Künstliche Intelligenz 2 - Summary

Sebastian Rietsch

July 29, 2017

1 Probablistic Reasoning, Part I: Basics

1.1 Introduction

Sources of Uncertainty in Decision-Making:

- Non-deterministic actions
- Partial observability with unreliable sensors
- Uncertainty about the domain behavior

What is **probabilistic reasoning**?

- Deducing probabilities from knowledge about *other* probabilities
- Determines probabilities that are difficult to assess, based on probabilities that are (relatively) easy to assess (*to asses:* the process of estimating a probability P using statistics)

Rational Agents:

- We have a choice of **actions**
- These can lead to different solutions with different probabilities
- The actions have different costs
- The results have different utilities
- A rational agent chooses the action with the **maximum expected utility**

1.2 Unconditional Probabilities

A **probability theory** is an assertion language for talking about possible worlds and an inference method for quantifying the degree of belief in such assertions (Example: we roll two dice with six sides, then we have 36 possible worlds: $(1,1),(2,1),\ldots,(6,6)$).

A **probability model** $\langle \Omega, P \rangle$ consists of a set Ω of possible worlds called the sample space and a probability function $P : \Omega \to \mathbb{R}$, such that $0 \leq P(\omega) \leq 1$ for all $\omega \in \Omega$ and $\sum_{\omega \in \Omega} P(\omega) = 1$. We restrict ourselves to a discrete, countable sample space.

Random variables:

- A random variable is a variable quantity whose value depends on possible outcomes of unknown variables and processes we do not understand
- Given a random variable X, $P(X = x)$ denotes the prior probability, or unconditional probability, that X has value x
- We will refer to the fact $X = x$ as an event or outcome
- For Boolean variable $Name$, we write name for $Name = \top$ and $\neg name$ for $Name = \bot$

The **probability distribution** for a random variable X, written $P(X)$, is the vector of probabilities for the (ordered) domain of X.

Given a subset $Z \subseteq \{X_1, \ldots, X_n\}$ of random variables, an event is an assignment of values to the variables in Z. The **joint probability distribution**, written $P(Z)$, lists the probabilities of all events.

An **atomic event** is an assignment of values to all variables.

A **proposition** describes a set of multiple atomic events (**Example:** we roll two dice and are interested in the cases where they add up to 11). The probability associated with a proposition is defined by the

sum of probabilities of the worlds in which it holds: For any proposition $\phi, P(\phi) = \sum_{\omega \in \phi} P(\phi)$.

Kolmogorov axioms:

1. The probability of an event is a non-negative real number:

$$P(E) \in \mathbb{R}, P(E) \geq 0, \forall E \in F$$

   where F is the event space.

2. The probability that at least one of the elementary events in the entire sample space will occure is 1: $P(\Omega) = 1$.
3. Any countable sequence of disjoint sets (mutually exclusive events) $E_1, E_2, \ldots$ satisfies

$$P(\bigcup_{i=1}^{\infty}(E_i)) = \sum_{i=1}^{\infty} P(E_i).$$

From this follows $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$.

1.3 Conditional Probabilities

In the presence of additional information, we can no loger use the unconditional (**prior**) probabilities (because probabilies model our belief, thus they depend on our knowledge).

Given propositions A and B, $P(a|b)$ denotes the **conditional probability** of a given that all we know is b. The conditional, or posterior probability of a given b is defined as:

$$P(a|b) = \frac{P(a \wedge b)}{P(b)}$$

The **conditional probability distribution** of $P(X|Y)$ is the table of all conditional probabilies of values of X given values of Y.

1.4 Independence

Problem with full joint probility distributions: Given n random variables with k values each, the joint pdf containts k^n probabilities. This motivates the use of conditional probabilities while exploiting the **(conditional) independence** property.

Events a and b are **independent** if $P(a \wedge b) = P(a) \cdot P(b)$, from which follows that $P(a|b) = P(a)$ and vice versa.

In case of independence, the joint probability distribution of multiple variables can be reconstructed from smaller, statistically independent joint probability distributions.

