Statistical Learning

Philipp Koehn

10 November 2015

Outline

- Learning agents
- Inductive learning
- Decision tree learning
- Measuring learning performance
- Bayesian learning
- Maximum *a posteriori* and maximum likelihood learning
- Bayes net learning
  - ML parameter learning with complete data
  - linear regression

learning agents

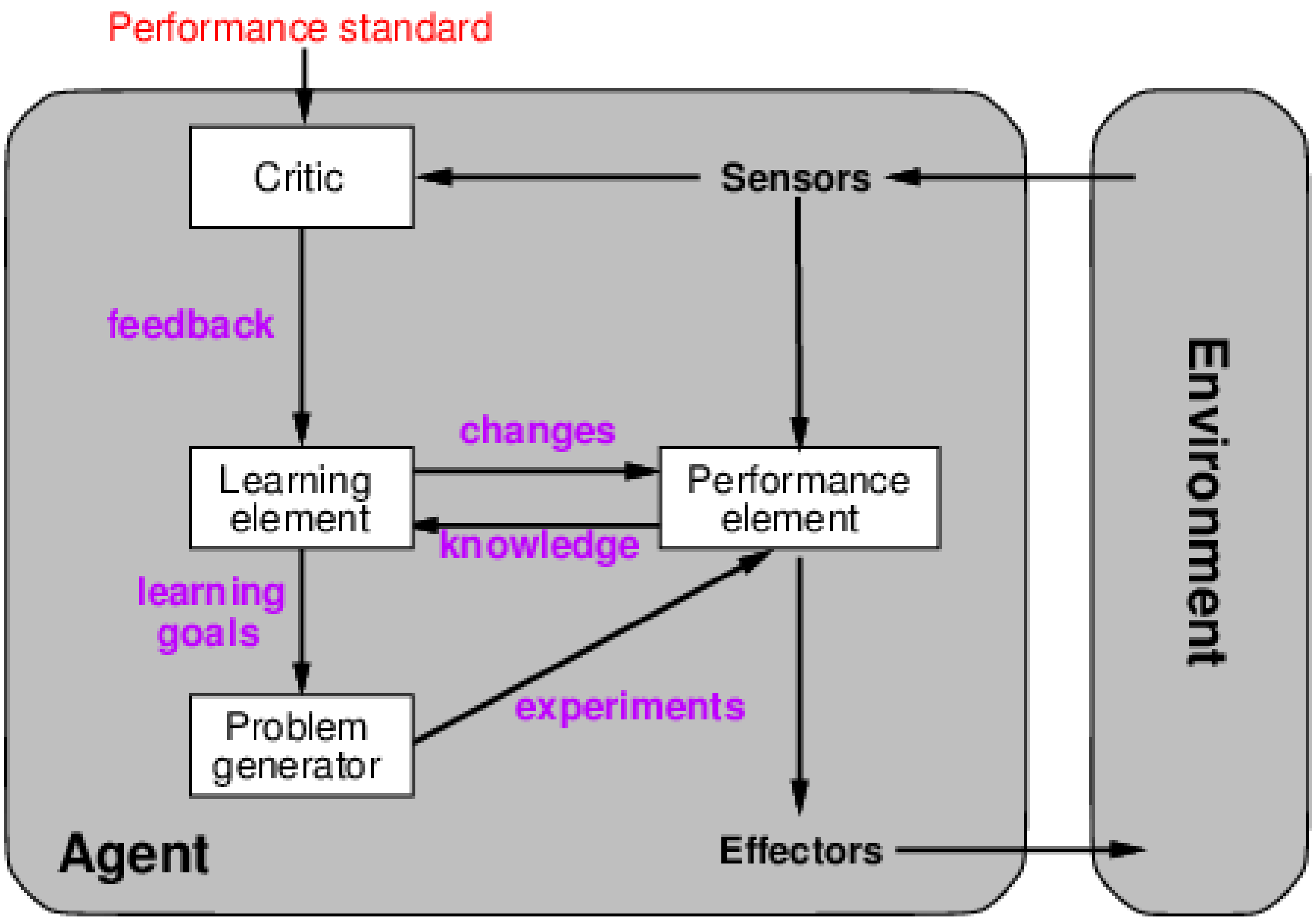
Learning

- Learning is essential for unknown environments,
  i.e., when designer lacks omniscience

- Learning is useful as a system construction method,
  i.e., expose the agent to reality rather than trying to write it down

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

Learning Agents



Learning Element

- Design of learning element is dictated by
  - what type of performance element is used
  - which functional component is to be learned
  - how that functional compoent is represented
  - what kind of feedback is available

- Example scenarios:

Performance element	Component	Representation	Feedback
Alpha–beta search	Eval. fn.	Weighted linear function	Win/loss
Logical agent	Transition model	Successor–state axioms	Outcome
Utility–based agent	Transition model	Dynamic Bayes net	Outcome
Simple reflex agent	Percept–action fn	Neural net	Correct action

Feedback

- Supervised learning
  - correct answer for each instance given
  - try to learn mapping $x \to f(x)$

- Reinforcement learning
  - occasional rewards, delayed rewards
  - still needs to learn utility of intermediate actions

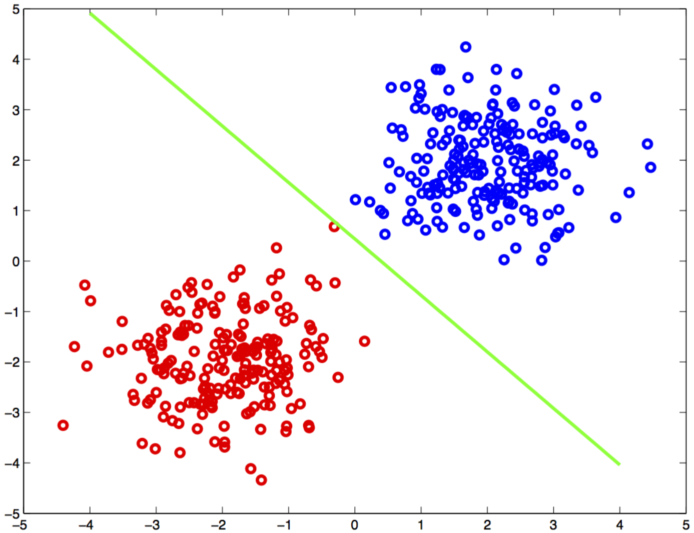
- Unsupervised learning
  - density estimation
  - learns distribution of data points, maybe clusters

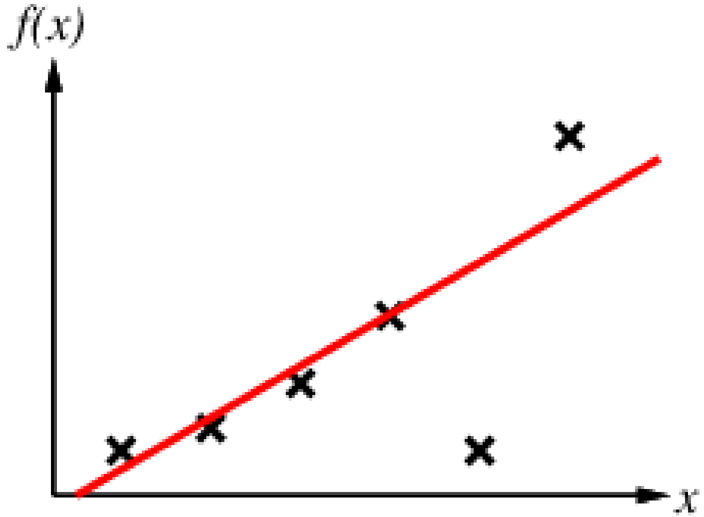
What are we Learning?

