Learning from examples [Decision Trees] (Chapter 19)

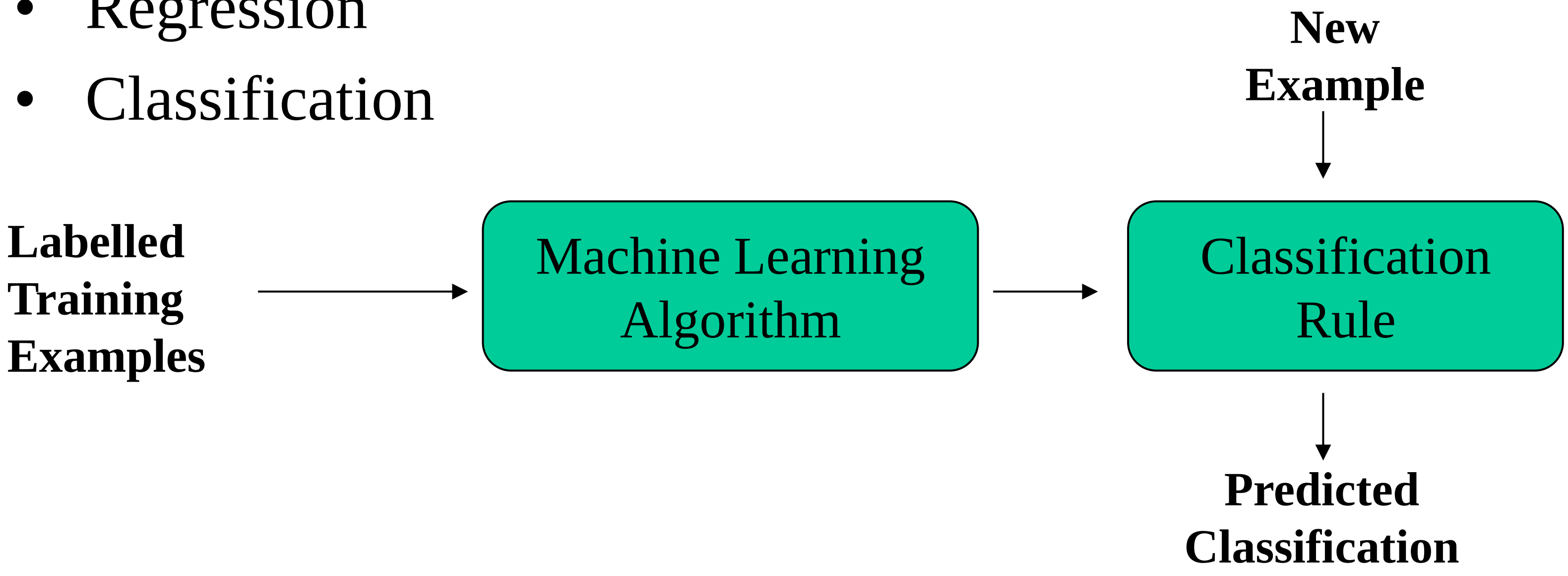
- Introduction to Machine Learning
- Learning Agents
- Inductive Learning
- Decision Tree Learning
- Performance Measurement

Machine Learning

- Studies how to use past observations to automatically learn to make accurate predictions
- Learning is NOT learning by heart
- Any computer could do this, the difficulty is to generalise a behaviour to a novel situation

Types of Problem

- Regression
- Classification



Applications

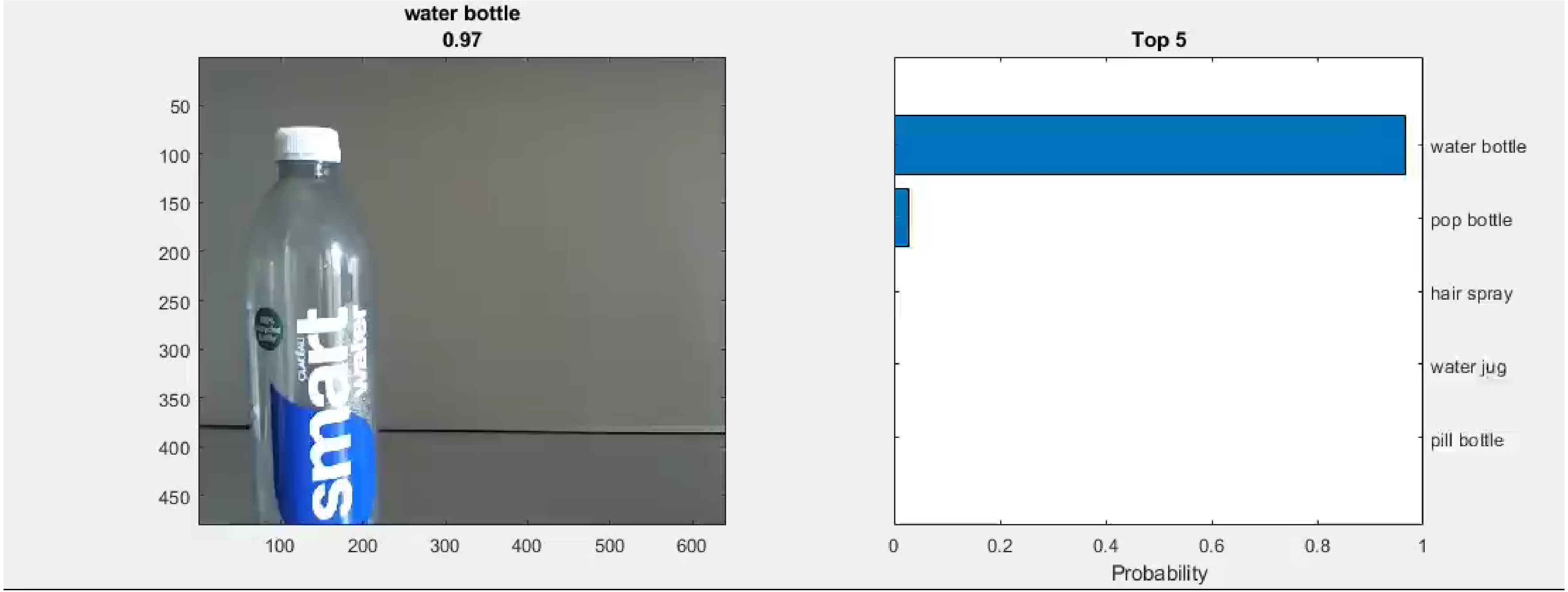
- **Computer Vision**
  - Face Detection / verification
  - Handwriting recognition

- **Speech Processing**
  - Word/Sentence/Person/Emotion Recognition

- **Others**
  - Finance: asset prediction
  - Telecom: Traffic prediction
  - Data Mining
  - Games
  - Control

