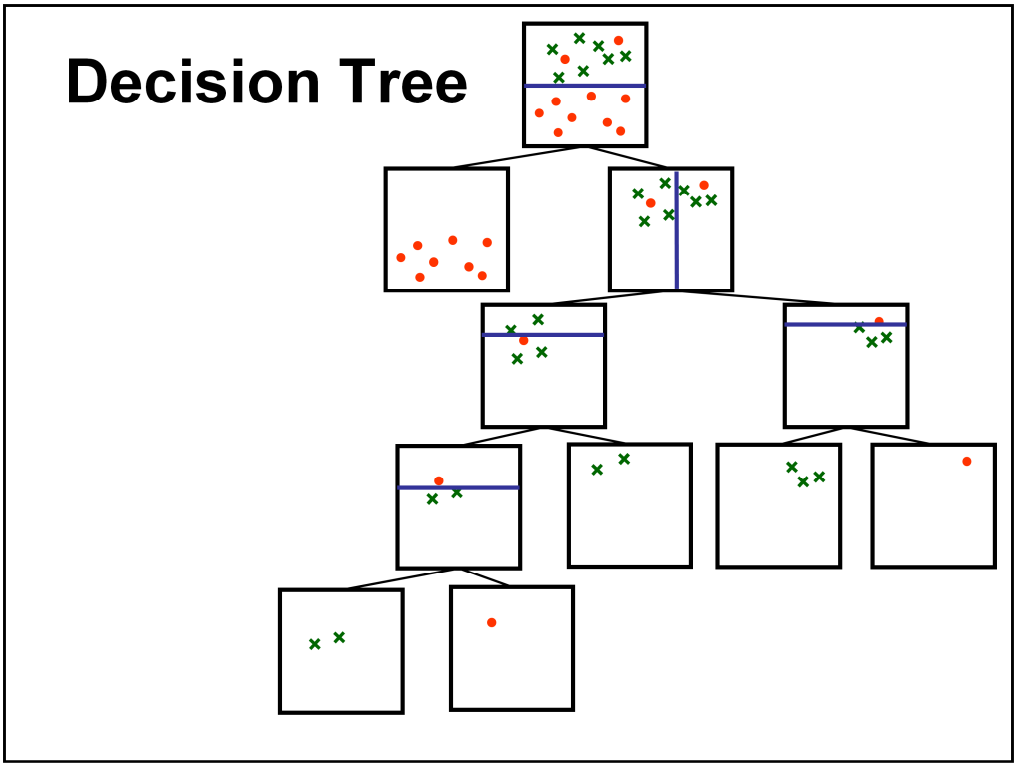
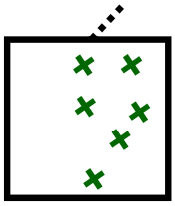
Decision Trees II

Revenge of the Entropy



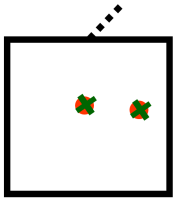
Note we're working with continuous splits here, which is an interesting problem we covered in the last lecture.

Pure and Impure Leaves and When to Stop Splitting



All data in node comes from one class:

We declare the node to be a leaf and stop splitting. Leaf will output the class of the data it contains



Data points have exactly the same attributes even though they are from *different* classes:

We cannot split any further. We still declare the node to be a leaf, but it outputs the class that is the majority of the classes in the node

What if points have exactly same classes? Output whichever is the majority. If there's no majority, you could fip a coin, or just pick a "default" class.

DT Algorithm (Continuous Attributes)

LearnTree(X,Y)

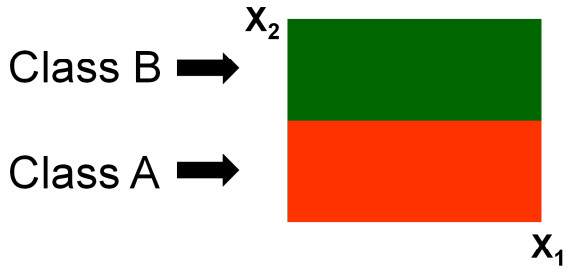
- Input:
  - Set X of R training vectors, each containing the values (x_1,..,x_M) *of* M attributes (X_1,..,X_M)
  - A vector Y of R elements, where y_j = class of the j^{th} datapoint
- If all the datapoints in X have the same class value y, return a leaf node that predicts y as output
- If all the datapoints in X have the same attribute value (x_1,..,x_M), return a leaf node that predicts the majority of the class values in Y as output
- Try all the possible attributes X_j and threshold t and choose the one, j^*, for which IG(Y|X_j,t) is maximum
- X_L, Y_L= set of datapoints for which $x_{j^*} < t$ and corresponding classes
- X_H, Y_H = set of datapoints for which $x_{j^*} >= t$ and corresponding classes
- Left Child ← LearnTree(X_L,Y_L)
- Right Child ← LearnTree(X_H,Y_H)

Decision Trees So Far

- **Given *R* observations from training data, each with *M* attributes *X* and a class attribute *Y*, construct a sequence of tests (decision tree) to predict the class attribute *Y* from the attributes *X***
- **Basic strategy for defining the tests ("when to split") is to maximize the information gain on the training data set at each node of the tree**
- **Problems (next):**
  - **The tree will end up being much too big → *pruning***
  - **Evaluating the tree on training data is dangerous (*overfitting)***

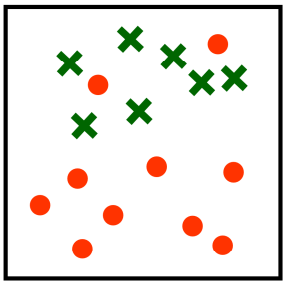
We saw that if we make enough splits, we can pretty much get 100% accuracy (with some exceptions if points of different classes have the same attributes)

The Overfitting Problem: Example



- **Suppose that, in an ideal world, class B is everything such that $X_2 \geq 0.5$ and class A is everything with $X_2 < 0.5$**
- **Note that attribute X_1 is irrelevant**
- **Seems like a decision tree would be trivial**