- Assignment to a class

  (maybe just binary yes/no decision)

  ⇒ Classification





- Real valued number

  ⇒ Regression

Inductive Learning

- Simplest form: learn a function from examples (**tabula rasa**)

- f is the target function

- An example is a pair x, $f(x)$, e.g.,

$$\begin{array}{c|c|c} O & O & X \\ \hline & X & \\ \hline X & & \end{array} \quad , \quad +1$$

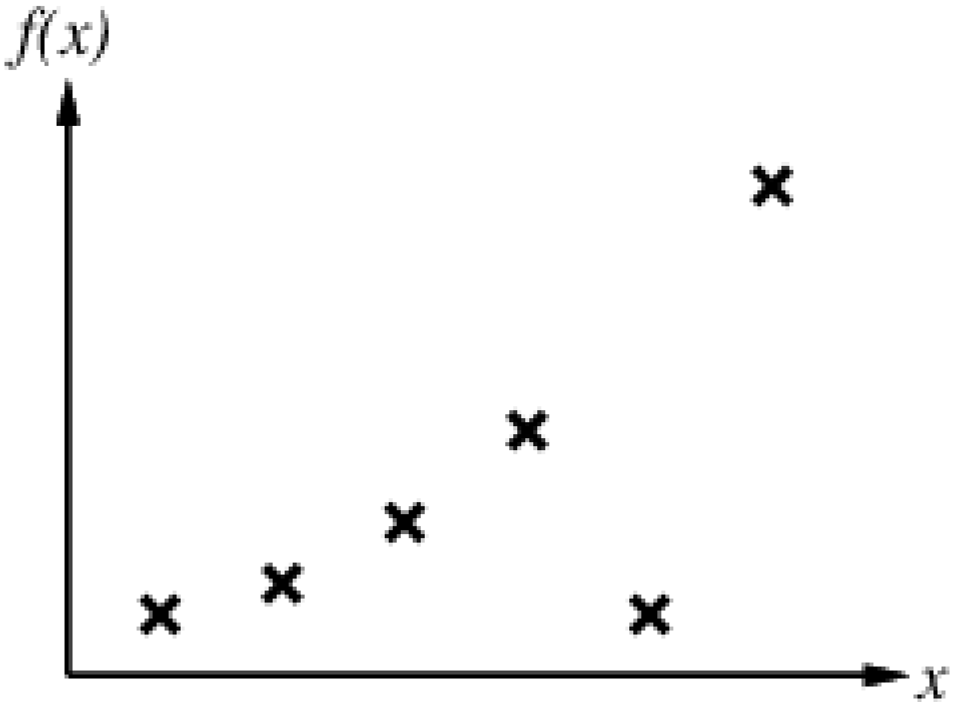
- Problem: find a(n) hypothesis h
  such that $h \approx f$
  given a training set of examples

- This is a highly simplified model of real learning
  - Ignores prior knowledge
  - Assumes a deterministic, observable "environment"
  - Assumes examples are **given**
  - Assumes that the agent **wants** to learn f

Inductive Learning Method

- Construct/adjust h to agree with f on training set
  (h is consistent if it agrees with f on all examples)

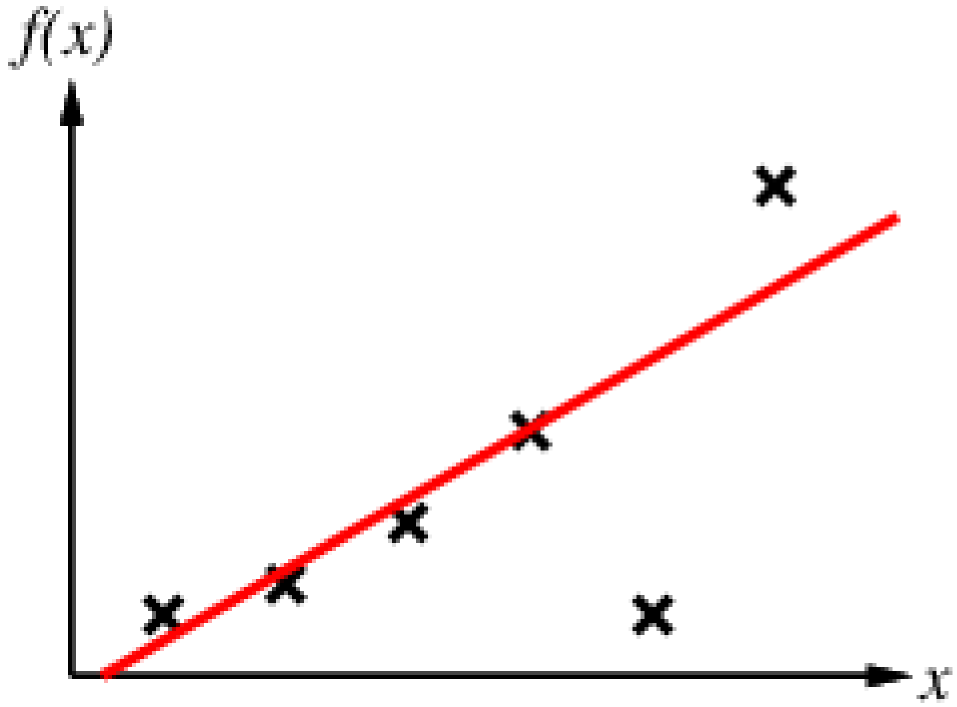
- E.g., curve fitting:



Inductive Learning Method

- Construct/adjust h to agree with f on training set
  (h is consistent if it agrees with f on all examples)

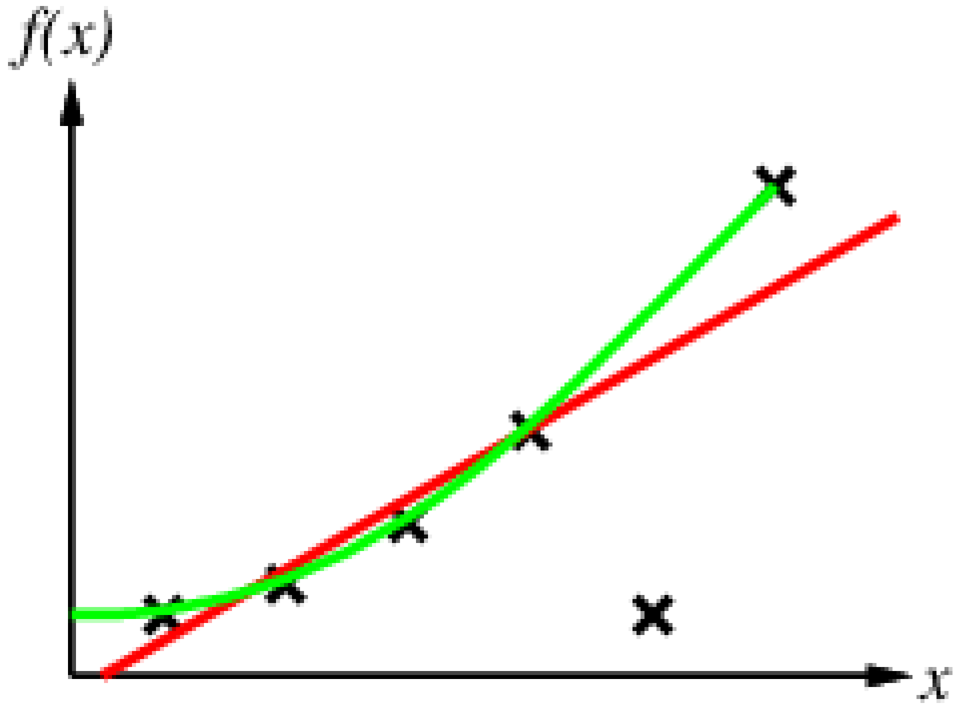
- E.g., curve fitting:



Inductive Learning Method

- Construct/adjust h to agree with f on training set
  (h is consistent if it agrees with f on all examples)