Applications

GoogleNet object classifier



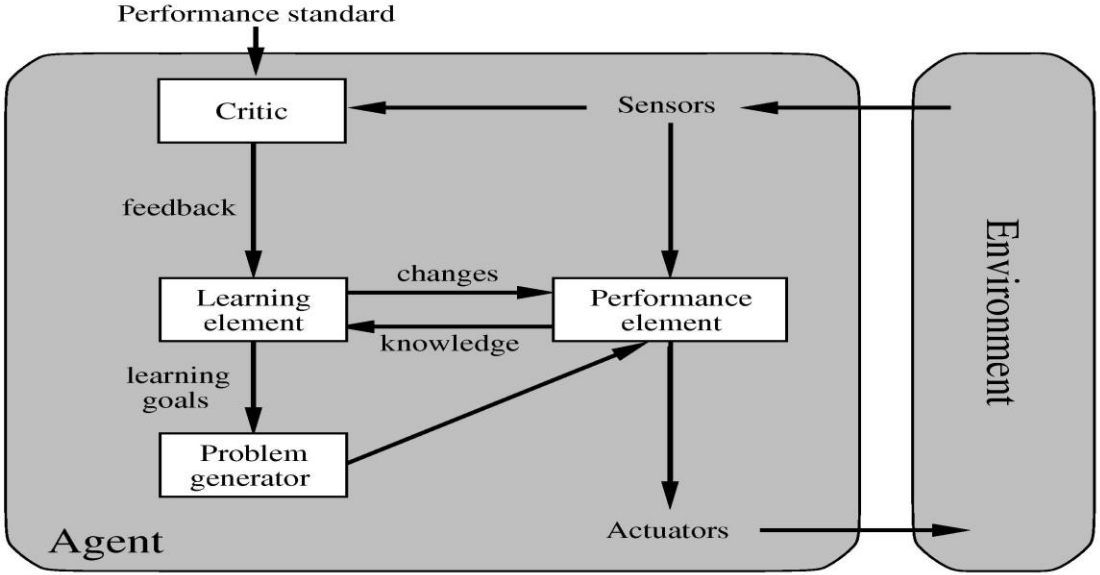
Learning In Agents

- Learning is essential for unknown environments

- Learning is useful as a system construction method
  - Expose agent to reality rather than trying to write it down

- Learning modifies the agent's decision mechanism to improve performance

Learning agents



Forms of Learning

- Design of the learning element is affected by three major issues:
  - Components of performance element to be learnt
  - Feedback available
  - Representation used
- Supervised learning
  - correct answers for each instance
- Unsupervised learning
  - no specific output values are supplied
- Reinforcement learning: occasional rewards

Training a Mario-playing RL Agent

https://pytorch.org/tutorials/intermediate/mario_rl_tutorial.html

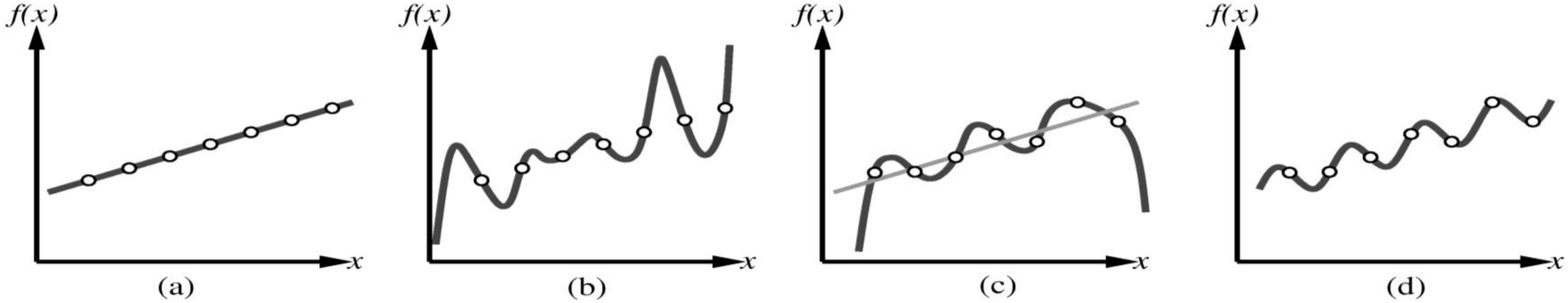
Inductive Learning

- Simplest form: Learn a function from examples
- Problem:

  For target function f

  Find a hypothesis h such that h≈f

  Given a training set of examples



Test your intuition!

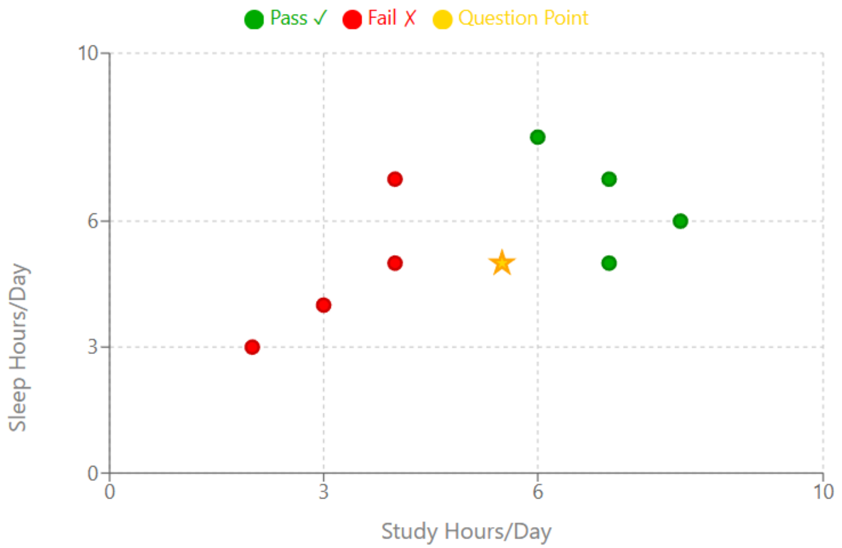
Student ID	Study Hours/Day	Sleep Hours/Day	Exam Result
1	8	6	Pass
2	7	7	Pass
3	6	8	Pass
4	3	4	Fail
5	4	5	Fail
6	2	3	Fail
7	7	5	Pass
8	4	7	Fail
9	5.5	5	?

Join at:
vevox.app

ID:
181-264-559

Will the 9th student pass or fail?

Test your intuition!