The Overfitting Problem: Example

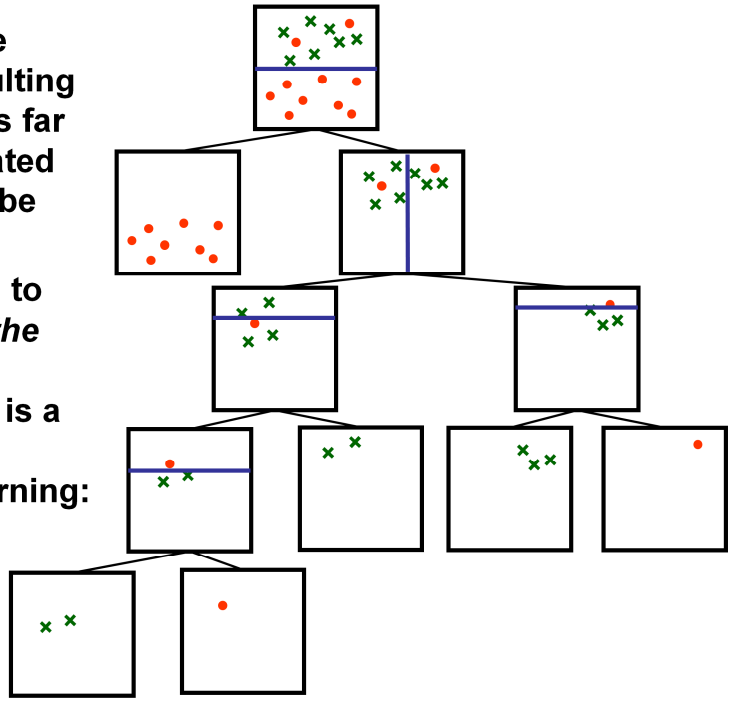


- **However, we collect training examples from the perfect world through some imperfect observation device**
- **As a result, the training data is corrupted by *noise***

Welcome to the real world, where data is noisy and the answers aren't in the back of the book.

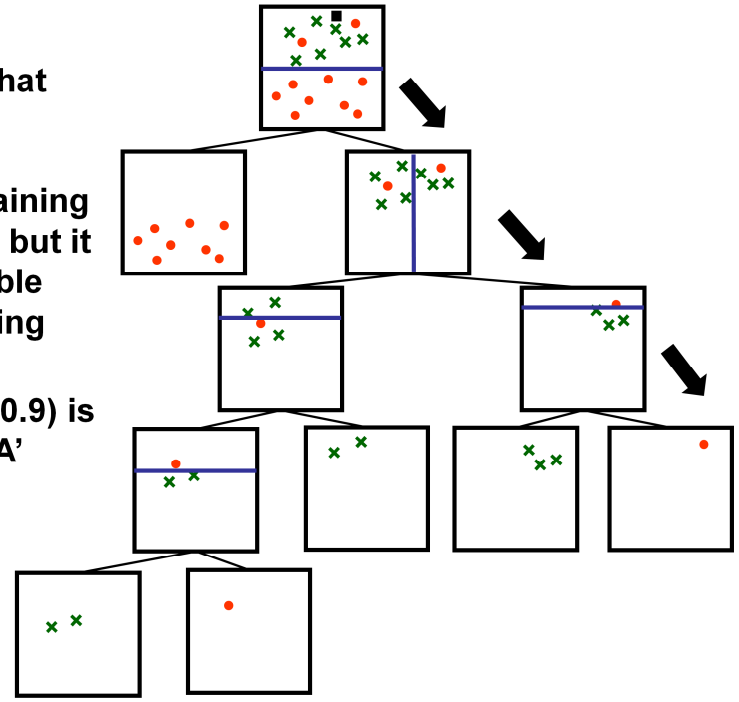
Because of the noise, the resulting decision tree is far more complicated than it should be

The learning algorithm tries to classify *all of the training set perfectly*. This is a fundamental problem in learning: *overfitting*