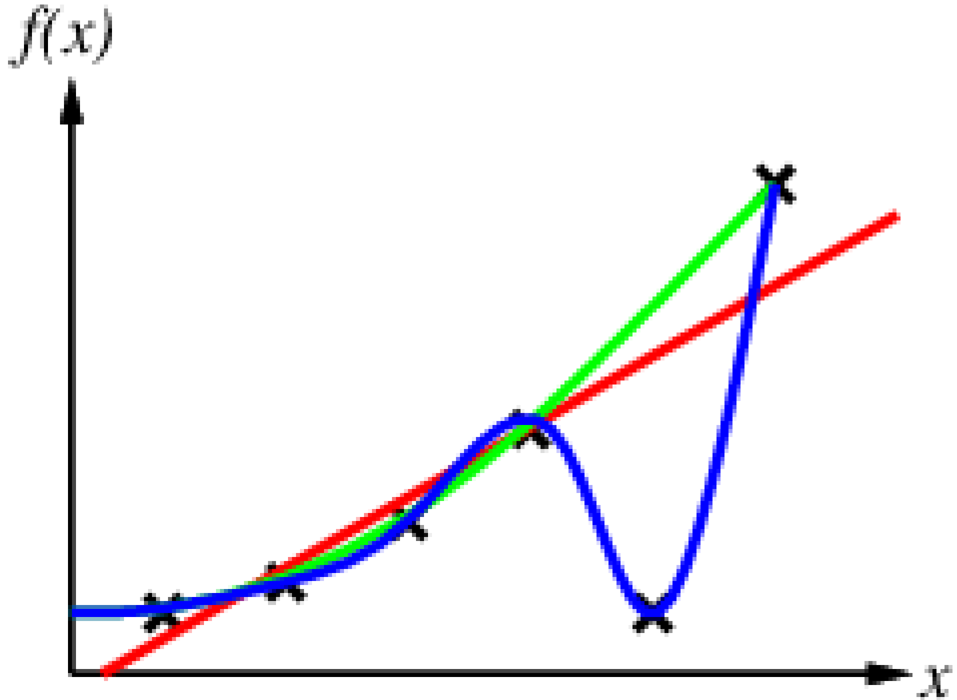
- E.g., curve fitting:



Inductive Learning Method

- Construct/adjust h to agree with f on training set
  (h is consistent if it agrees with f on all examples)

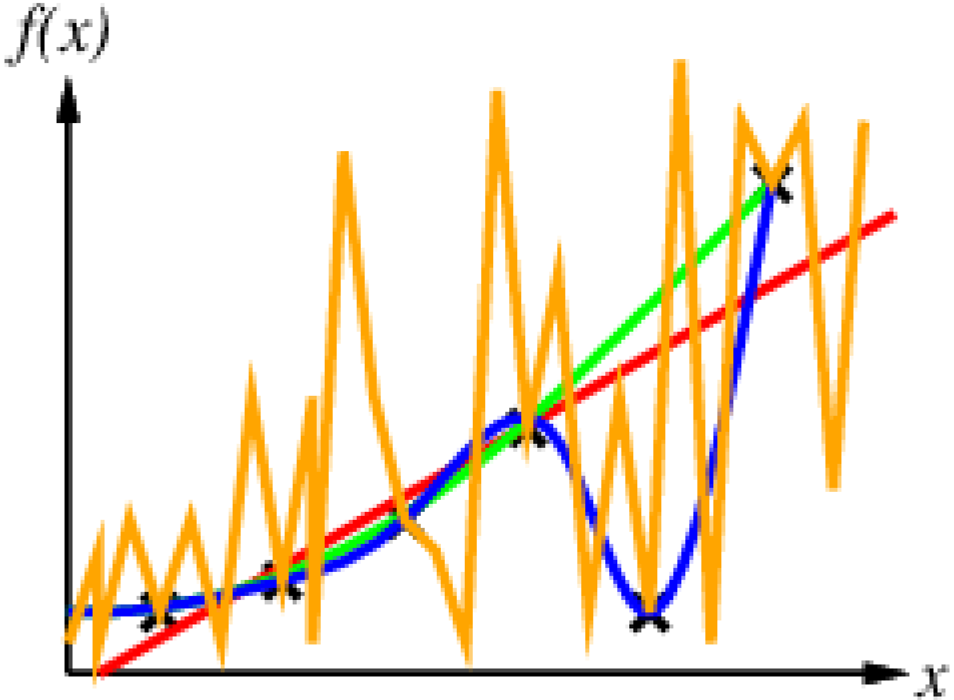
- E.g., curve fitting:



Inductive Learning Method

- Construct/adjust h to agree with f on training set
  (h is consistent if it agrees with f on all examples)

- E.g., curve fitting:



Ockham's razor: maximize a combination of consistency and simplicity

decision trees

Attribute-Based Representations

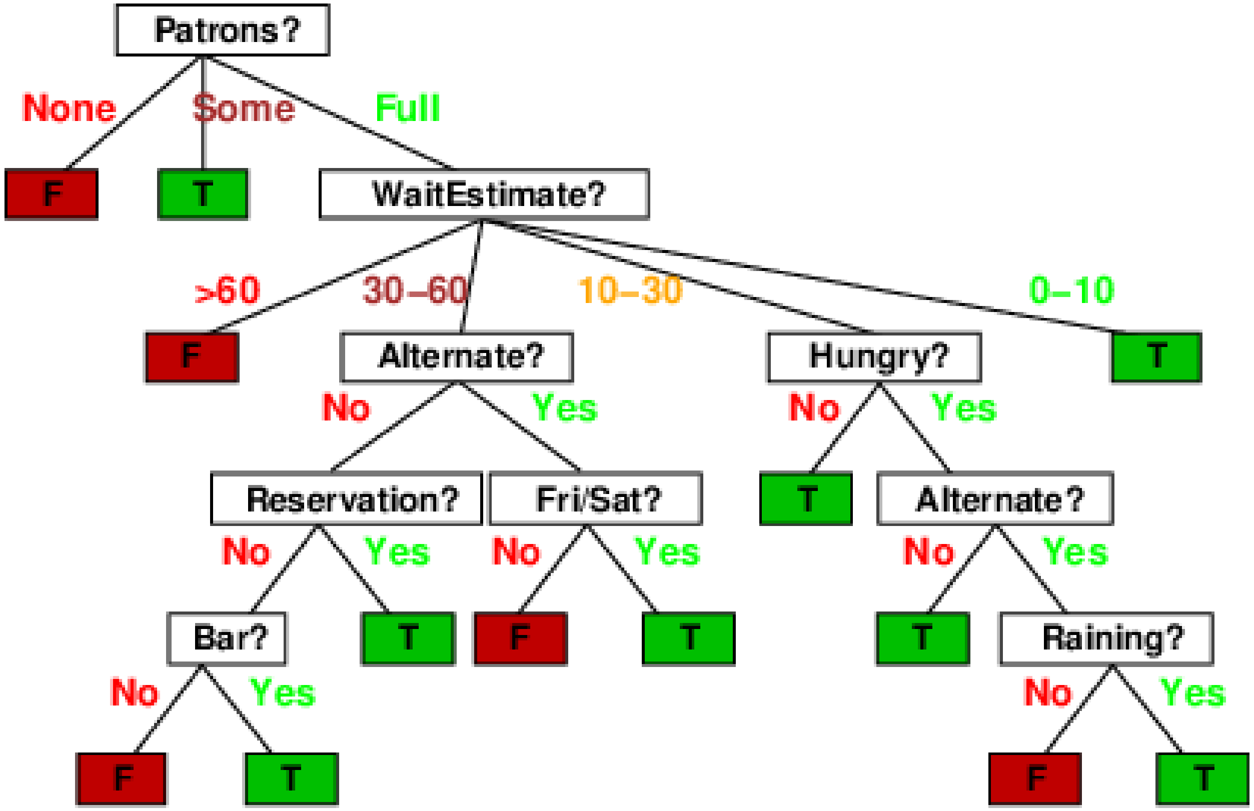
- Examples described by attribute values (Boolean, discrete, continuous, etc.)
- E.g., situations where I will/won't wait for a table:

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
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)

Decision Trees

- One possible representation for hypotheses
- E.g., here is the "true" tree for deciding whether to wait:

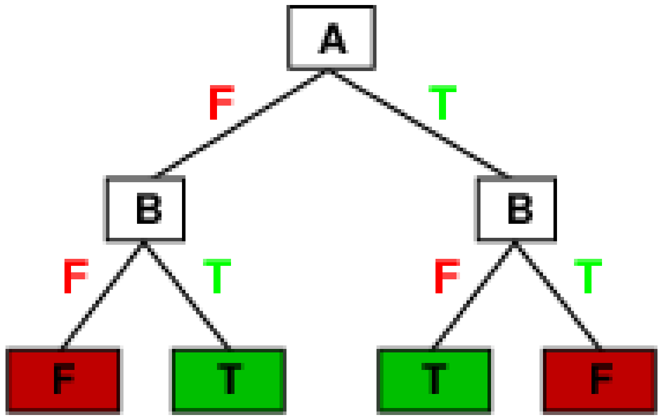


Expressiveness

- Decision trees can express any function of the input attributes.