1.5 Basic Probabilistic Reasoning Methods

- **Product Rule:** $P(a \wedge b) = P(a|b)P(b)$
- **Chain Rule:** $P(X_1, \ldots, X_n) = P(X_n|X_{n-1}, \ldots X_1) \cdot \cdots \cdot P(X_2|X_1) \cdot P(X_1)$ (This works for any ordering of the variables)
- **Marginalization:** Given sets X and Y of random variables, we have:

$$P(X) = \sum_{y \in Y} P(X, y)$$

- **Normalization:** Given a vector $\langle w_1, \ldots, w_k \rangle$ of numbers in $[0, 1]$ where $\sum_{i=1}^{k} w_i \leq 1$, the normalization constant α is $\alpha\langle w_1, \ldots, w_k \rangle = 1/\sum_{i=1}^{k} w_i$
  - Given a random variable X and an event e, we have $P(X|e) = \alpha P(X, e)$ with $\alpha = 1/P(e)$
  - Normalization + Marginalization: Given "query variable" X, "observed event" e, and "hidden variables" set Y:

$$P(X|e) = \alpha P(X, e) = \alpha \sum_{y \in Y} P(X, e, y)$$

1.6 Bayes' Rule

Given propositions A and B where $P(a) \neq 0$ and $P(b) \neq 0$, we have:

$$P(a|b) = \frac{P(b|a)P(a)}{P(b)}$$

1.7 Conditional Independence

Given sets of random variables Z_1, Z_2 and Z, we say Z_1 and Z_2 are **conditionally independent given** Z if:

$$P(Z_1, Z_2|Z) = P(Z_1|Z) \cdot P(Z_2|Z)$$

If Z_1 and Z_2 are conditionally independent given Z, the $P(Z_1|Z_2, Z) = P(Z_1|Z)$.

1.8 Summary

- Uncertainty is unavoidable in many environsments, namely whenever agents do not have perfect knowledge
- Probabilities express the degree of belief of an agent, given its knowledge, into an event
- Conditional probabilities express the likelihood of an event given observed evidence
- Assessing a probability means to use statistics to approximate the likelihood of an event
- Bayes' rule allows us to derive, from probabilities that are easy to assess, probabilities that arn't easy to assess
- Given mulitple evidence, we can exploit conditional independencie

2 Probabilistic Reasoning, Part II: Bayesian Networks

2.1 What is a Bayesian Network

Given random variables $X_1, \ldots, X_n$ with finite domains $D_1, \ldots, D_n$, a **Bayesian network** is an acyclic directed graph $BN = \langle \{X_1, \ldots, X_n\}, E \rangle$. We denote $Parents(X_i) = \{X_j | (X_j, X_i) \in E\}$. Each X_i is associated with a function $CPT(X_i) : D_i \times \prod_{X_j \in Parents(X_i)} D_j \to [0, 1]$, the **conditional probability table**. Is is a so called **graphical model**.

A node X is conditionally independent of its non-descendants (nicht Nachkommen) given its parents. A node X is conditionally independent of all other nodes in the network given its Markov blanket (parents, children, and children's parents).

2.2 Recovering the Full Joint Probability Distribution

Chain rule: For any ordering $X_1, \ldots, X_n$ we have

$$P(X_1, \ldots, X_n) = P(X_n | X_{n-1} \ldots, X_1) \cdot P(X_{n-1} | X_{n-2} \ldots, X_1) \ldots P(X_1)$$

Choose $X_1, \ldots, X_n$ consistent with $BN : X_j \in Parents(X_i) \Rightarrow j < i$

With BN assumption, we can use $P(X_i | Parents(X_i))$, instead of $P(X_i | X_{i-1} \ldots X_1)$:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | Parents(X_i))$$

2.3 Constructing Bayesian Networks

Reminder: Conditional Independence

Events R and B are conditionally independent given Y if and only if $P(R, B|Y) = P(R|Y)P(B|Y)$ or equivalently $P(R|B, Y) = P(R|Y)$.

A Bayesian network is only a correct representation of a domain if each node is conditionally independent of its other predecessors (Vorfahren) in the node ordering, given its parents. We can satisfy this condition with this methodology:

1. Nodes: First determine the set of variables that are required to model the domain. Now order them, $X_1, \ldots, X_n$. Any order will work, but the resulting network will be more compact if the variables are orderes such that cuases precede effects.
2. Links: For $i = 1$ to n do:
   - Choose, from $X_1, \ldots, X_{i-1}$ a minimal set of parents for X_i, such that conditional dependence is satisfied
   - For each parent insert a link from parent to X_i
   - CPTs: Write down $P(X_i | Parents(X_i))$