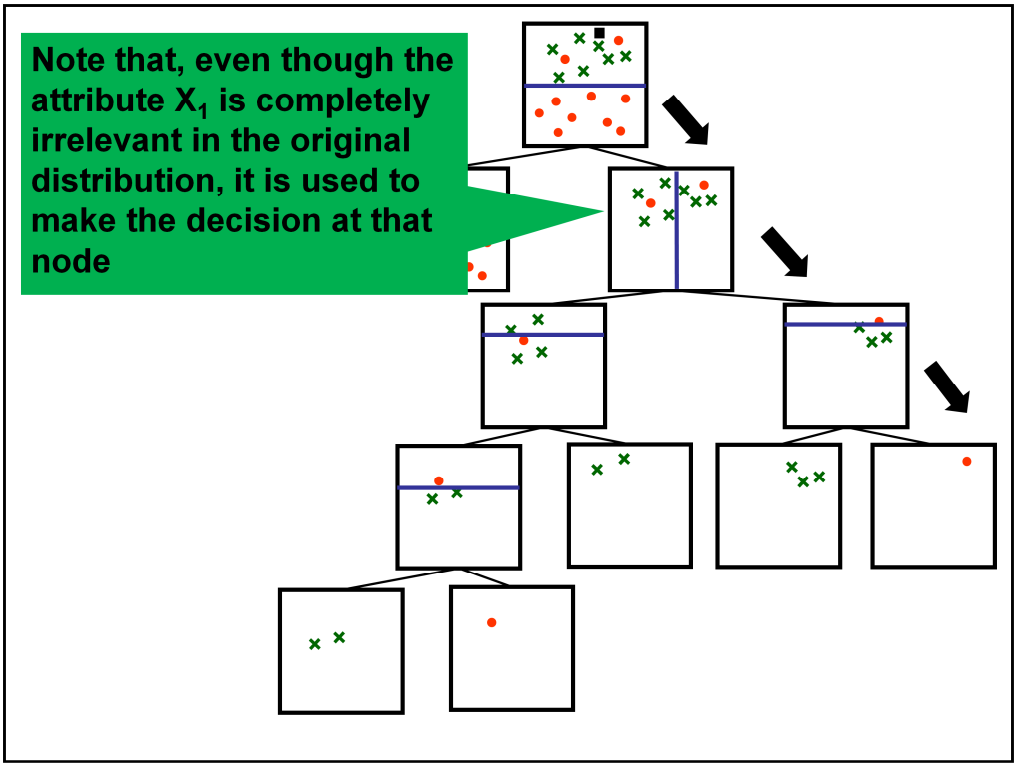


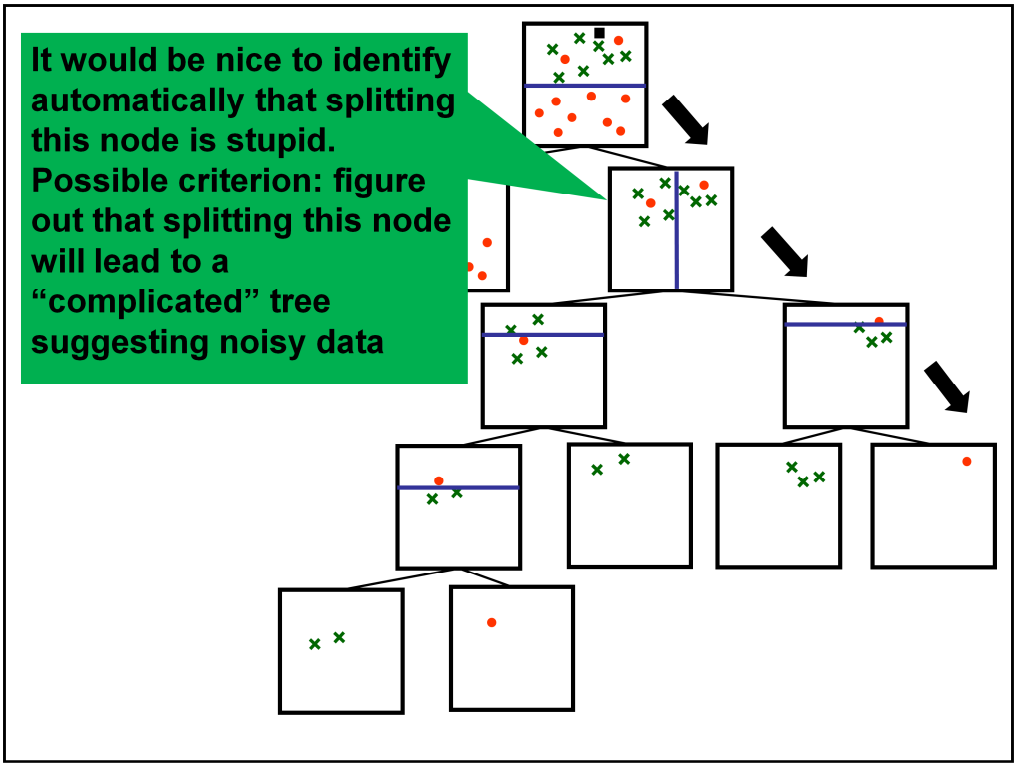
The effect of overfitting is that the tree is guaranteed to classify the training data perfectly, but it may do a terrible job at classifying new test data.

Example: (0.6,0.9) is classified as 'A'



This is not only a noise problem, but a matter of sparse data. Suppose you had one female age 37 with brown hair and a white blood cell count of X and a family history of cancer, and she has cancer. Does that mean that any 37-year-old women brunettes with wbc count X and a family history have cancer?


Note that, even though the attribute X_1 is completely irrelevant in the original distribution, it is used to make the decision at that node

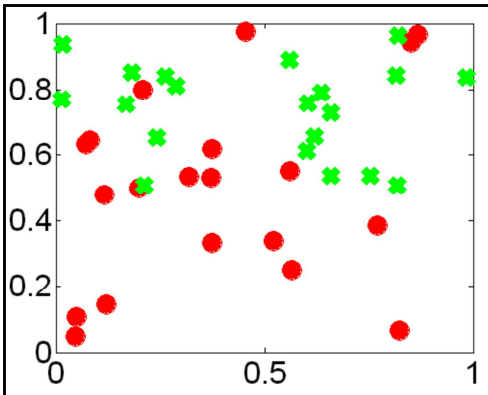

It would be nice to identify automatically that splitting this node is stupid. Possible criterion: figure out that splitting this node will lead to a "complicated" tree suggesting noisy data

Possible Overfitting Solutions

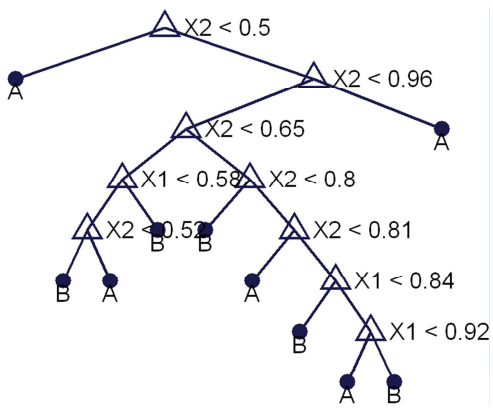
- **Grow tree based on training data (*unpruned* tree)**
- **Prune the tree by removing useless nodes based on:**
  - **Additional test data (not used for training)**
  - **Statistical significance tests**

Training and Test sets are a big deal. You never use all the data to train. Use some fraction of the data for training, and the other is testing.

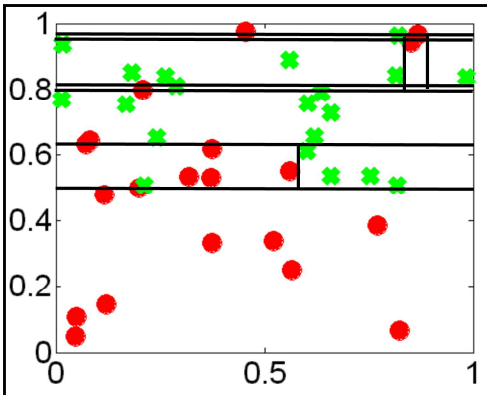
When you're training, you're automatically going to overfit. Any data sample will have bias and noise. So you use the test data to test its "real" accuracy.