- E.g., for Boolean functions, truth table row → path to leaf:

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



- Trivially, there is a consistent decision tree for any training set w/ one path to leaf for each example (unless f nondeterministic in x) but it probably won't generalize to new examples

- Prefer to find more **compact** decision trees

Hypothesis Spaces

- How many distinct decision trees with n Boolean attributes?

  = number of Boolean functions
  = number of distinct truth tables with 2^n rows = 2^{2^n}

- E.g., with 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees

- How many purely conjunctive hypotheses (e.g., $Hungry \wedge \neg Rain$)?

- Each attribute can be in (positive), in (negative), or out
  $\Longrightarrow$ 3^n distinct conjunctive hypotheses

- More expressive hypothesis space
  - increases chance that target function can be expressed ☺
  - increases number of hypotheses consistent w/ training set
    $\Longrightarrow$ may get worse predictions ☹

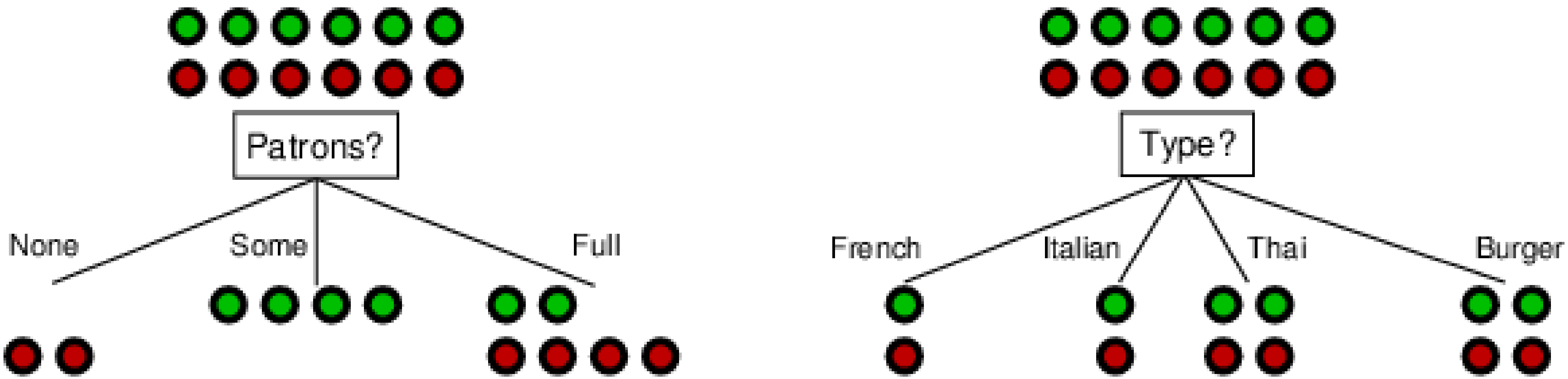
Decision Tree Learning

- Aim: find a small tree consistent with the training examples
- Idea: (recursively) choose "most significant" attribute as root of (sub)tree

```
function DTL(examples, attributes, default) returns a decision tree
  if examples is empty then return default
  else if all examples have the same classification then return the classification
  else if attributes is empty then return MODE(examples)
  else
      best ← CHOOSE-ATTRIBUTE(attributes, examples)
      tree ← a new decision tree with root test best
      for each value v_i of best do
          examples_i ← {elements of examples with best = v_i}
          subtree ← DTL(examples_i, attributes − best, MODE(examples))
          add a branch to tree with label v_i and subtree subtree
      return tree
```

Choosing an Attribute

- Idea: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"



- *Patrons?* is a better choice—gives **information** about the classification

Information

- Information answers questions
- The more clueless I am about the answer initially,
  the more information is contained in the answer
- Scale: 1 bit = answer to Boolean question with prior $\langle 0.5, 0.5 \rangle$
- Information in an answer when prior is $\langle P_1, \ldots, P_n \rangle$ is

$$H(\langle P_1, \ldots, P_n \rangle) = \sum_{i=1}^{n} -P_i \log_2 P_i$$

(also called entropy of the prior)

Information

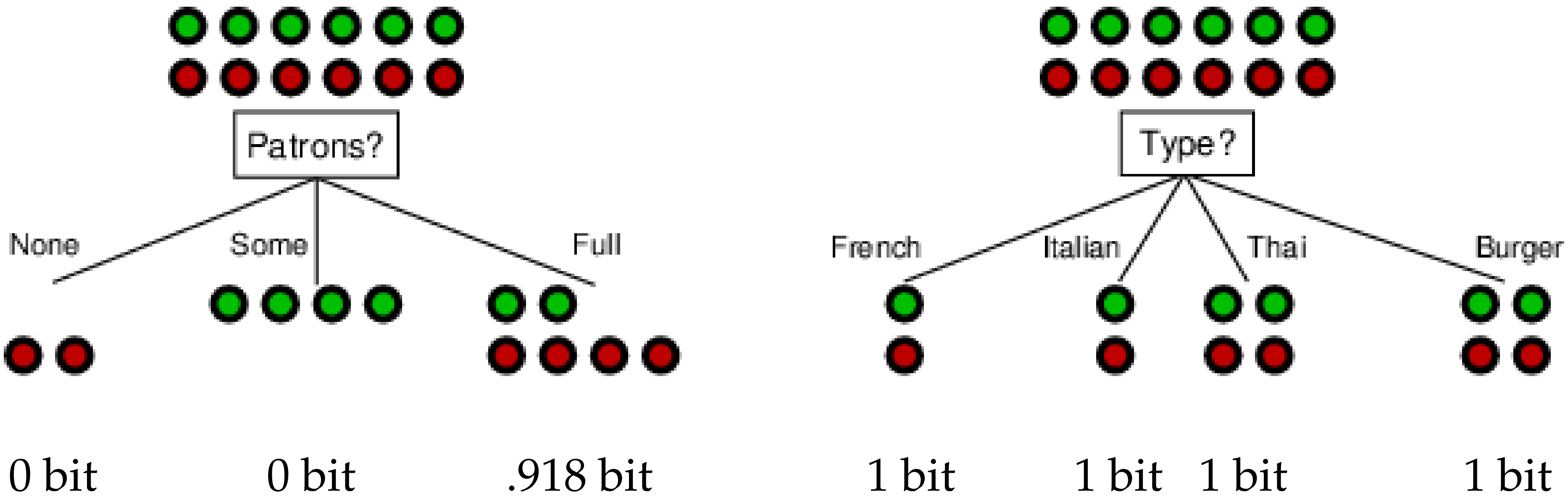
- Suppose we have p positive and n negative examples at the root
  $\Longrightarrow$ $H(\langle p/(p+n), n/(p+n)\rangle)$ bits needed to classify a new example
  E.g., for 12 restaurant examples, $p=n=6$ so we need 1 bit

- An attribute splits the examples E into subsets E_i
  each needs less information to complete the classification