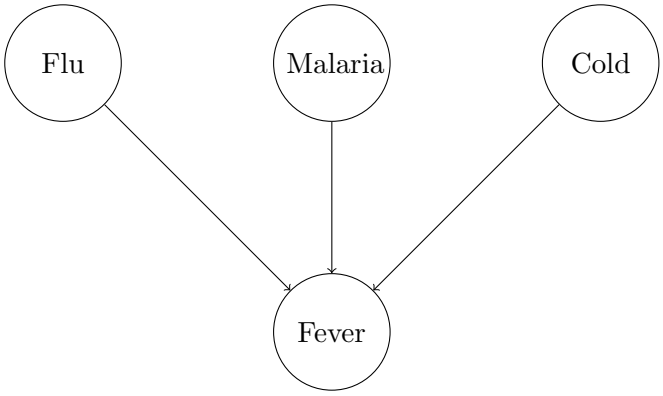
The size on a Bayesian Network $size(BN)$ is defined as the total number of entries in the CPTs. Bayesian Networks are compact is each variable is directly influenced by only a few of its predecessor variables.

2.4 Efficient Representation of Conditional Distributions

Even if the maximum number of parents k is smallish, filling in the CPT for a node requires up to $O(2^k)$ numbers and perhaps a great deal of experience with all the possible conditioning cases.

Deterministic node: has its value specified exactly by the values of its parents, with no uncertainty $\Rightarrow$ no explicit CPT needed

Uncertain relationships can often be characterized by so-called noisy logical relationships, where the **noisy-OR** relation is the standard example (a good exmaple is the relationship between diseases and symptoms).



The causal relationship between parent and child may be inhibited (e.g. no fever but has flu). Two assumptions:

- All the possible causes are listed (If some are missing, we can always add a so-called leak node that covers "miscellaneous causes")
- Inhibition of each parent is independent of any other parent (Whatever inhibits malaria from causing a fever is independet of whatever inhibits flu from causing fever)

Given these assumptions, $Fever$ is false if and only if all its $true$ parents are inhibited

$\Rightarrow P(x_i|parents(X_i)) = 1 - \prod_{j:X_j=true} q_j$, where q_j are the inhibition probabilities.

The full CPT can now be described using $O(k)$ parameters.

2.5 Inference in Bayesian Networks (Inference by enumeration)

The basic task for any probabilistic inference system is to compute the posterior probability distritubtion for a set of **query variable**(s) X, given some observed **event** e - that is, some assignment of values to a set of **evidence variables** $E = \{E_1, \ldots, E_m\}$. $Y = \{Y_1, \ldots, Y_l\}$ will denote the nonevidence, nonquery variables, also called **hidden variables**.

Conditional probabilities can be computed by summing terms from the full joint probability distribution:

$$P(X|e) = \alpha P(X, e) = \alpha \sum_y P(X, e, y)$$

Reminder: $P(X|e) = \frac{P(X,e)}{P(e)} = \alpha P(X, e),\ \alpha = \frac{1}{P(e)}$

A query can be answered using a Bayesian network by computing sums of products of conditional probabilities from the network.

Reminder: $P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|Parents(X_i))$

Example (hidden variables: *Earthquake* and *Alarm*):

$$\begin{aligned} P(Burglary|JohnCalls = \top, MaryCalls = \top) = P(B|j,m) &= \alpha \sum_e \sum_a P(B,j,m,e,a) \\ &= \alpha \sum_e \sum_a P(b)P(e)P(a|b,e)P(j|a)P(m|a) \\ &= \alpha P(b) \sum_e P(e) \sum_a P(a|b,e)P(j|a)P(m|a) \end{aligned}$$

Variable Elimnation can make computation increadibly faster in the case of polytrees (there is at most one undirected path between any two nodes in the graph).

General probabilistic inference is $\#P$-hard, which is harder then NP. In the hard cases, approxiamation is key.

2.6 Conclusion

- **Bayesian networks** are a wide-spread tool to model uncertainty, and to reason about it. A BN represents **conditional independence** relations between random variables. It consists of a graph encoding the variable dependencies, and of conditional probability tables (CPTs).
- Given a variable order, the BN is small (concerning CPT sizes) if every variable depends only on a few of its predecessors
- **Probabilistic inference** requires to compute the probability distribution of a set of query variables, given a set of evidence variables whose values we know. The remaining variables are hidden.
- Inference by enumeration takes a BN as input, then applies **Normalization and Marginalization**, the **Chain rule**, and exploits conditional independence.
- **Variable elimination** avoids unnecessary computation. It runs in polynomial time for polytree BNs. In general, exact probabilistic inference is $\#P-hard$. Approximate probabilistic inference methods exist.