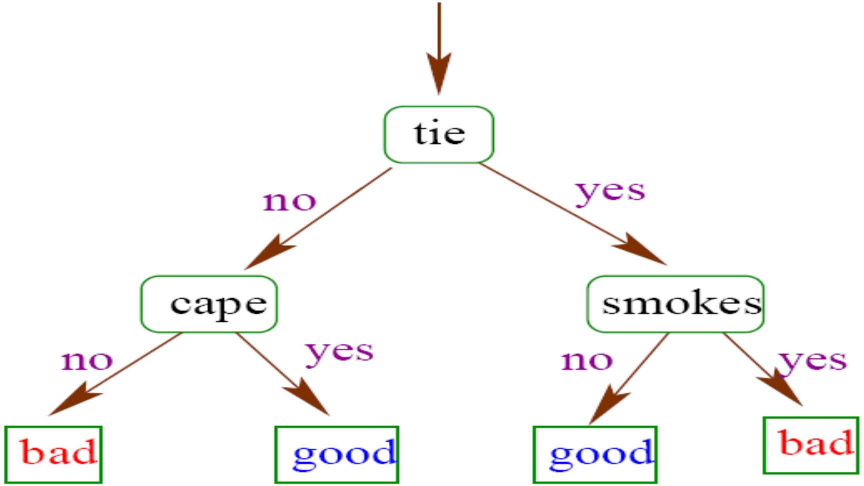


Good versus Evil

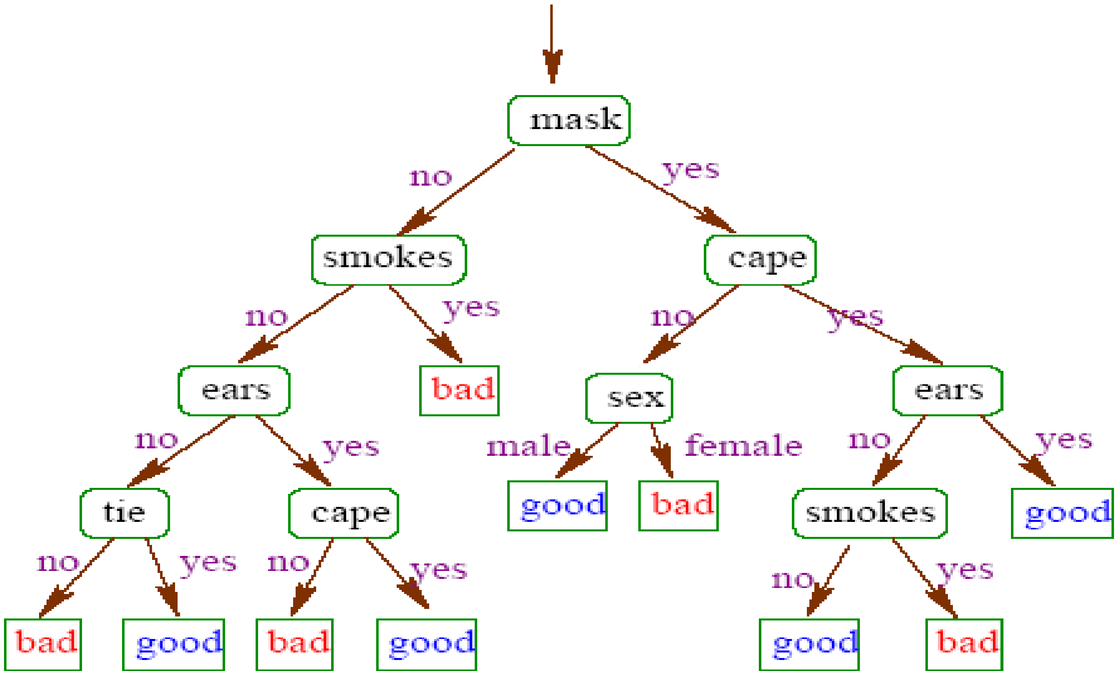
	Gender	mask	cape	tie	ears	smokes	class
batman	M	Y	Y	N	Y	N	G
robin	M	Y	Y	N	N	N	G
alfred	M	N	N	Y	N	N	G
penguin	M	N	N	Y	N	Y	B
catwoman	F	Y	N	N	Y	N	B
joker	M	N	N	N	N	N	B
batgirl	F	Y	Y	N	Y	N	??
riddler	M	Y	N	N	N	N	??

www.cs.princeton.edu/~schapire

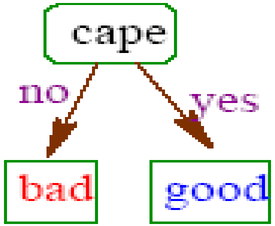
An example classifier



Overly complex classifier



Too Simple



Decision Trees

- Input: set of attributes of object or situation
- Output: Decision – predicted output value for the input
- Inputs and outputs may have discrete or continuous values

- **Classification learning**
  - Learning a Discrete valued function
- **Regression learning**
  - Learning a Continuous valued function

Learning decision trees

Problem: decide whether to wait for a table at a restaurant, based on the following attributes:

1. Alternate: is there an alternative restaurant nearby?
2. Bar: is there a comfortable bar area to wait in?
3. Fri/Sat: is today Friday or Saturday?
4. Hungry: are we hungry?
5. Patrons: number of people in the restaurant (None, Some, Full)
6. Price: price range ($, $$, $$$)
7. Raining: is it raining outside?
8. Reservation: have we made a reservation?
9. Type: kind of restaurant (French, Italian, Thai, Burger)
10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)

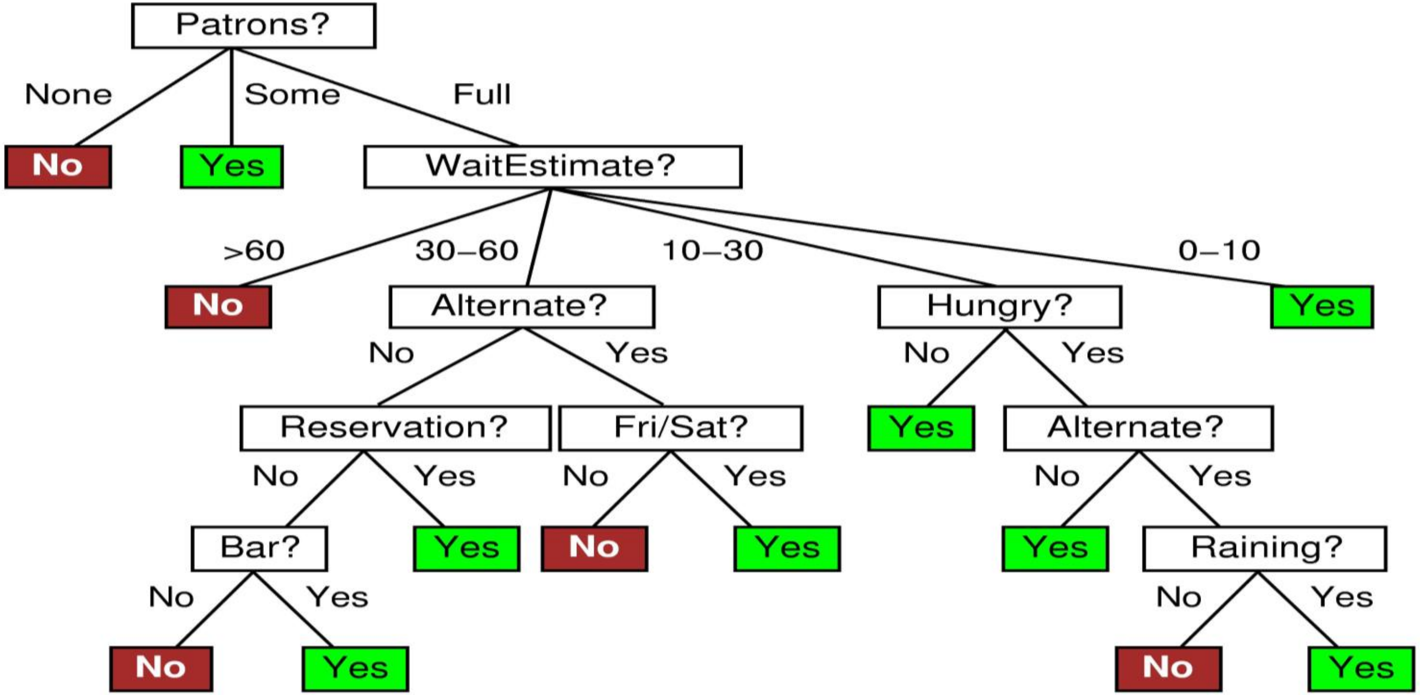
Attribute-based representations

- Examples described by attribute values (Boolean, discrete, continuous)
- E.g., situations where I will/won't wait for a table:

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

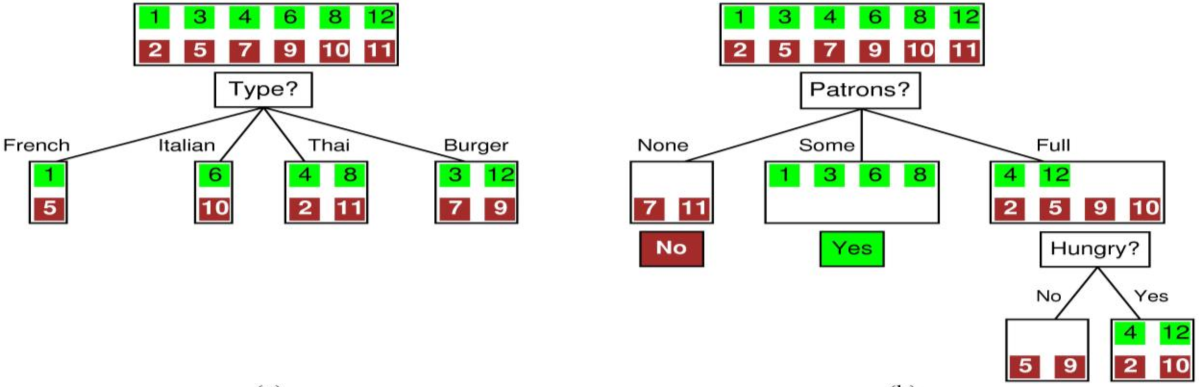
- Classification of examples is positive (T) or negative (F)

Example



Patrons?
None
Some
Full
No
Yes
WaitEstimate?
>60
30–60
10–30
0–10
No
Alternate?
Hungry?
Yes
No
Yes
No
Yes
Reservation?
Fri/Sat?
Yes
Alternate?
No
Yes
No
Yes
No
Yes
Bar?
Yes
No
Yes
Yes
Raining?
No
Yes
No
Yes
No
Yes
No
Yes

Splitting examples based on attributes



(a) (b)

Join at:
vevox.app

ID:
181-264-559

Which tree is more useful?