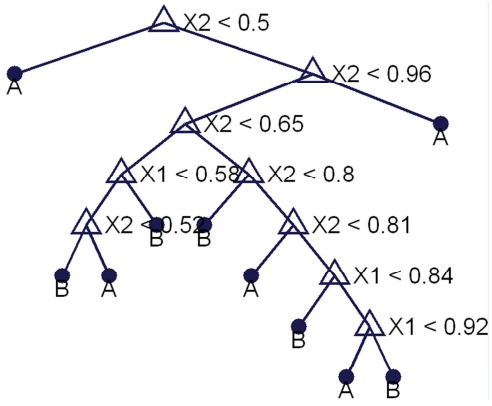
Training Data



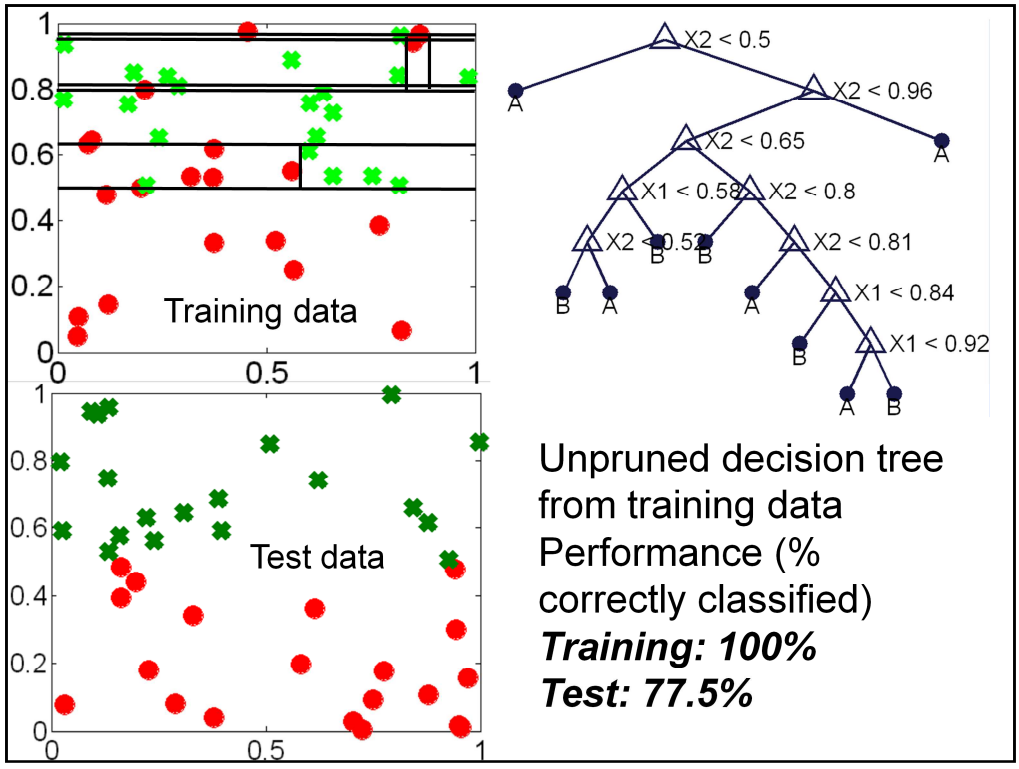
Unpruned decision tree from training data



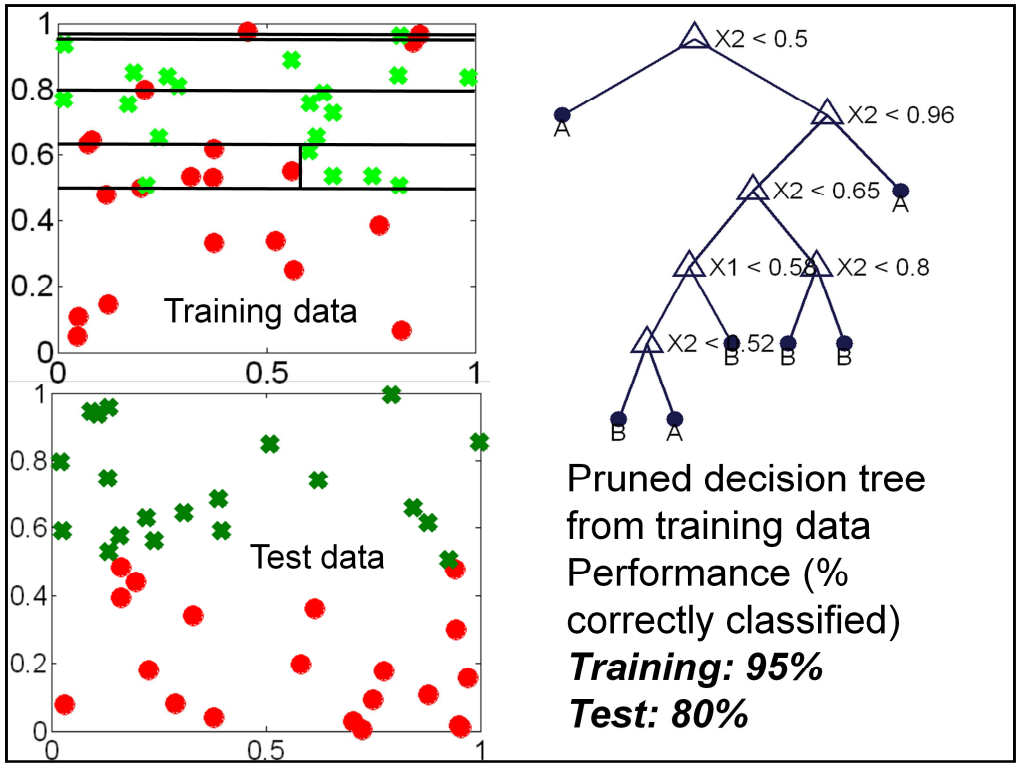
Training data with the partitions induced by the decision tree (Notice the tiny regions at the top necessary to correctly classify the 'A' outliers!)