- Let E_i have p_i positive and n_i negative examples
  $\Longrightarrow$ $H(\langle p_i/(p_i+n_i), n_i/(p_i+n_i)\rangle)$ bits needed to classify a new example
  $\Longrightarrow$ **expected** number of bits per example over all branches is

$$\sum_i \frac{p_i+n_i}{p+n} H(\langle p_i/(p_i+n_i), n_i/(p_i+n_i)\rangle)$$

Select Attribute

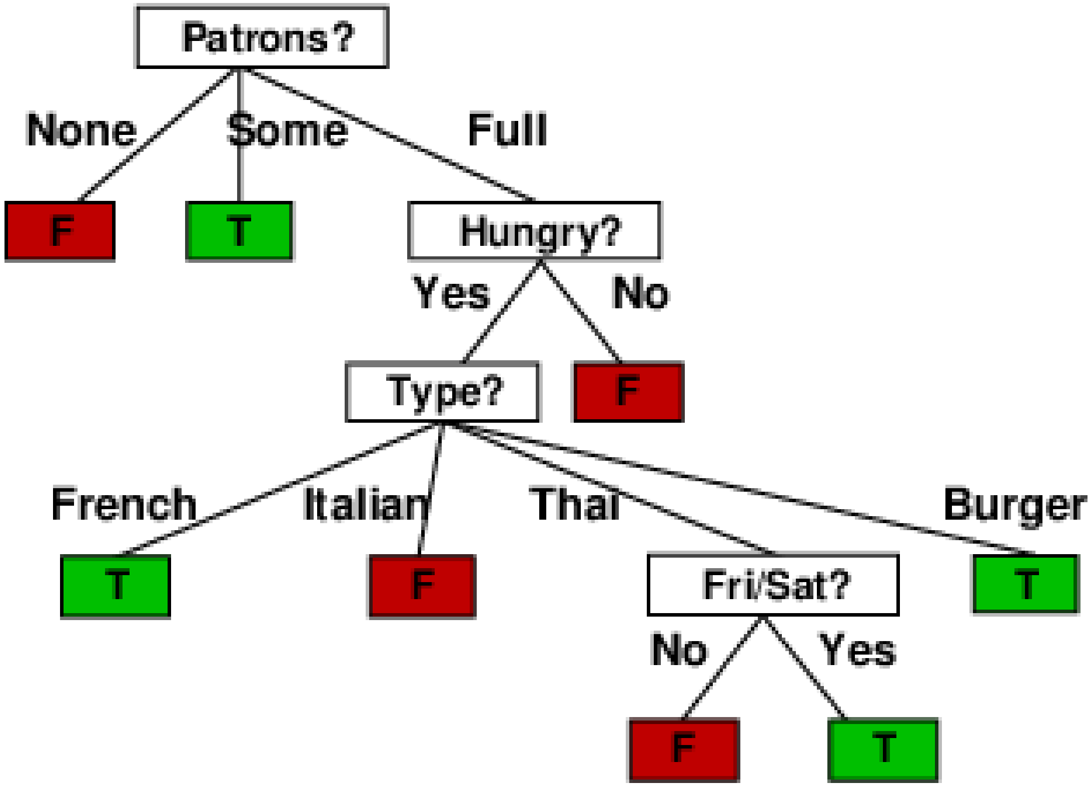


- *Patrons?*: 0.459 bits
- *Type*: 1 bit

⇒ Choose attribute that minimizes remaining information needed

Example

- Decision tree learned from the 12 examples:



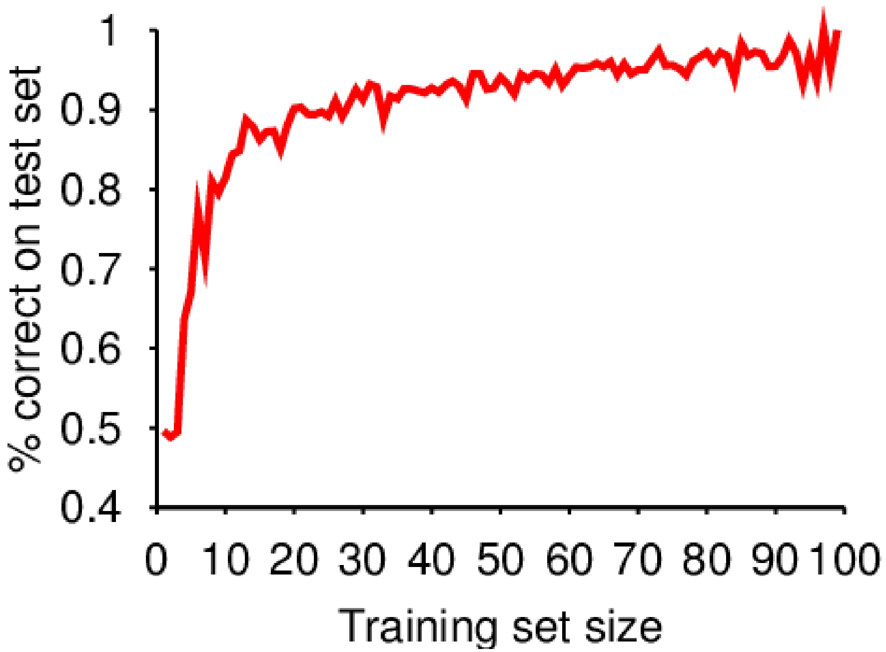
- Substantially simpler than "true" tree
  (a more complex hypothesis isn't justified by small amount of data)

performance measurements

Performance Measurement

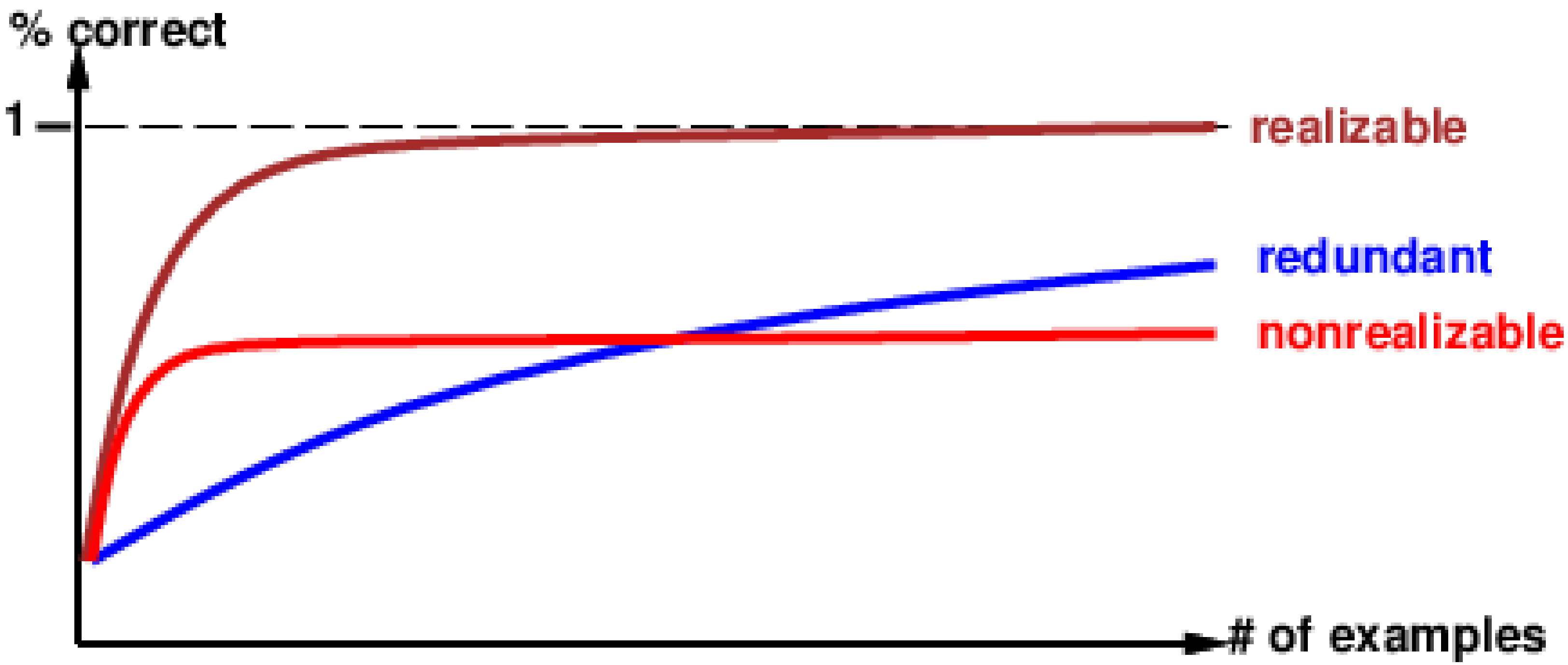
- How do we know that $h \approx f$? (Hume's **Problem of Induction**)
  - Use theorems of computational/statistical learning theory
  - Try h on a new test set of examples
    (use **same distribution over example space** as training set)