3 Making Simple Decisions Rationally

3.1 Rational Preference

Decision theory investigates how an agent a deals with choosing among actions based on the desirability of their outcomes. We restrict ourself to **episodic** decision theory, which deals with choosing among actions based on the desirabilty of their immediate outcomes. We have to deal with non-deterministic, partially observable environments.

$Result(a)$ is a random variable, whose values are possible outcome states of an action. The probability of outcome s', given evidence observations e, is written $P(Result(a) = s'|a, e)$.

The agent's preferences are captured by a **utility function** $U(s)$, which assigns a single number to express the desirability of a state. The **expected utility** of an action given the evidence, $EU(a|e)$, is just the average utility value of the outcomes, weighted by the probability that the outcome occurs:

$$EU(a|e) = \sum_{s'} P(Result(a) = s'|a, e)U(s')$$

The principle of **maximum expected utility** (MEU) says that a rational agent should choose the action that maximizes the agent's expected utility:

$$action = argmax_a EU(a|e)$$

3.1.1 Preference

An agent chooses among **outcomes** (A, B, etc.) and **lotteries**, i.e., situations with uncertain prizes.

$$Lottery\ L = [p_1, S_1; p_2, S_2; \ldots p_n, S_n]$$

- $A \succ B$: A **preffered** over B
- $A \sim B$: **indifference** between A and B
- $A \succeq B$: A prefered over B or indifferent

Six constraint that we require any reasonable preference relation to obey: Orderability, Transitivity, Continuity, Substituability, Monotonicity, Decomposability (Axioms of utility theory).

3.1.2 Preference lead to utility

- **Existence of Utility Function:** If an agent's preferences obey the axioms of utility, then there exists a function U such that $U(A) > U(B)$ if and only if A is preferred to B, and $U(A) = U(B)$ if and only if the agent is indifferent between A and B.

$$U(A) > U(B) \Leftrightarrow A \succ B$$

- **Expected Utility of a Lottery:** The utility of a lottery is the sum of probability of each outcome times the utility of that outcome.

$$U([p_1, S_1; \ldots; p_n, S_n]) = \sum_i p_i U(S_i)$$

We call a total preference ordering on states a **value function** or **ordinal utility function**.

Note: An agent doesn't implicitly have to maximize an utility function. Rational behavior can be generated in any number of ways. By observing a rational agent's preference, however, an observer can construct the utility function that represents what the agent is actually trying to achieve (even if the agent doesn't know it)

3.1.3 Utility assessment and utility scales

Goal: working out a agent's utility function (**preference elicitation**). This process involves presenting choices to the agent and using the observed preferences to pin down the underlying utility function.

Fix the utility of a "best possible price" at $U(S) = u_\top$ and a "worst possible catastrophe" at $U(S) = u_\bot$. **Normalized utilities** use a scale with $u_\bot = 0$ and $u_\top = 1$.

Given a utility scale between $u_\bot$ and $u_\top$, we can assess the utility of any particular prize S by asking the agent to choose between S and a **standard lottery** $[p, u_\top; (1-p), u_\bot]$. The probability p is adjusted until the agent is indifferent between S and the standard lottery. Assuming normalized utilities, the utility of S is given by p. Once this is done for each prize, the utilities for all lotteries involving those prizes are determined.

3.2 Multiattribute Utility Functions

Problems, in which outcomes are characterized by two or more attributes, are handled by **multiattribute utility theory**. We will call the attributed $X = X_1, \ldots, X_n$; a complete vector of assignments will be $x = \langle x_1, \ldots, x_n \rangle$, where each x_i is either a numeric or discrete value. We will assume that higher values of an attribute correspond to higher utilities.

3.2.1 Dominance

A choice S_1 is said to **strictly** dominate S_2, iff $X_i(S_1) > X_i(S_2),\ \forall i$. For the non-deterministic case, all possible concrete outcomes for S_1 have to strictly dominante all possible outcomes for S_2 (very rare).