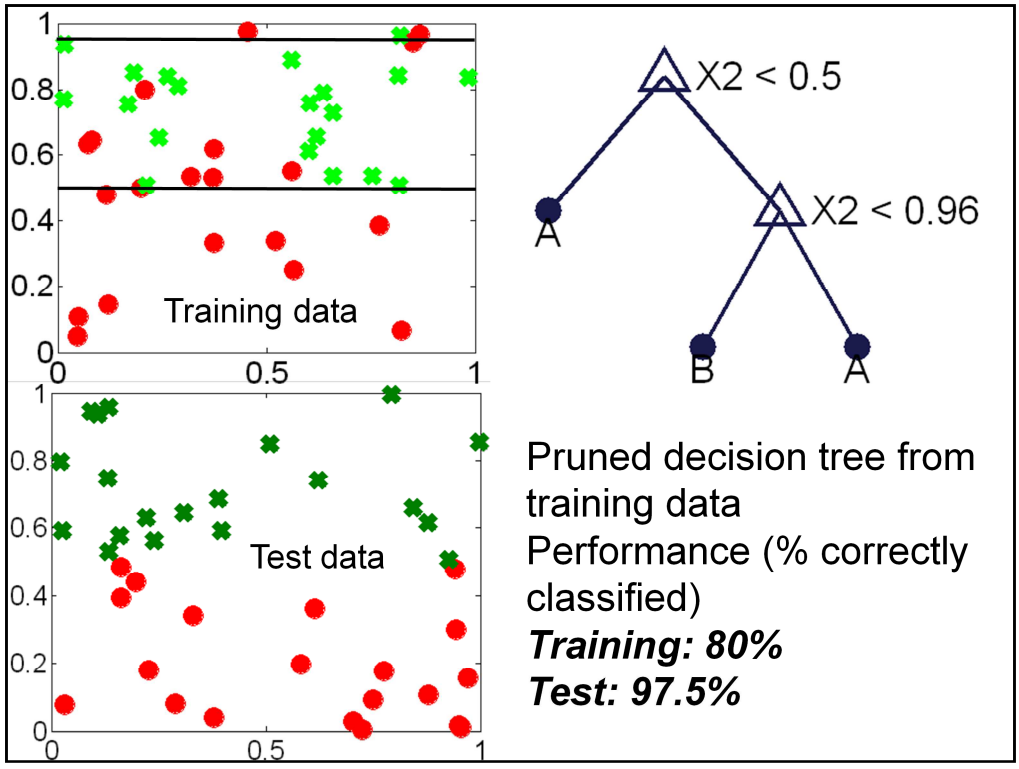
Unpruned decision tree from training data



Oh hey we have 100% accuracy, awesome! Oh wait.

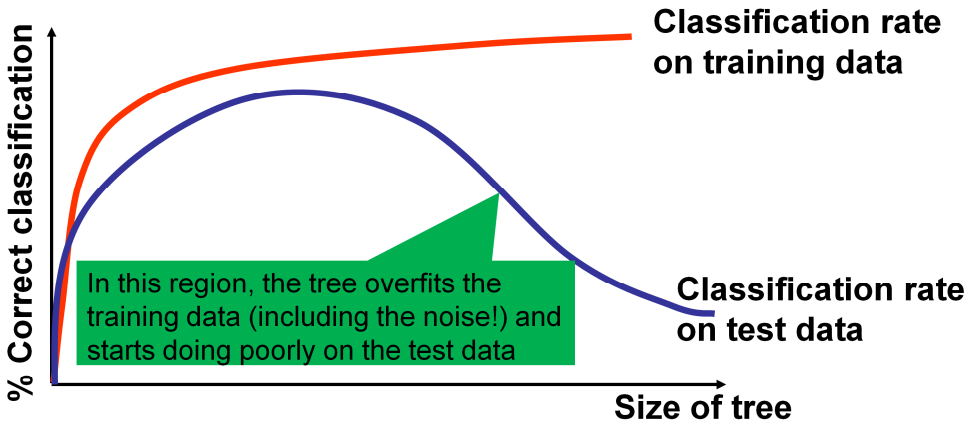


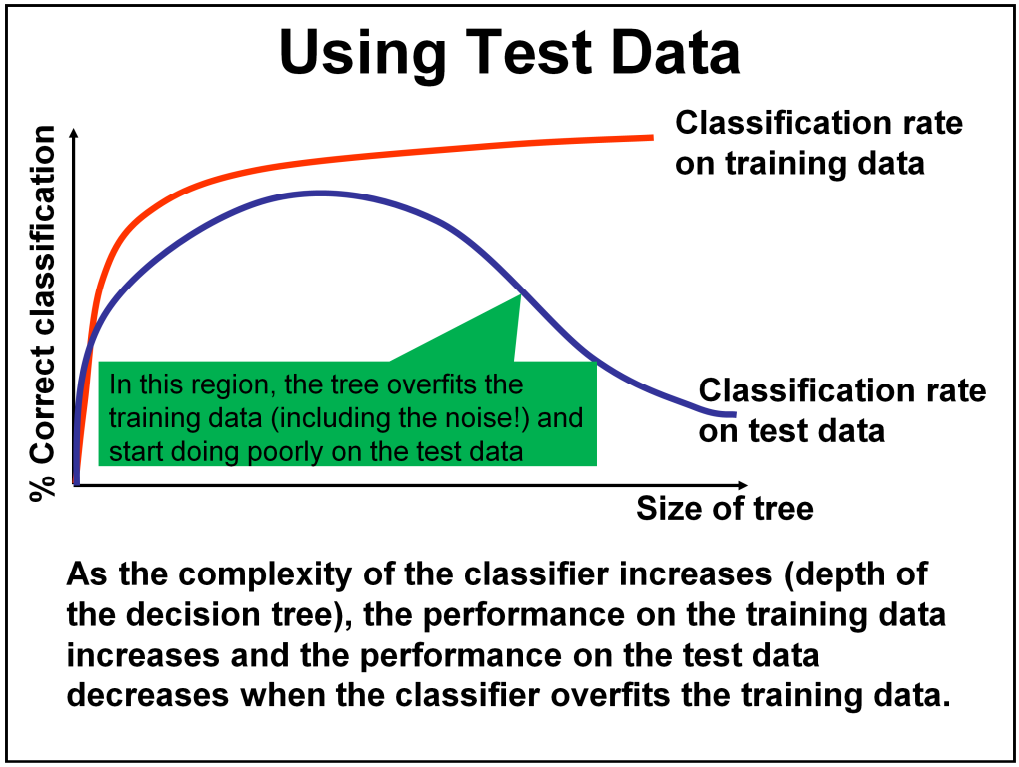
Well this improves our test a little bit, if we try pruning based on our training data (which still has bias)



Prune it a little more and the error is mostly from noise.

Using Test Data





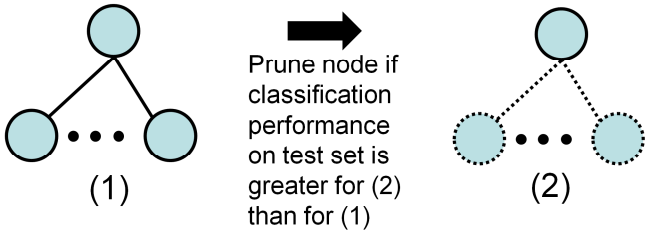
Why is testing usually below training accuracy? Because the training data is what we're actually trying to fit on.