- Learning curve = % correct on test set as a function of training set size

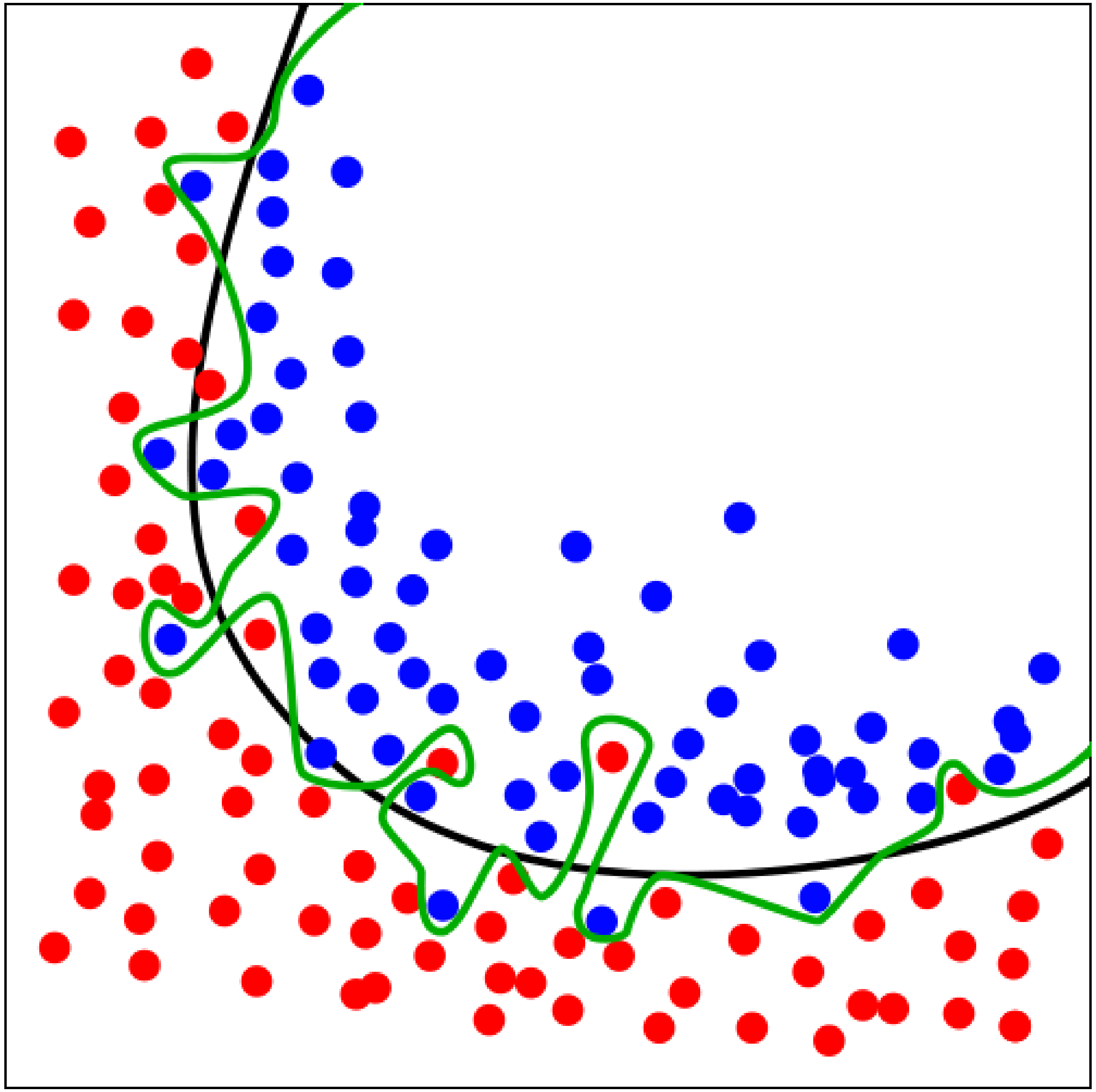


Performance Measurement

- Learning curve depends on
  - realizable (can express target function) vs. non-realizable
    non-realizability can be due to missing attributes
    or restricted hypothesis class (e.g., thresholded linear function)
  - redundant expressiveness (e.g., loads of irrelevant attributes)



Overfitting



bayesian learning

Full Bayesian Learning

- View learning as Bayesian updating of a probability distribution over the hypothesis space
  - H is the hypothesis variable, values $h_1, h_2, \ldots$, prior $\mathbf{P}(H)$
  - jth observation d_j gives the outcome of random variable D_j training data $\mathbf{d} = d_1, \ldots, d_N$
- Given the data so far, each hypothesis has a posterior probability:

$$P(h_i|\mathbf{d}) = \alpha P(\mathbf{d}|h_i)P(h_i)$$

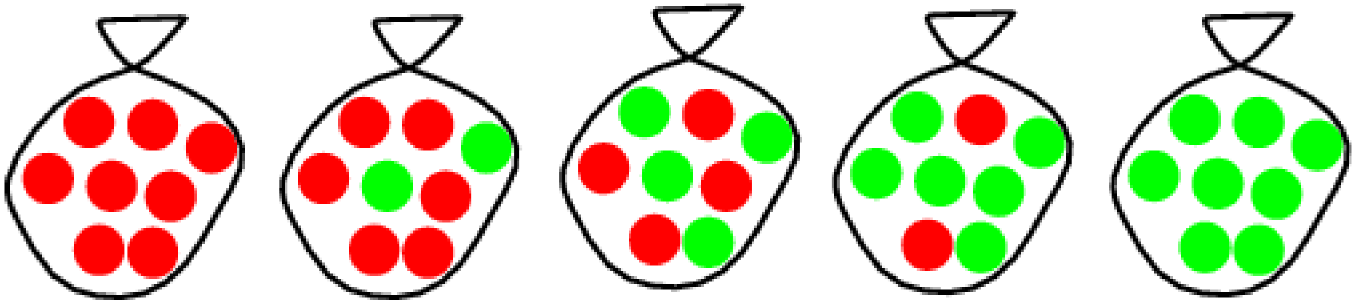
where $P(\mathbf{d}|h_i)$ is called the likelihood

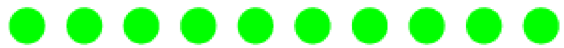
- Predicting next data point uses likelihood-weighted average over hypotheses:

$$\mathbf{P}(X|\mathbf{d}) = \sum_i \mathbf{P}(X|\mathbf{d}, h_i)P(h_i|\mathbf{d}) = \sum_i \mathbf{P}(X|h_i)P(h_i|\mathbf{d})$$