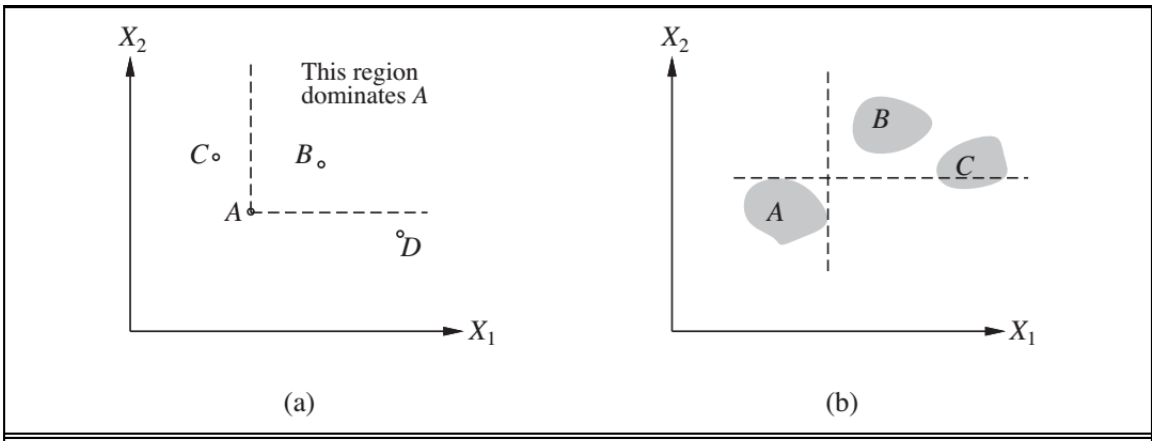


Figure 16.4 Strict dominance. (a) Deterministic: Option A is strictly dominated by B but not by C or D. (b) Uncertain: A is strictly dominated by B but not by C.

If two actions A_1 and A_2 lead to probability distributions $p_1(x)$ and $p_2(x)$ on attribute X, then A_1

statistically dominates A_2 if

$$\forall x \int_{-\infty}^{x} p_1(x')dx' \leq \int_{-\infty}^{x} p_2(x')dx'$$

(the cumulative distribution of S_1 is always to the right of the one for S_2).

If A_1 stochatically dominates A_2, then for any monotonically nondecreaing utility function $U(x)$, the expected utility of A_1 is at least as high as the expected utility of A_2. Hence, if an action is stochastically dominated by another action on all attributes, then it can be discarded.

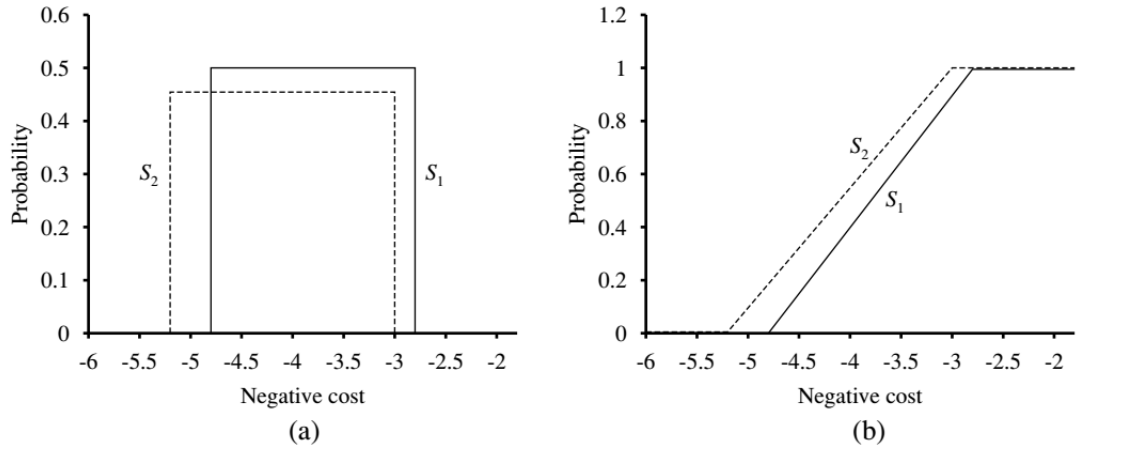


Figure 16.5 Stochastic dominance. (a) S_1 stochastically dominates S_2 on cost. (b) Cumulative distributions for the negative cost of S_1 and S_2.

3.2.2 Preference structure and multiattribute utility

Suppose we have n attributes, each of which has d distinct possible values. To specify the complete utility function $U(x_1, \ldots, x_n)$, we need d^n values in the worst case. Now, the worst case corresponds to a situation in which the agent's preferences have no regularity at all. Multiattribute utility theory is based on the suppostition that the preferences of typical agents have much more structure than that.