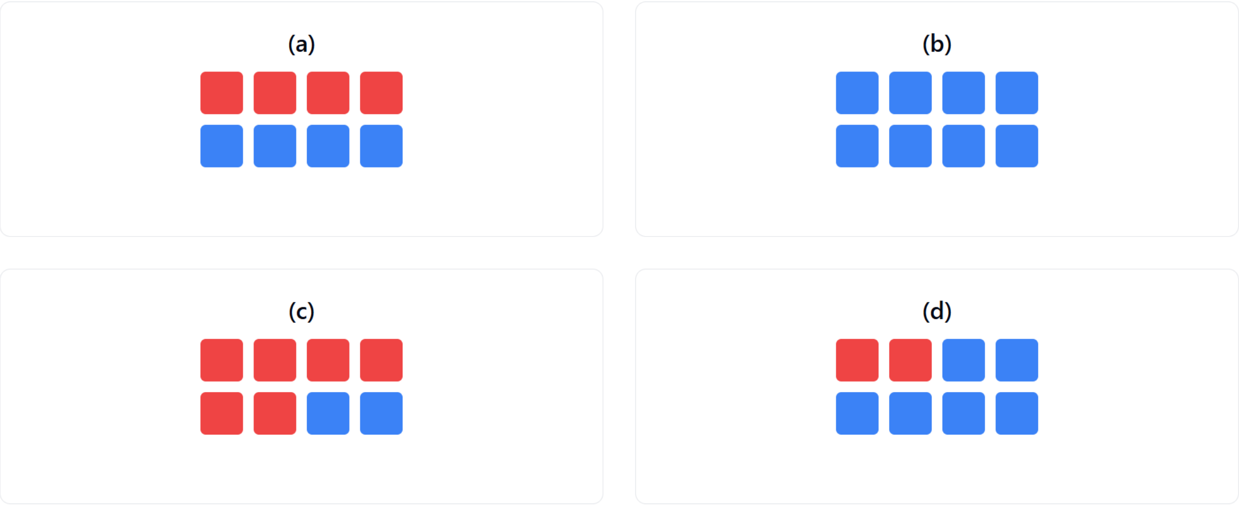
Information Gain

- We want to determine which attribute in a given set of training feature vectors is most useful for discriminating between the classes to be learned.
- Information gain tells us how important a given attribute of the feature vectors is.
- We will use it to decide the ordering of attributes in the nodes of a decision tree.

Information Gain

- **Entropy**: Entropy is a measure of impurity or disorder in a set of data.
- **Information Gain**: Information gain measures the effectiveness of an attribute in classifying the data. It quantifies the reduction in entropy.

Which group has the highest entropy?

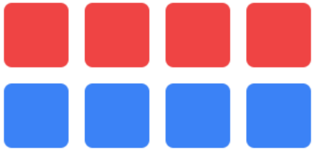


Join at:
vevox.app

ID:
181-264-559

Which group has the highest entropy?

(a)



Entropy: 1.000 bits

(b)

Entropy: 0.000 bits

(c)

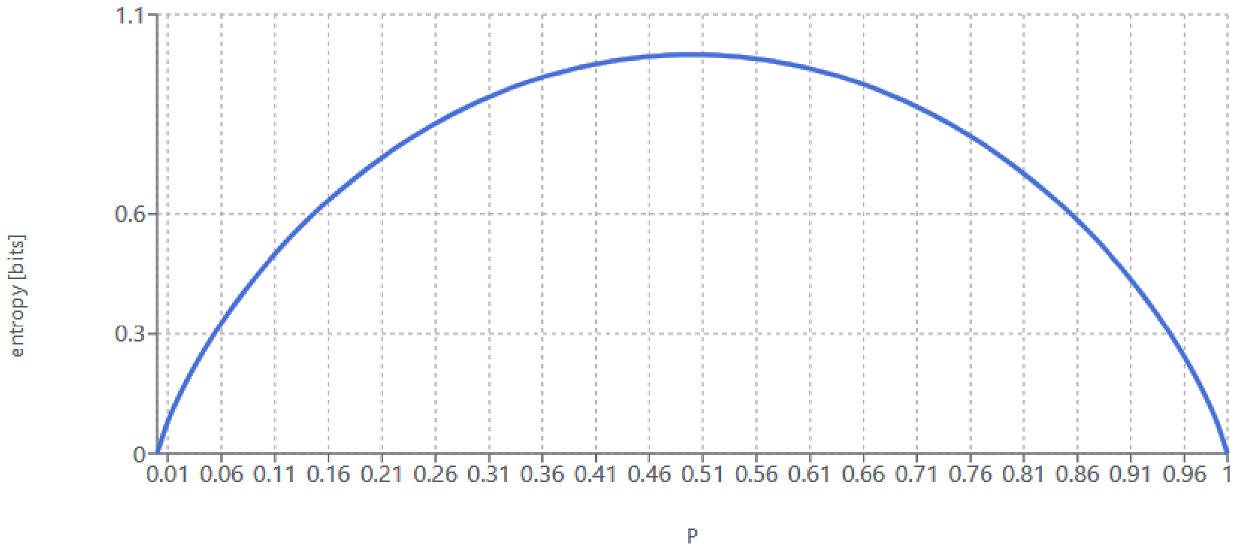
Entropy: 0.811 bits

(d)

Entropy: 0.811 bits

Entropy

Binary Entropy Function



- Maximum entropy (1 bit) occurs at P = 0.5
- Entropy approaches 0 as P approaches 0 or 1
- $H(P) = -P \log_2(P) - (1-P) \log_2(1-P)$

Entropy

- Choose attributes based on the expected amount of **information** they provide (Shannon & Weaver (1949))
- Entropy in an answer when prior is <$P(v_1)$,$P(v_n)$> is

$$\mathrm{H}(P(v_1), \ldots, P(v_n)) = \sum_{i=1}^{n} -P(v_i) \log_2 P(v_i)$$

- Scale: 1 bit = entropy of a Boolean distribution with probability <0.5, 0.5>
- Maximum entropy occurs with uniform distribution (equal probabilities)
- Minimum entropy occurs when one probability is 1 (complete certainty)

Information Gain

- For p positive and n negative examples at a node, the entropy is: $$H\left(\frac{p}{p+n}, \frac{n}{p+n}\right) \text{ bits}$$

  For 12 restaurant examples p=n=6, therefore 1 bit.

  Each attribute splits the examples into subsets E_i, each of which needs less information to complete the classification, or in other words have less entropy.