Example

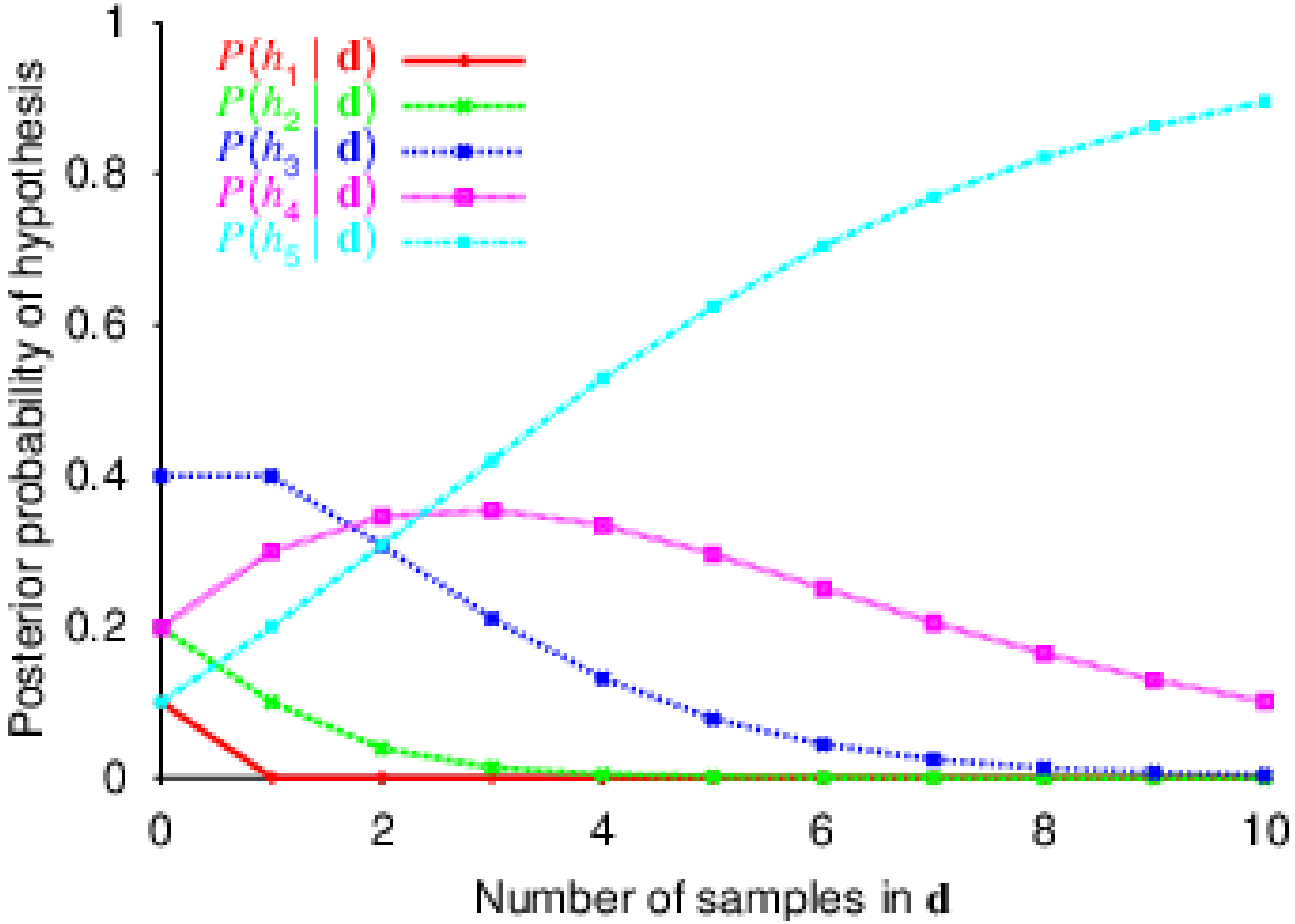
- Suppose there are five kinds of bags of candies:
  10% are h_1: 100% cherry candies
  20% are h_2: 75% cherry candies + 25% lime candies
  40% are h_3: 50% cherry candies + 50% lime candies
  20% are h_4: 25% cherry candies + 75% lime candies
  10% are h_5: 100% lime candies



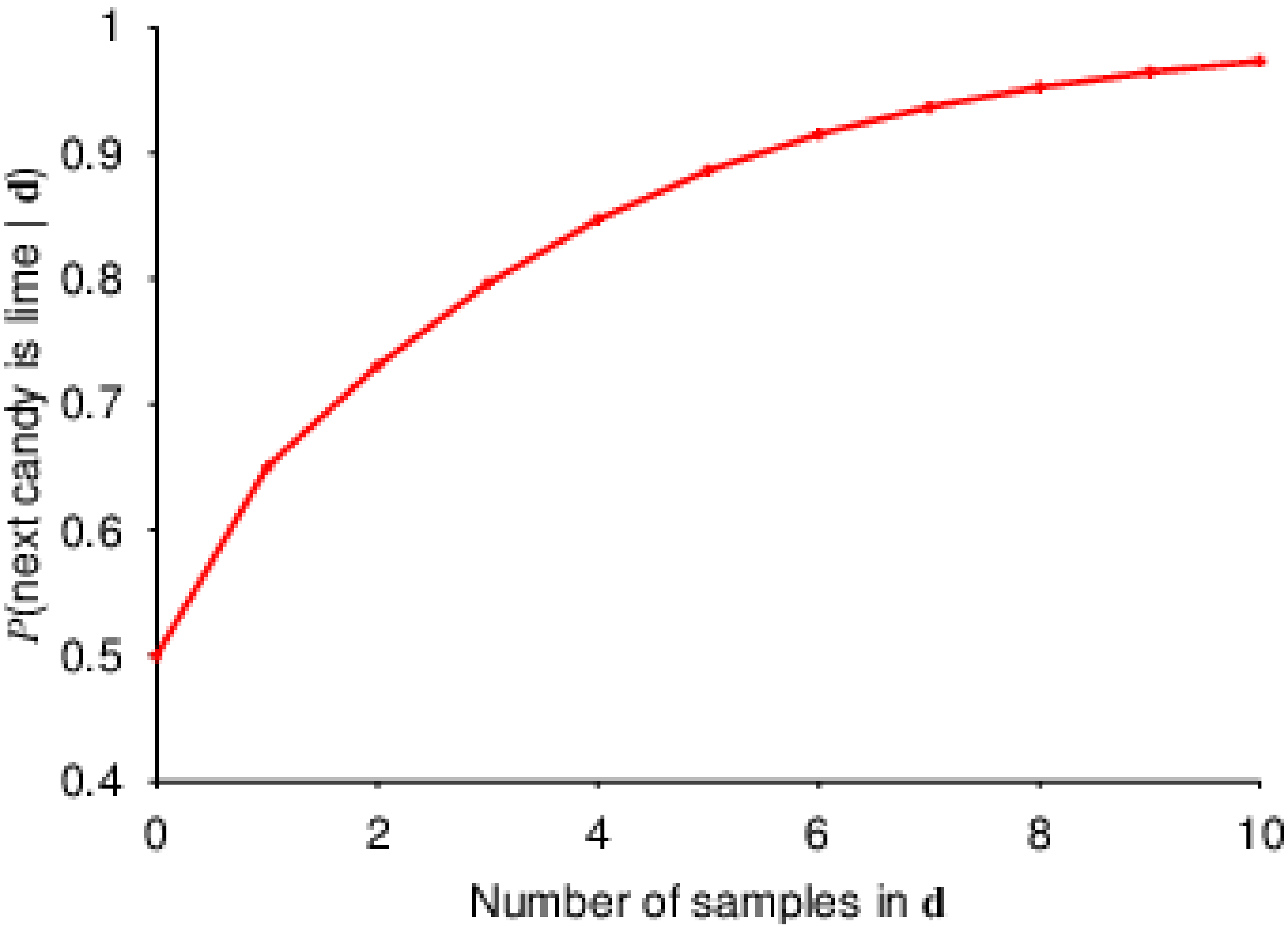
- Then we observe candies drawn from some bag: 

- What kind of bag is it? What flavour will the next candy be?

Posterior Probability of Hypotheses



Prediction Probability



Maximum A-Posteriori Approximation

- Summing over the hypothesis space is often intractable
  (e.g., 18,446,744,073,709,551,616 Boolean functions of 6 attributes)

- Maximum a posteriori (MAP) learning: choose h_{MAP} maximizing $P(h_i|\mathbf{d})$

- I.e., maximize $P(\mathbf{d}|h_i)P(h_i)$ or $\log P(\mathbf{d}|h_i) + \log P(h_i)$

- Log terms can be viewed as (negative of)

  bits to encode data given hypothesis + bits to encode hypothesis

  This is the basic idea of minimum description length (MDL) learning

- For deterministic hypotheses, $P(\mathbf{d}|h_i)$ is 1 if consistent, 0 otherwise
  $\Longrightarrow$ MAP = simplest consistent hypothesis

Maximum Likelihood Approximation

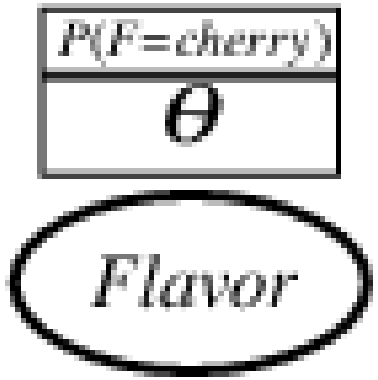
- For large data sets, prior becomes irrelevant

- Maximum likelihood (ML) learning: choose h_{ML} maximizing $P(\mathbf{d}|h_i)$

⇒ Simply get the best fit to the data; identical to MAP for uniform prior (which is reasonable if all hypotheses are of the same complexity)

- ML is the "standard" (non-Bayesian) statistical learning method

ML Parameter Learning in Bayes Nets



- Bag from a new manufacturer; fraction θ of cherry candies?