The basic approach is to identify regularities in the preference behavior and to use what are called **representation theorems** to show that an agent with a certain kind of preference structure has a utility function

$$U(x_1, \ldots, x_n) = F[f_1(x_1), \ldots, f_n(x_n)]$$

where F is, we hope, a simple function such as addition.

... whatever (Was wollen sie mir hier genau beibringen Herr Kohlhase?)

3.3 Decision Networks

Decision networks combine Bayesian networks with additional node types for actions and utilities.

3.3.1 Representing a decision problem with a decision network

In it's most general form, a decision network represents information about the agent's current state, its possible actions, the state that will result from the agent's action, and the utility of that state.

Node types:

- **Chance nodes** (oval) represent random variables. Each node has associated with it a conditional distribution that is indexed by the state of the parent nodes.
- **Decision nodes** (rectangle) represent points where the decision maker has a choice of actions.
- **Utility nodes:** (diamonds) represent the agent's utility function. The utility node has as parents all variables describing the outcome that directly affect utility. Associated with the utility node is a description of the agent's utility as a function of the parents attributes.

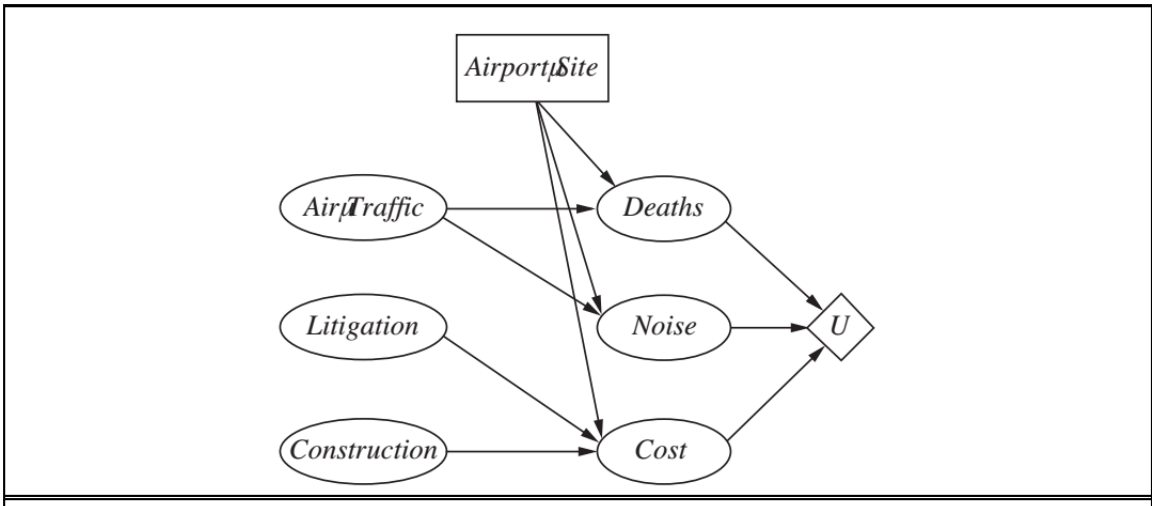


Figure 16.6 A simple decision network for the airport-siting problem.

Actions are selected by evaluating the decision network for each possible setting of the decision node.
Algorithm:

1. Set the evidence variables to the current state
2. For each possible value of the decision node:
   a) Set the decision node to that value
   b) Calculate the posterior probabilities for the parent nodes of the utility node, using a standard probabilistic inference algorithm
   c) Calculate the resulting utility for the action
3. Return the action with the highest utility

3.4 The Value of Information

Information value theory enables an agent to choose what information to acquire.

3.4.1 Example

- Oil company wants to buy one of n blocks of ocean-drill rights
- One of the blocks contains oil worth C dollar, while others are worthless
- Suppose seismologist offers the company results of a survey of block number 3, which indicates definitively wether the block contains oil
- How much should the company be willing to pay for the information?

With probability $1/n$, the survey will indicate oil in block 3 and the company will buy block 3 for

C/n dollars

$\Rightarrow Profit = C - C/n = (n-1)C/n$

With probability $n-1)/n$, the survey will show that block contains no oil, in which case the company will buy a different block

$\Rightarrow Expected\ Profit = C/(n-1) - C/n = C/(n-1)n$

Expected profit, given the survey information: $\frac{1}{n} \times \frac{(n-1)C}{n} + \frac{n-1}{n} \times \frac{C}{n(n-1)} = C/n$

Therefore, the company should be willing to pay the seismologist up to C/n dollars for the information: the information is worth as much as the block itself.