Information Gain

- The information gain from the attribute A is given by

$$\text{Gain }(A) = H\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right) \text{-Remainder (A)}$$

- Remainder(A) is the expected number of bits per example over all branches of A (Average Entropy of children)

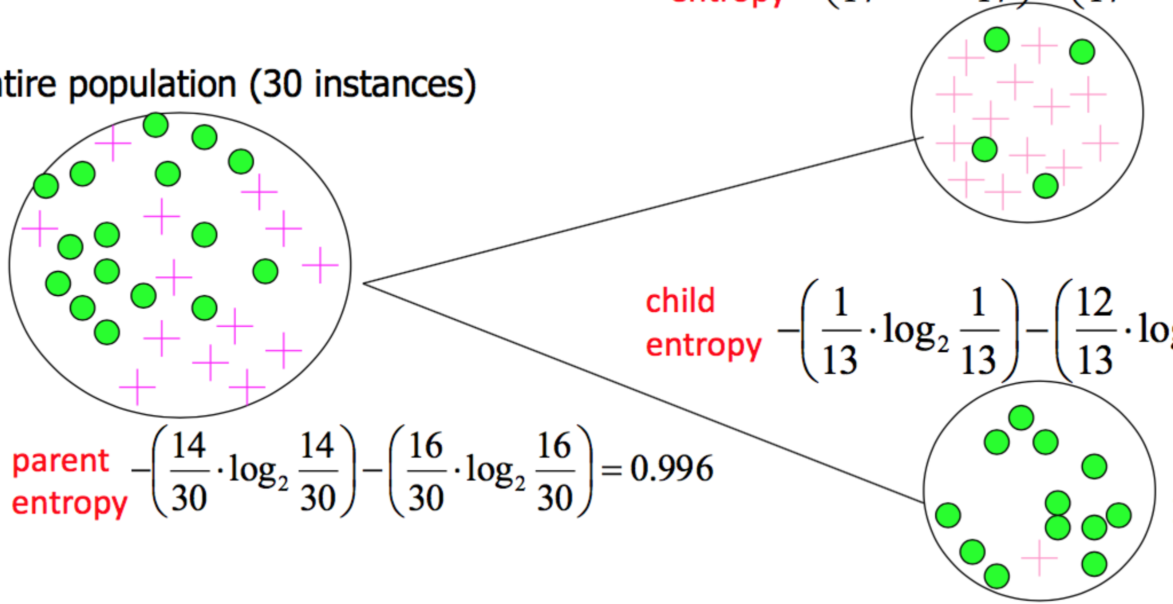
$$\text{Remainder}(A) = \sum_i \frac{p_i+n_i}{p+n} H\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right)$$

Calculating Information Gain

Information Gain = entropy(parent) – [average entropy(children)]

child entropy $-\left(\frac{13}{17}\cdot\log_2\frac{13}{17}\right)-\left(\frac{4}{17}\cdot\log_2\frac{4}{17}\right)=0.787$

Entire population (30 instances)



17 instances

child entropy $-\left(\frac{1}{13}\cdot\log_2\frac{1}{13}\right)-\left(\frac{12}{13}\cdot\log_2\frac{12}{13}\right)=0.391$

13 instances

parent entropy $-\left(\frac{14}{30}\cdot\log_2\frac{14}{30}\right)-\left(\frac{16}{30}\cdot\log_2\frac{16}{30}\right)=0.996$

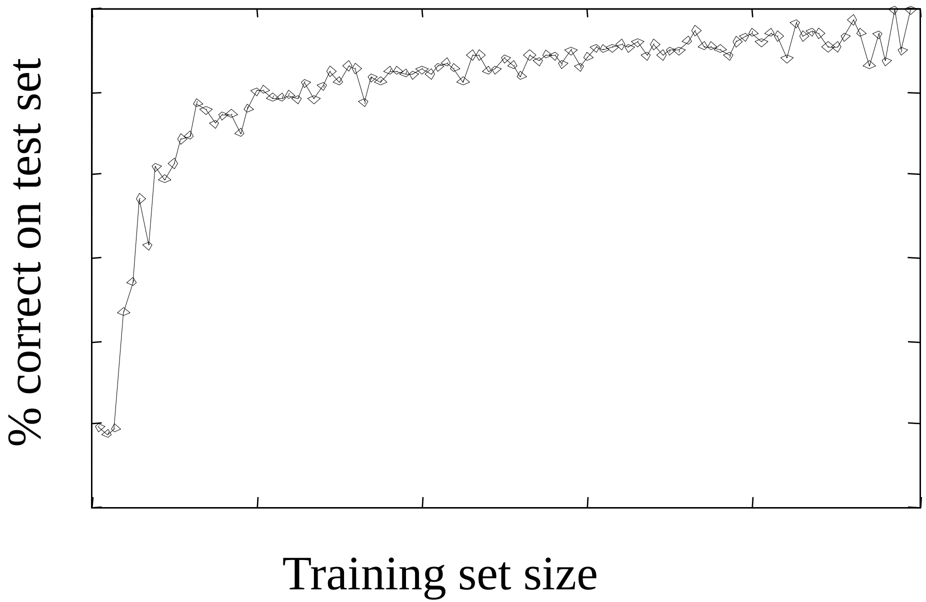
(Weighted) Average Entropy of Children = $\left(\frac{17}{30}\cdot 0.787\right)+\left(\frac{13}{30}\cdot 0.391\right)=0.615$

Performance Measurement

How do we know that we have an accurate classifier ?

1. Collect large set of training data
2. Divide into two disjoint sets: training and test sets
3. Apply the learning algorithm to the training set, generating a hypothesis (h)
4. Measure the percentage of examples in the test set that are correctly classified by h.
5. Repeat steps 1-4 for different sizes of training set

Performance Measurement



Good Performance

- Need:
  - Enough training examples
  - Good performance on the training / test set
  - Classifier/Model that is not too complex
    - Complexity is measured by:
      - Number of bits needed to write it down
      - Number of parameters

Ockham’s Razor (principle of parsimony)

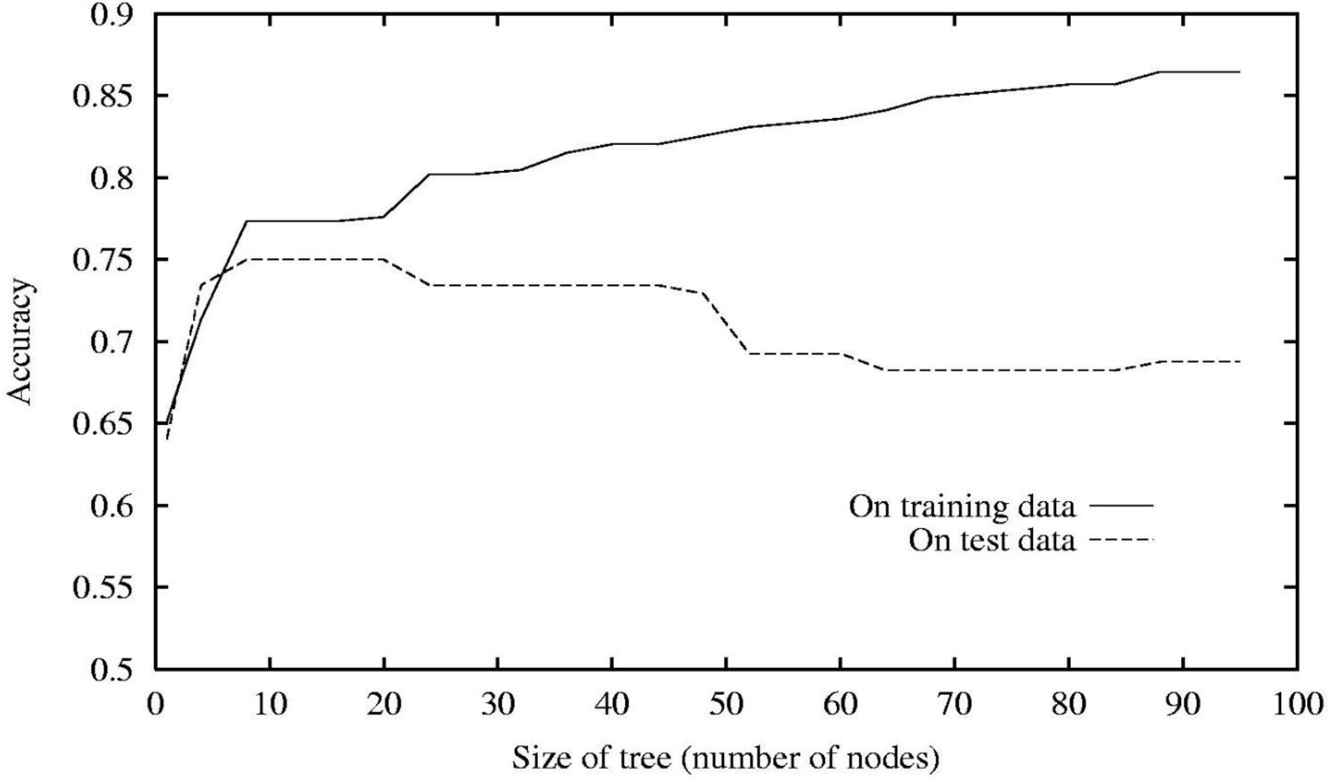
- Principle stated by William of Ockham (1285-1347)
- **Idea**: The simplest consistent explanation is the best
- Therefore, the smallest decision tree that correctly classifies all of the training examples is best
- Finding the provably smallest decision tree is NP-hard
    - So instead of constructing the absolute smallest tree consistent with the training examples, construct one that is pretty small