Basic Questions

- How to choose the attribute/value to split on at each level of the tree?
- When to stop splitting? When should a node be declared a leaf?
- If a leaf node is impure, how should the class label be assigned?
- **If the tree is too large, how can it be pruned?**

Decision Tree Pruning

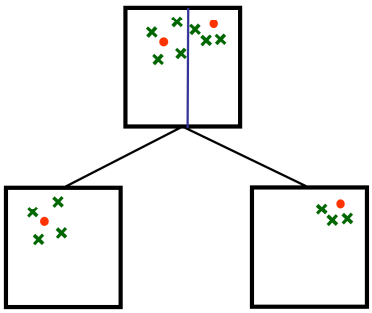
- **Construct the entire tree as before**
- **Starting at the leaves, recursively eliminate splits:**
  - **Evaluate performance of the tree on test data (also called validation data, or hold out data set)**
  - **Prune the tree if the classification performance increases by removing the split**



Possible Overfitting Solutions

- Grow tree based on training data (*unpruned* tree)
- Prune the tree by removing useless nodes based on:
  - Additional test data (not used for training)
  - **Statistical significance tests**

Detecting Useless Splits



# of A in root node is N_A=2
# of B in root node is N_B=7

# of A in left node is N_{AL}=1
# of B in left node is N_{BL}=4

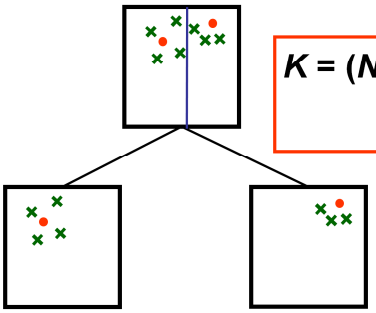
Problem is we split whenever the IG increases, but never check if change in entropy is *statistically significant*

Reasoning:

- **Proportion of data going to the left node: $p_L = (N_{AL} + N_{BL})/(N_A+N_B) = 5/9$**
- **Suppose now that the data is *completely randomly* distributed (i.e., it does not make sense to split):**
- **The expected number of class A in the left node: $N'_{AL} = N_A \times p_L = 10/9$**
- **The expected number of class B in the left node: $N'_{BL} = N_B \times p_L = 35/9$**
- ***Question:* Are N_{AL} and N_{BL} sufficiently different from N'_{AL} and N'_{BL}. *If not, it means the split is not statistically significant and we should not split.***

This keeps us from going too far down in the tree.

Detecting Useless Splits



Measure of statistically significance:

$$K = (N'_{AL} - N_{AL})^2/N'_{AL} + (N'_{BL} - N_{BL})^2/N'_{BL} + (N'_{AR} - N_{AR})^2/N'_{AR} + (N'_{BR} - N_{BR})^2/N'_{BR}$$

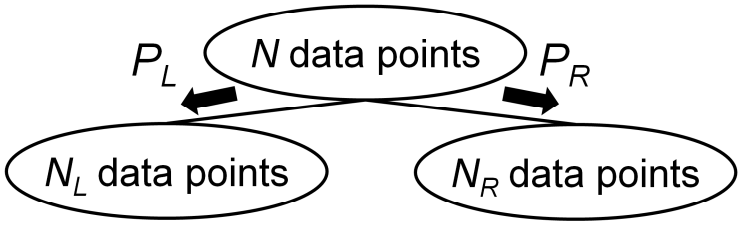
K measures how much the split deviates from what we would get if the data where random

K small means increase in IG of split is not significant

In this example (primes are expected):

$K = (10/9 - 1)^2/(10/9) + (35/9 - 4)^2/(35/9) + \ldots = 0.0321$

χ^2 Criterion: General Case

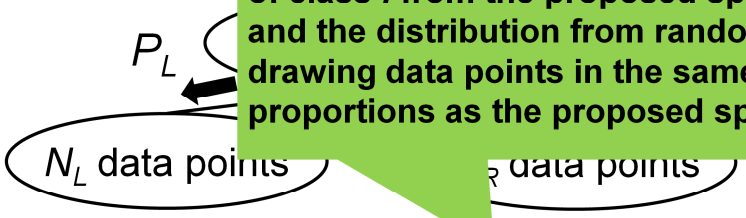


$$K = \sum_{\substack{\text{all classes } i \\ \text{children } j}} \frac{(N_{ij} - N'_{ij})^2}{N'_{ij}}$$

- N_{ij} = Number of points from class *i* in child *j*
- N'_{ij} = Number of points from class *i* in child *j assuming a random selection*
- $N'_{ij} = N_i \times P_j$

χ^2 Criteri

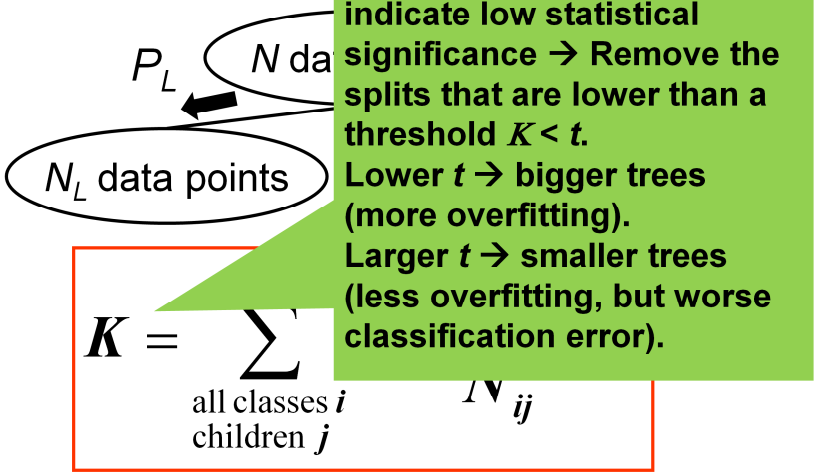
Difference between the distribution of class *i* from the proposed split and the distribution from randomly drawing data points in the same proportions as the proposed split



$$K = \sum_{\substack{\text{all classes } i \\ \text{children } j}} \frac{(N_{ij} - N'_{ij})^2}{N'_{ij}}$$

- N_{ij} = Number of points from class *i* in child *j*
- N'_{ij} = Number of points from class *i* in child *j assuming a random selection*
- $N'_{ij} = N_i \times P_j$

