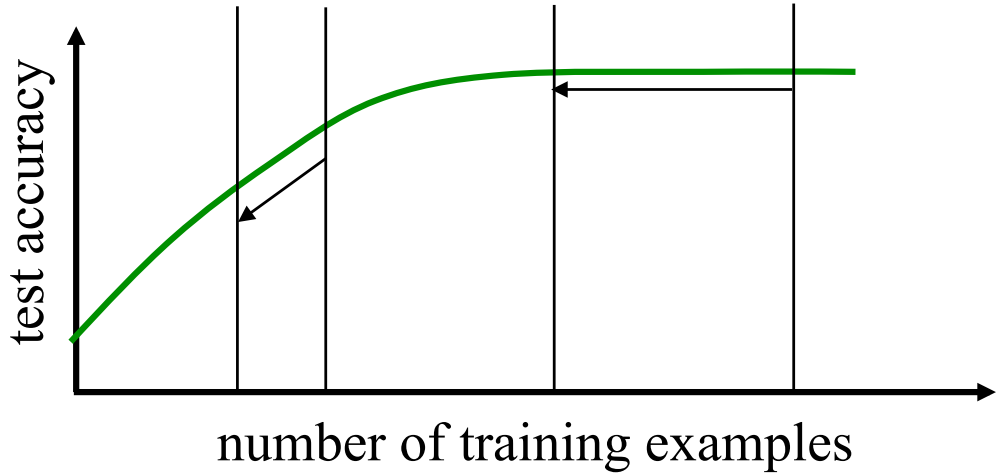
# Reduced Error Pruning

- A post-pruning, cross-validation approach.

Partition training data in "grow" and "validation" sets.
Build a complete tree from the "grow" data.
Until accuracy on validation set decreases do:
    For each non-leaf node, n, in the tree do:
        Temporarily prune the subtree below n and replace it with a leaf labeled with the current majority class at that node.
        Measure and record the accuracy of the pruned tree on the validation set.
    Permanently prune the node that results in the greatest increase in accuracy on the validation set.

# Issues with Reduced Error Pruning

- The problem with this approach is that it potentially "wastes" training data on the validation set.
- Severity of this problem depends where we are on the learning curve:



# Cross-Validating without Losing Training Data

- If the algorithm is modified to grow trees breadth-first rather than depth-first, we can stop growing after reaching any specified tree complexity.
- First, run several trials of reduced error-pruning using different random splits of grow and validation sets.
- Record the complexity of the pruned tree learned in each trial. Let $C$ be the average pruned-tree complexity.
- Grow a final tree breadth-first from all the training data but stop when the complexity reaches $C$.
- Similar cross-validation approach can be used to set arbitrary algorithm parameters in general.

# Additional Decision Tree Issues

- Better splitting criteria
  - Information gain (IG) prefers features with many values.
  - Gain Ratio : taking into account the number and size of daughter nodes into which an attribute splits the dataset, disregarding any information about the class ( = SplitInfo, using entropy measure used again), GR = IG/SplitInfo
  - Gini $2*p_{pos}*(1-p_{pos})$ , sqrt(Gini), Variance, …
- Features with costs
- Misclassification costs
- Incremental learning
- Mining large databases that do not fit in main memory
- Most common: C4.5 (here) and CART (at tutorials)

# C4.5

- Based on ID3 algorithm (Ross Quinlan)
- gain ratio used
- C4.5 ver.8 -> j48 (java)

Scheme of C4.5 algorithm:

Run several time and choose the best tree

Inner: Take L% of learning data randomly

Call ID3 (pre-pruning, see –m parameter)

Prune the tree (post-pruning, -cf)

Take T% of unseen learning data for validation

If validation criterion holds, exit

Otherwise add L% to L and go to Inner