Overfitting in Decision Trees

- Many kinds of “noise” can occur in the examples:
  - Two examples have same attribute/value pairs, but different classifications
  - Some values of attributes are incorrect because of errors in the data acquisition process or the preprocessing phase
  - The instance was labeled incorrectly (+ instead of -)

- Also, some attributes are irrelevant to the decision-making process
  - e.g., color of a dice is irrelevant to its outcome

Overfitting in Decision Trees



Based on Slide by Pedro Domingos

Avoiding Overfitting in Decision Trees

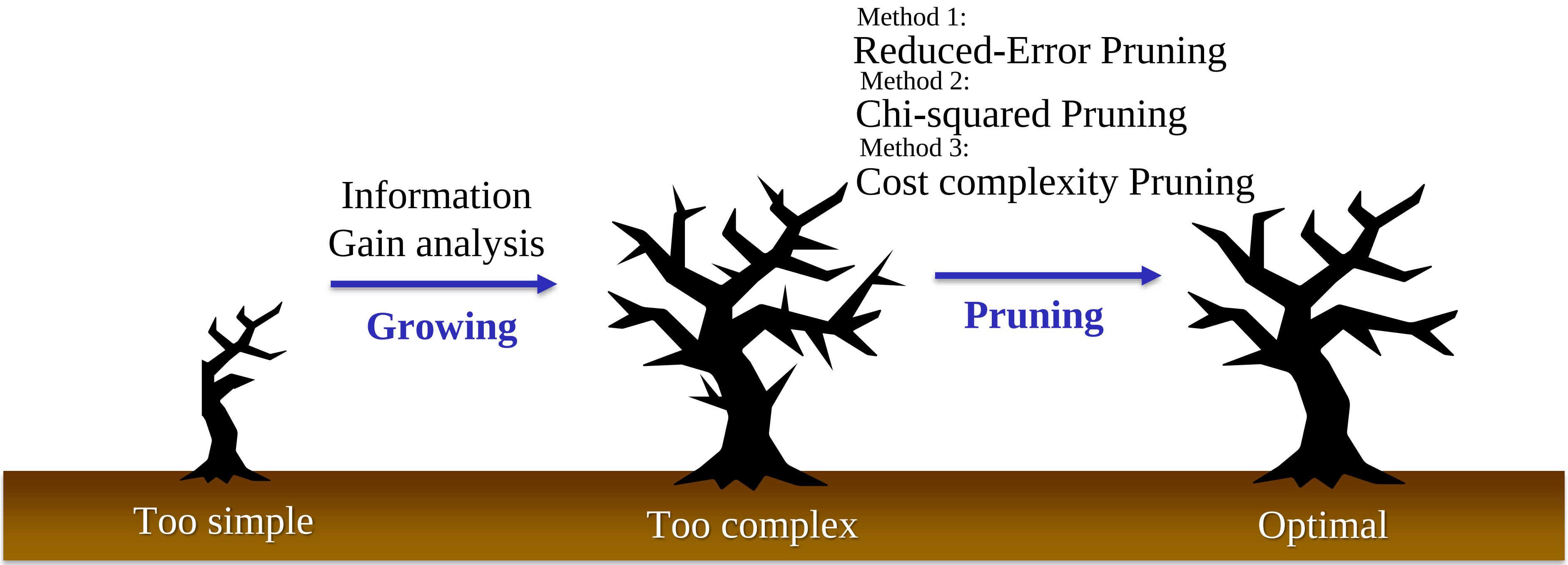
How can we avoid overfitting?

- Stop growing when data split is not statistically significant
- Acquire more training data
- Remove irrelevant attributes (manual process – not always possible)
- **Grow full tree, then post-prune**

How to select "best" tree:

- Measure performance over training data
- Measure performance over separate validation data set
- Add complexity penalty to performance measure (heuristic: simpler is better)

Growing a full tree and then pruning it

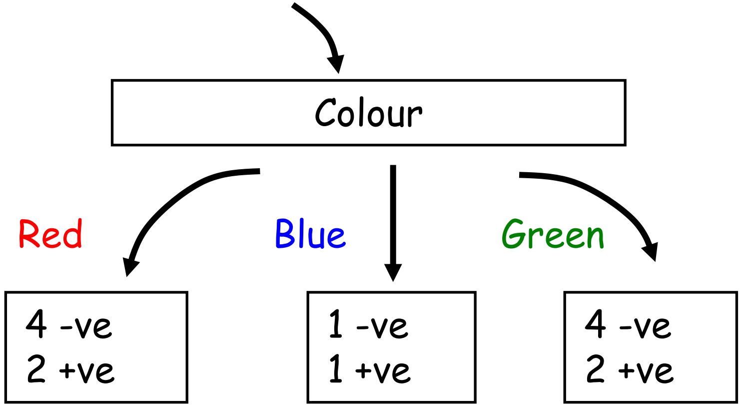


Method 1: Reduced-Error Pruning

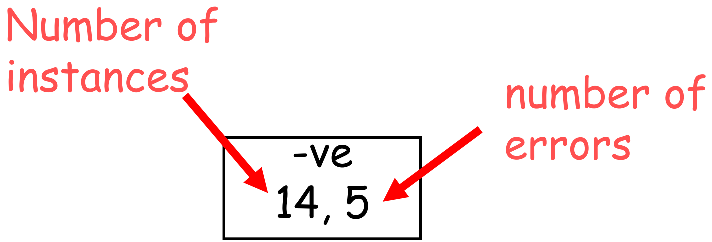
- Split training data further into *training* and *validation* sets
  Grow tree based on *training set*
- Prune the tree until further pruning is harmful:
  - Evaluate impact on validation set if pruning each possible node (plus those below it).
  - Greedily remove the node that as the result the accuracy on *validation set* improves the most.