χ^2 Criterion:



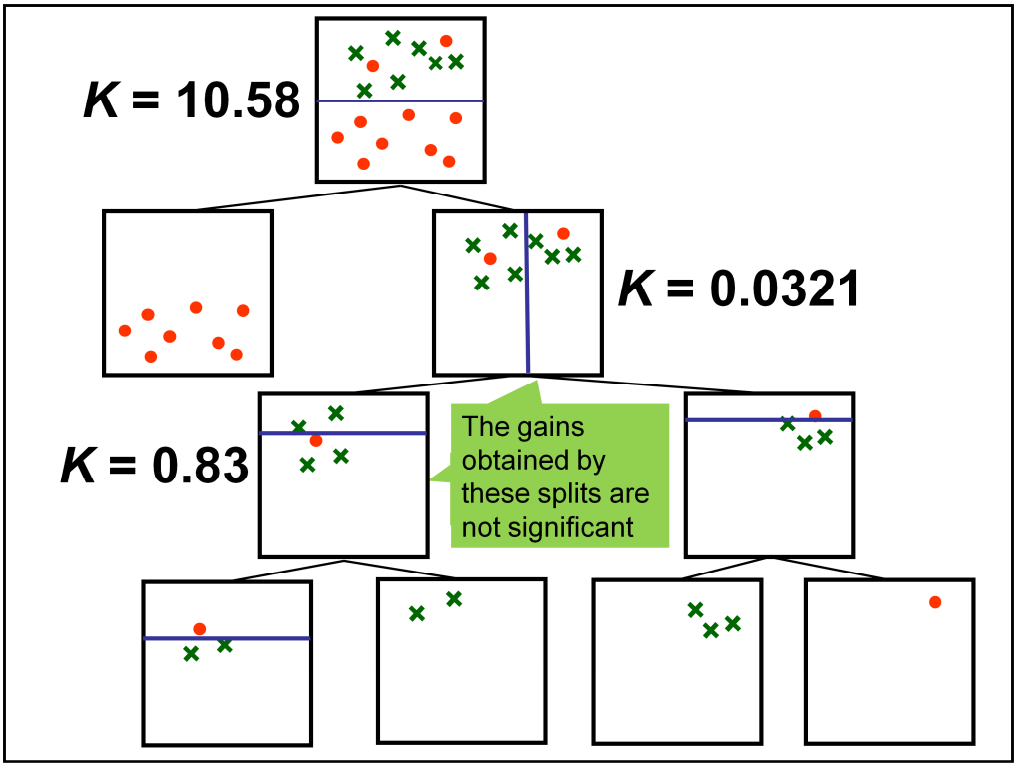
Small (Chi-square) values indicate low statistical significance → Remove the splits that are lower than a threshold $K < t$.
Lower t → bigger trees (more overfitting).
Larger t → smaller trees (less overfitting, but worse classification error).

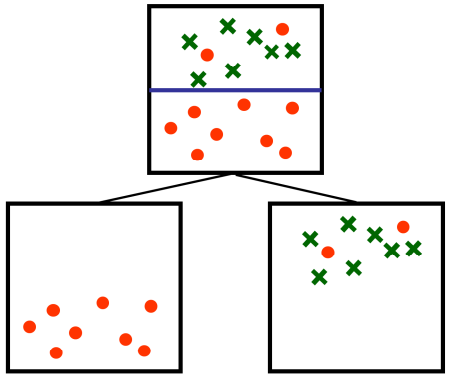
- **N_{ij} = Number of points from class i in child j**
- **N'_{ij} = Number of points from class i in child j *assuming a random selection***
- **$N'_{ij} = N_i \times P_j$**

You need to choose a threshold, which is hacky.

Decision Tree Pruning

- Construct the entire tree as before
- Starting at the leaves, recursively eliminate splits:
  - At a leaf $\mathcal{N}$:
    - Compute the *K* value for $\mathcal{N}$ and its parent $\mathcal{P}$.
    - If the *K* values is lower than the threshold *t*:
      - Eliminate all of the children of $\mathcal{P}$
      - $\mathcal{P}$ becomes a leaf
  - Repeat until no more splits can be eliminated


K = 10.58
K = 0.0321
K = 0.83
The gains obtained by these splits are not significant



- **By thresholding *K* we end up with the decision tree that we would expect (i.e., one that does not overfit the data)**
- **Note: The approach is presented with continuous attributes in this example but it works just as well with discrete attributes**

When reporting accuracy, you'll still want to use the test set.