In general, the value of a given piece of information is defined to be the difference in expected value between best action before and after information is obtained.

3.4.2 A general formula for perfect imformation

We assume that exact evidence can be obtained about the value of some random variable E_j (that is, we learn $E_j = e_j$, so the phrase **value of perfect information** (VPI) is used.

Let the agent's initial evidence be e. Then the value of the current best action α is defined by

$$EU(\alpha|e) = max_a \sum_{s'} P(Result(a) = s'|a, e)U(s')$$

and the value of the next best action (after the new evidence $E_j = e_j$ is obtained) will be

$$EU(\alpha_{e_j}|e, e_j) = max_a \sum_{s'} P(Result(a) = s'|a, e, e_j)U(s')$$

But E_j is a random variable whose value is currently unknown, so to determine the value of discovering E_j, given current information e we must average over all possible values e_{jk} that we might discover, using our current beliefs about its value:

$$VPI_e(E_j) = (\sum_k P(E_j = ejk|e)EU(\alpha_{e_{jk}}|e, E_j = e_{jk})) - EU(\alpha|e)$$

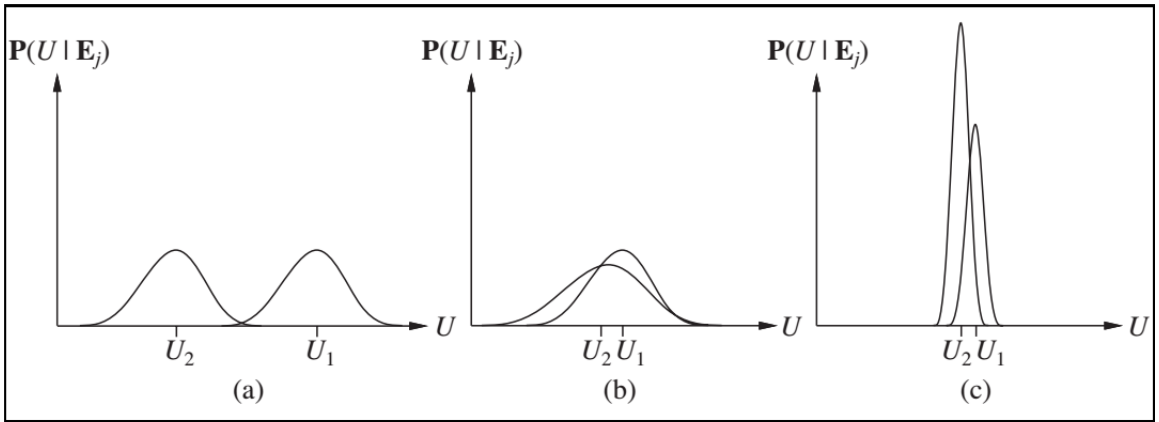


Figure 16.8 Three generic cases for the value of information. In (a), a_1 will almost certainly remain superior to a_2, so the information is not needed. In (b), the choice is unclear and the information is crucial. In (c), the choice is unclear, but because it makes little difference, the information is less valuable. (Note: The fact that U_2 has a high peak in (c) means that its expected value is known with higher certainty than U_1.)

- Suppose a_1, a_2 represent two different routes through mountain range
- (a)
  - a_1 nice and chill, a_2 winding road, might snow as fuck and way might be blocked
  - Clearly: $U_1 > U_2$
  - Possibility to obtain satellite images
  - Distribution of new expectations shows that, about 99.9% of the time the plans won't change
- (b)
  - Choosing between two similar roads which could be blocked and we are carrying a seriously injured passenger
  - U_1 and U_2 are quite close
  - Distributions show that sattelite information might change utilities drastically (because very broad)
- (c)
  - Choosing between two dirt roads in summertime, when blockage is unlikely (Utilities pretty close)
  - Distributions show that additional information might quite likely change our plans
  - But: The difference in utility is likely to be very small, so it isn't worth the effort

In sum, information has value to the extent that it is likely to cause a change of plan and to the extent that the new plan will be significantly better than the old plan.

3.4.3 Implementation of an information-gathering agent

A sensible agent should ask questions in a reasonable order, should avoid asking questions that are irrelevant, should take into account the importance of each piece of information in relation to its cost, and should stop asking questions when that is appropriate.