Reduced-Error Pruning

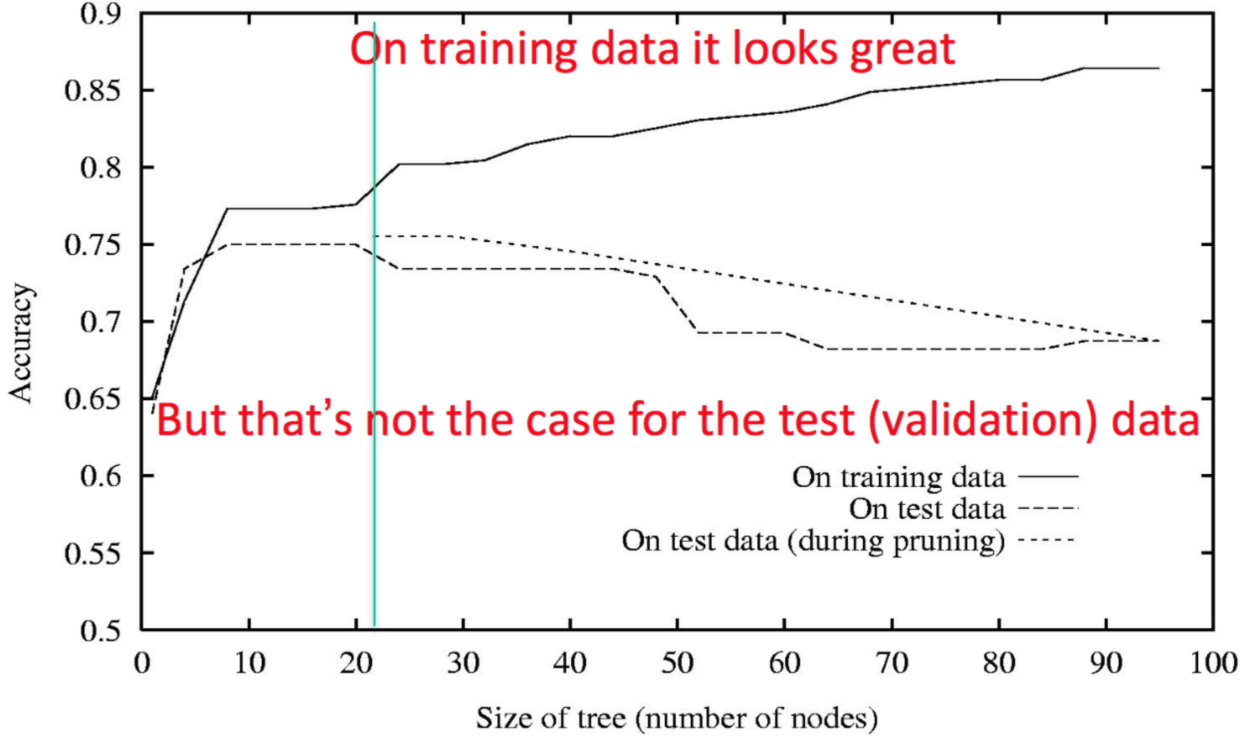
- Pruning of the decision tree is done by replacing a whole subtree by a leaf node.
- The replacement takes place if a decision rule establishes that the expected error rate in the subtree is greater than in the single leaf.
- Consider a classification problem where we're trying to predict if an object belongs to the positive (+ve) or negative (-ve) class. Our goal is to minimize classification errors.
- If we had simply predicted the majority class (-ve), we make 5 errors instead of 9 in the worst case.



We replace the subtree with a leaf:



Effect of Reduced-Error Pruning



Based on Slide by Pedro Domingos

Method 2: Chi-Squared Pruning

- Chi-squared (χ^2) test helps us determine if a split in our decision tree is meaningful or just due to random chance.
- It does that by comparing the observations and expectations
- It also considers the degree of freedom incorporating number of tree branches and number of classes

$$\chi^2 = \frac{(O-E)^2}{E}$$

O: Observation

E: Expectation

Chi-squared: separation criterion

(also called Δ)

Method 2: Chi-Squared Pruning

Full form:

$$\Delta = \chi^2 = \sum_{i=1}^{c} \sum_{j=1}^{r} \frac{(N_{ij} - N'_{ij})^2}{N'_{ij}}$$

Where:

- c = number of classes
- r = number of child branches (children nodes)
- N_{ij} = Observed number of instances of class i in child node j
- N'_{ij} = Expected number of instances of class i in child node j assuming random distribution
- N'_{ij} is calculated as: $N'_{ij} = N_i \times P_j$ where:
  - N_i = total number of instances of class i in the parent node
  - P_j = proportion of instances going to child j (N_j / N_{total})

Chi-Squared Pruning

$$\chi^2 = \frac{(O-E)^2}{E}$$

O: Observation
E: Expectation
Chi-squared: separation criterion
(also called Δ)

$$df = (r-1) \times (c-1)$$

r: is the number of branches
c: is the number of classes
df or k: is the degree of freedom

Significance Level

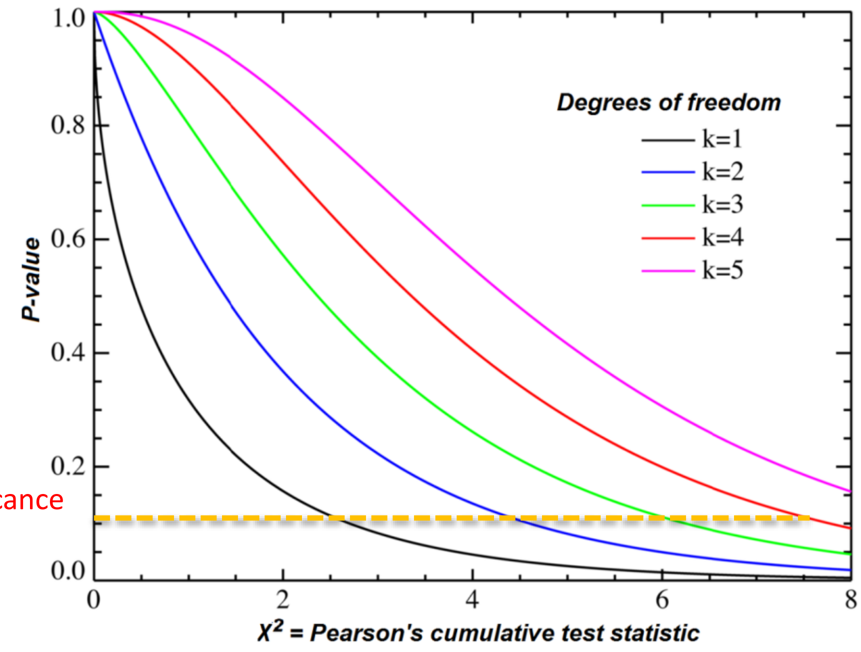
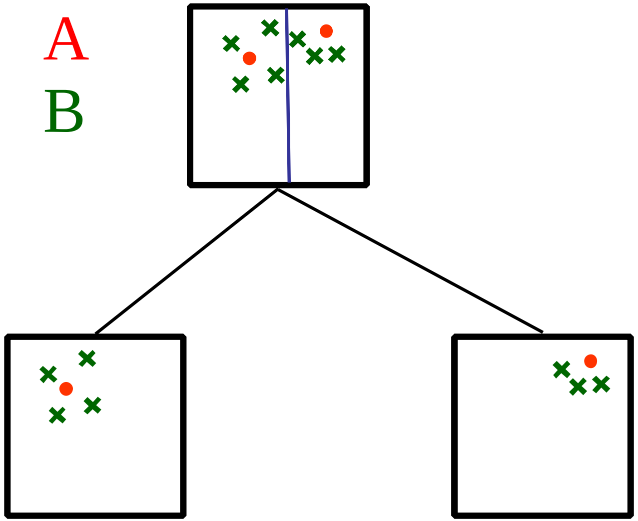