- Any θ is possible: continuum of hypotheses h_θ
  θ is a parameter for this simple (binomial) family of models

- Suppose we unwrap N candies, c cherries and $\ell = N - c$ limes
  These are i.i.d. (independent, identically distributed) observations, so

$$P(\mathbf{d}|h_\theta) = \prod_{j=1}^{N} P(d_j|h_\theta) = \theta^c \cdot (1-\theta)^\ell$$

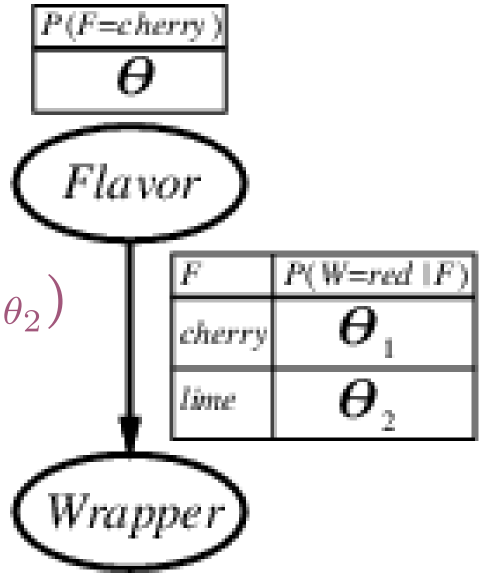
- Maximize this w.r.t. θ—which is easier for the log-likelihood:

$$L(\mathbf{d}|h_\theta) = \log P(\mathbf{d}|h_\theta) = \sum_{j=1}^{N} \log P(d_j|h_\theta) = c\log\theta + \ell\log(1-\theta)$$

$$\frac{dL(\mathbf{d}|h_\theta)}{d\theta} = \frac{c}{\theta} - \frac{\ell}{1-\theta} = 0 \qquad \Longrightarrow \qquad \theta = \frac{c}{c+\ell} = \frac{c}{N}$$

Multiple Parameters

- Red/green wrapper depends probabilistically on flavor



- Likelihood for, e.g., cherry candy in green wrapper

$$\begin{aligned} & P(F = cherry, W = green | h_{\theta,\theta_1,\theta_2}) \\ = \quad & P(F = cherry | h_{\theta,\theta_1,\theta_2}) P(W = green | F = cherry, h_{\theta,\theta_1,\theta_2}) \\ = \quad & \theta \cdot (1 - \theta_1) \end{aligned}$$

- N candies, r_c red-wrapped cherry candies, etc.:

$$P(\mathbf{d} | h_{\theta,\theta_1,\theta_2}) = \theta^c (1-\theta)^\ell \cdot \theta_1^{r_c} (1-\theta_1)^{g_c} \cdot \theta_2^{r_\ell} (1-\theta_2)^{g_\ell}$$

$$\begin{aligned} L = \quad & [c \log \theta + \ell \log(1-\theta)] \\ + \quad & [r_c \log \theta_1 + g_c \log(1-\theta_1)] \\ + \quad & [r_\ell \log \theta_2 + g_\ell \log(1-\theta_2)] \end{aligned}$$

Multiple Parameters

- Derivatives of L contain only the relevant parameter:

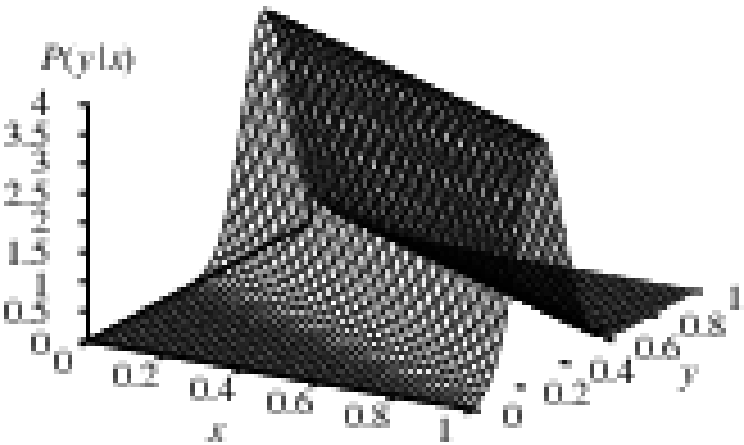
$$\frac{\partial L}{\partial \theta} = \frac{c}{\theta} - \frac{\ell}{1-\theta} = 0 \qquad \Longrightarrow \qquad \theta = \frac{c}{c+\ell}$$

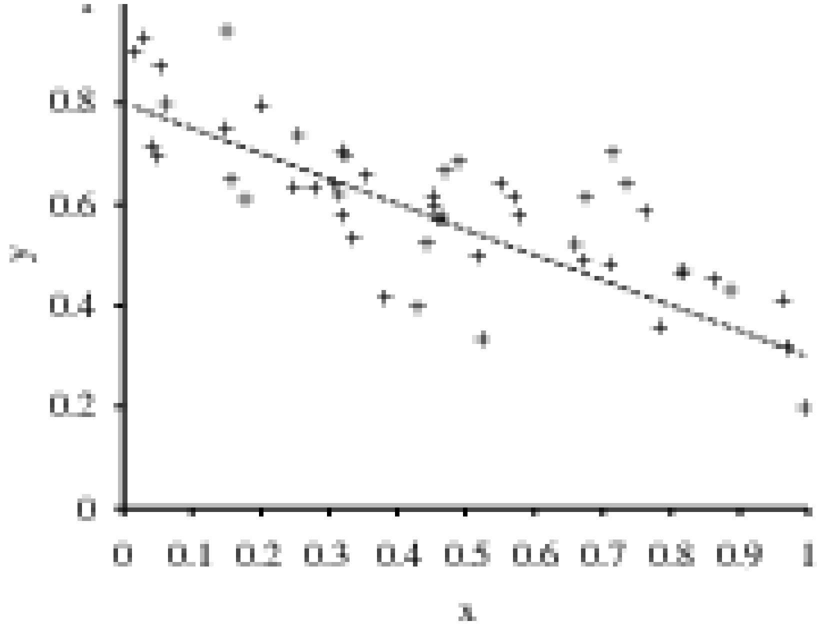
$$\frac{\partial L}{\partial \theta_1} = \frac{r_c}{\theta_1} - \frac{g_c}{1-\theta_1} = 0 \qquad \Longrightarrow \qquad \theta_1 = \frac{r_c}{r_c+g_c}$$

$$\frac{\partial L}{\partial \theta_2} = \frac{r_\ell}{\theta_2} - \frac{g_\ell}{1-\theta_2} = 0 \qquad \Longrightarrow \qquad \theta_2 = \frac{r_\ell}{r_\ell+g_\ell}$$

- With complete data, **parameters can be learned separately**

Regression: Gaussian Models





- Maximizing $P(y|x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y-(\theta_1 x+\theta_2))^2}{2\sigma^2}}$ w.r.t. θ_1, θ_2

  = minimizing $E = \sum_{j=1}^{N} (y_j - (\theta_1 x_j + \theta_2))^2$

- That is, minimizing the sum of squared errors gives the ML solution for a linear fit **assuming Gaussian noise of fixed variance**

Many Attributes

- Recall the "wait for table?" example: decision depends on *has-bar, hungry?, price, weather, type of restaurant, wait time, ...*

- Data point $\mathbf{d} = (d_1, d_2, d_3, ..., d_n)^T$ is high-dimensional vector

⇒ $P(\mathbf{d}|h)$ is very sparse

- Naive Bayes

$$P(\mathbf{d}|h) = P(d_1, d_2, d_3, ..., d_n|h) = \prod_i P(d_i|h)$$

(independence assumption between all attributes)

How To

1. Choose a parameterized family of models to describe the data
   requires substantial insight and sometimes new models

2. Write down the likelihood of the data as a function of the parameters
   may require summing over hidden variables, i.e., inference

3. Write down the derivative of the log likelihood w.r.t. each parameter

4. Find the parameter values such that the derivatives are zero
   may be hard/impossible; modern optimization techniques help

Summary

- Learning needed for unknown environments
- Learning agent = performance element + learning element
- Learning method depends on type of performance element, available feedback, type of component to be improved, and its representation
- Supervised learning
- Decision tree learning using information gain
- Learning performance = prediction accuracy measured on test set
- Bayesian learning