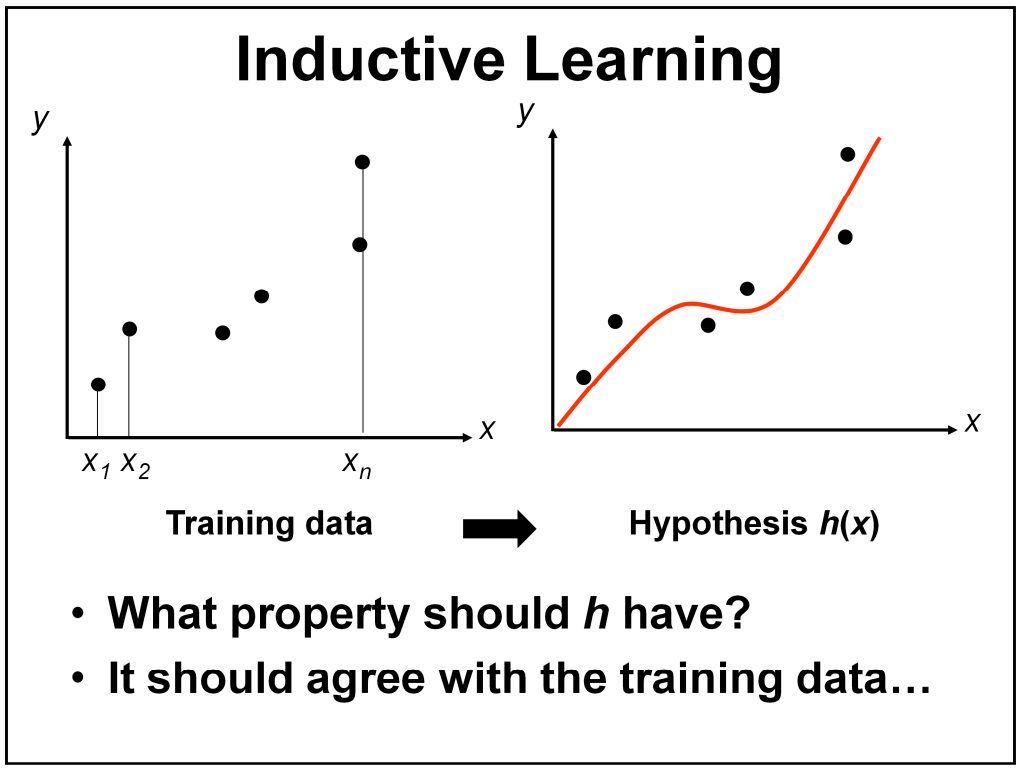
χ^2 Pruning

- The test on *K* is a version of a standard statistical test, the χ^2 ('chi-square') test.
- The value of *t* is retrieved from statistical tables. For example, *K* > *t* means that, with confidence 95%, the information gain due to the split is significant.
- If *K* < *t*, with high confidence, the information gain will be 0 over very large training samples
  - Reduces *overfitting*
  - Eliminates *irrelevant attributes*

Note: Inductive Learning

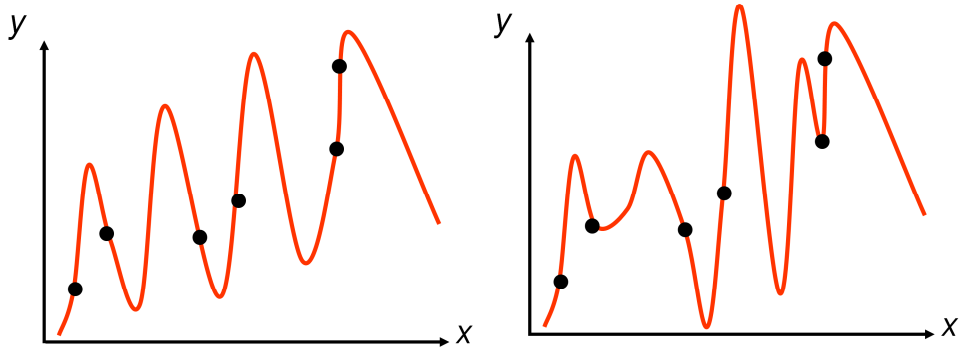
- **The decision tree approach is one example of an inductive learning technique:**
- **Suppose that data *x* is related to output *y* by a unknown function $y = f(x)$**
- **Suppose that we have observed training examples $\{(x_1,y_1),..,(x_n,y_n)\}$**
- ***Inductive learning problem*: Recover a function *h* (the "hypothesis") such that $h(x) \approx f(x)$**
- ***y* = *h*(*x*) predicts *y* from the input data *x***
- ***The challenge*: The hypothesis space (the space of all hypothesis *h* of a given form; for example the space of all of the possible decision trees for a set of *M* attributes) is huge + many different hypotheses may agree with the training data.**

Now here's a big problem. How do we decide what "features" to use?



For example, polynomial fitting. We could use linear regression, or quadratic, or …

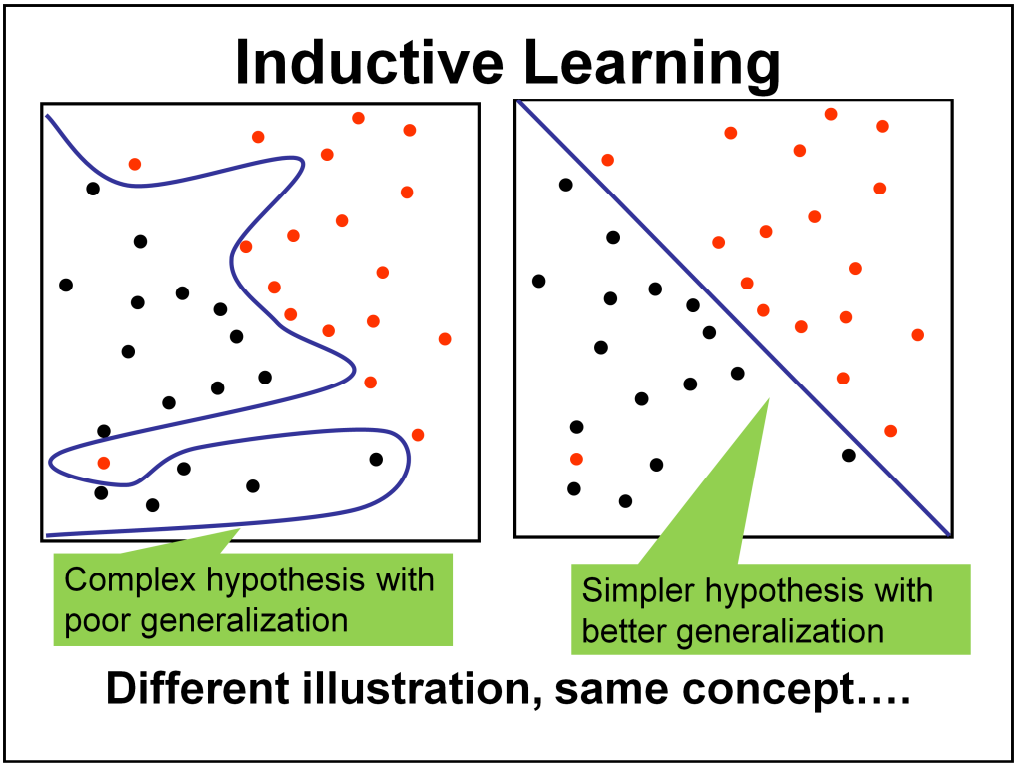
Inductive Learning



Two stupid hypotheses that fit the training data perfectly

- **What property should *h* have?**
- **It should agree with the training data…**
- **But that can lead to arbitrarily complex hypotheses and there are many of them; which one should we choose?…**

OR THIS. AWESOME.



Of course, there's a tradeoff between accuracy and simplicity. Note that on the right we have a lower accuracy. Chances are we can say that other point is an error, or noise, but it's not always that simple. Sometimes there's a big gap in accuracy for an only-slightly-simpler model. How do we decide how complicated we're willing to make it? For more on this, read the wikipedia article on regularization:

http://en.wikipedia.org/wiki/Regularization_(mathematics)