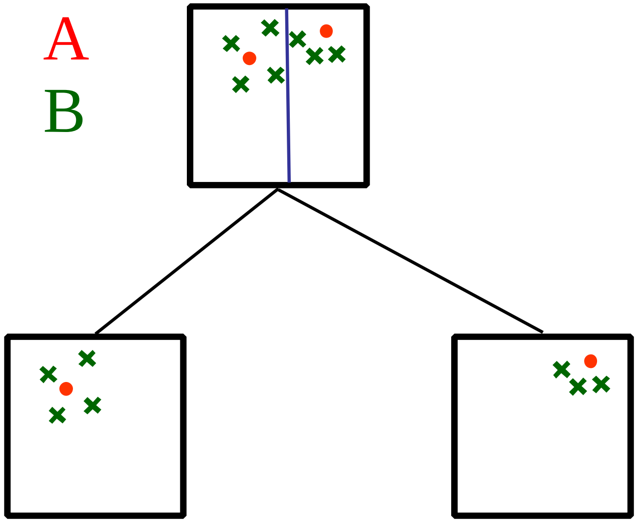
Chart from Wikipedia

Chi-Squared Pruning



- The number of class A in the root node is N_A =2
- The number of class B in the root node is N_B =7
- The number of class A in the left node is N_{AL}=1
- The number of class B in the left node is N_{BL}=4

Chi-Squared Pruning



The proportion of the data going to the left node is

$$p_L = (N_{AL} + N_{BL})/(N_A + N_B) = 5/9$$

Suppose now that the data is *completely randomly* distributed

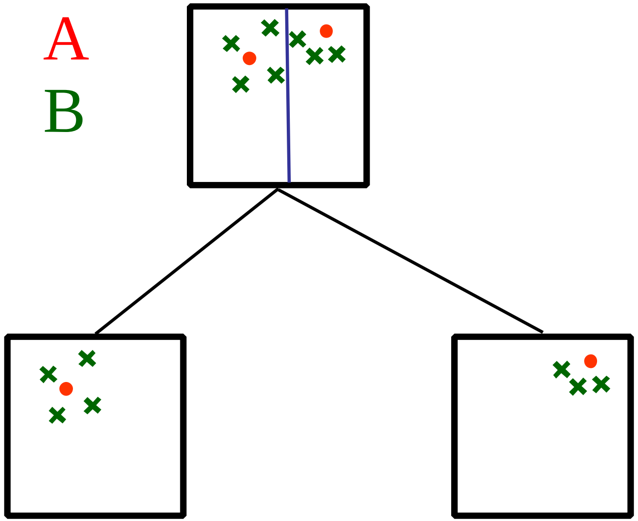
The expected number of class A in the left node given random splitting:

$N'_{AL} = N_A . p_L = 2x5/9 \quad \approx 1$

The expected number of class B in the left node given random splitting:

$N'_{BL} = N_B . p_L = 35/9 \quad \approx 3.5$

Chi-Squared Pruning



Measure of statistical significance:

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

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

Δ small: The increase in IG of the split is not significant

In this example

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

Chi-Squared Pruning

- Construct the entire tree as before
- Starting at the leaves, recursively eliminate splits:
  - At a leaf N:
    - Compute the Δ value for Node and its parent P.
    - If the obtained *p value* based on the calculated Δ *is low* (<0.05)
      - Split is likely to be significant
    - Otherwise
      - Eliminate all of the children of P
      - P becomes a leaf
  - Repeat until no more splits can be eliminated

Chi-Squared Pruning

- Delta (Δ) is also called chi-squared (χ^2) statistic, calculated as the sum of squared differences between observed and expected frequencies, divided by expected frequencies. This statistic measures whether the split's distribution significantly differs from random chance
- The resulting p-value is obtained from the chi-squared distribution table or charts like the following.
- If p is small the information gain due to the split is significant.
  - Reduces *overfitting*
  - Eliminates *irrelevant attributes*

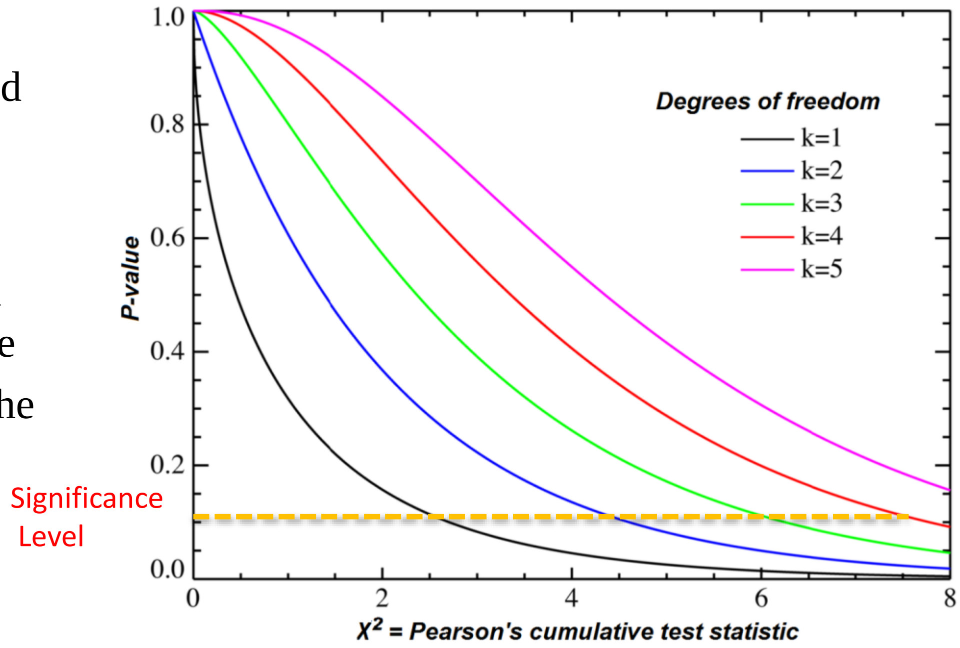


Chart from Wikipedia

Method 3: Cost Complexity Pruning

- Also known as "weakest link pruning"
- Introduced by Breiman et al. (1984)
- Controls tree size using a complexity parameter α
- Larger α = More pruning
- Smaller α = Less pruning
- Mathematical Form:
  - ✓ Cost = Training Error + $\alpha \times$ Tree Size
  - ✓ α balances accuracy vs complexity
  - ✓ $\alpha = 0$ gives unpruned tree
  - ✓ As α increases, subtrees are removed
  - ✓ Training Error = R(T) = Number of incorrect predictions / Total number

```
Root
├── Node1 (leaf)
├── Node2
│   ├── Node3 (leaf)
│   └── Node4 (leaf)
└── Node5 (leaf)
```

This tree has 4 leaves, so |T| = 4

Pruning approaches comparison

- Chi-squared Pruning
  - Local approach using statistical testing
  - Calculates Δ for each split
  - Compares observed vs expected distributions
  - Uses p-value to determine significance
  - Makes decisions node by node
  - More traditional statistical approach
  - Focused on split significance
- Cost Complexity Pruning
  - Global approach using parameter α
  - Minimizes: $R(T) + \alpha \times |T|$
  - $R(T)$: Training error
  - $|T|$: Number of leaves
  - α: Complexity parameter
  - Used in Python package scikit-learn (ccp_alpha)
  - Makes trade-off decisions across entire tree
  - Featured in coursework #2 tasks 8-12
- Reduced Error Pruning
  - Global approach using validation set
  - Uses separate validation dataset
  - Makes pruning decisions based on validation accuracy
  - Greedily removes subtrees that improve validation accuracy
  - Simple but requires extra data for validation
  - Makes decisions based on actual performance
  - Focused on error reduction