We assume that with each observable evidence variable E_j, there is an associated cost, $Cost(E_j)$. We assume that the result of the action $Request(E_j)$ is that the next percept provides the value of E_j.

function INFORMATION-GATHERING-AGENT(*percept*) **returns** an *action*
  persistent: D, a decision network

  integrate *percept* into D
  $j \leftarrow$ the value that maximizes $VPI(E_j)$ / $Cost(E_j)$
  if $VPI(E_j) > Cost(E_j)$
    return REQUEST(E_j)
  else return the best action from D

Figure 16.9 Design of a simple information-gathering agent. The agent works by repeatedly selecting the observation with the highest information value, until the cost of the next observation is greater than its expected benefit.

Form of information gathering: **myopic**. The algorithm uses the VPI formula shortsightedly, calculating the value of information as if only a single evidence variable will be acquired. If there is no

single evidence variable that will help a lot, a myopic agent might hastily take an action when it would have been better to request two or more variables first and then take actions.

4 Temporal probability models (Probabilistic Reasoning Over Time)

4.1 States and observation

We view the world as a series of snapshots, or **times slices**, each of which contains a set of random variables, some observable and some not. We will use X_t to denote the set of state variables at time t, which are assumed to be unobservable, and E_t to denote the set of observable evidence variables. The observation time t is $E_t = e_t$ for some set of values e_t. The state sequence starts at $t = 0$.

Notation: $X_{a:b}$ = set of variables from X_a to X_b.

4.2 Transition and sensor models

Two questions:

- How does the world evolve? ($\rightarrow$ **transition model**)
- How do evidence variables get their values ($\rightarrow$ **sensor model**)

The **transition model** specifies the probability distribution over the last state variables given the previous values, that is, $P(X_t|X_{0:t-1})$.

- Problem: the set $X_{0:t-1}$ is unbound in the size as t increases
- Solution: **Markov assumption** - the current state depends only on a *finite fixed number* of previous states
- Processes satisfying this assumption are called **Markov processes** or **Markov chains**
- Another problem: infinetly many possible values of t. Do we need specify a different distribution for each time step?
- Another solution: We assume that changes in the world state are caused by a **stationary process**, which doesn't change over time (e.g. $P(R_t|R_{t-1})$ is the same $\forall t$)

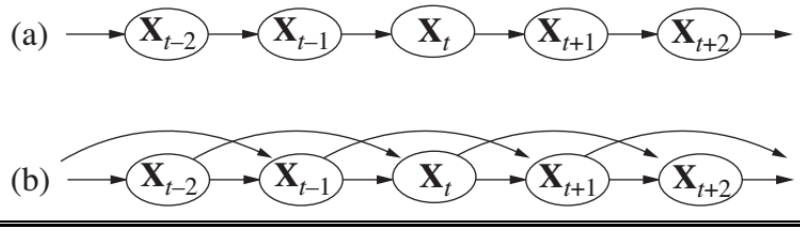


Figure 15.1 (a) Bayesian network structure corresponding to a first-order Markov process with state defined by the variables $\mathbf{X}_t$. (b) A second-order Markov process.

The **sensor model** specifies, how evidences are generated (on which state variables they depend). We make the **sensor Markov assumption**:

$$P(E_t|X_{0:t}, E_{0:t-1}) = P(E_t|X_t)$$

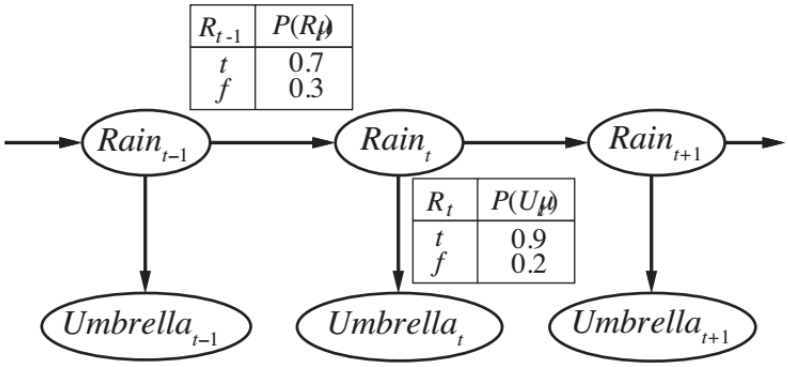


Figure 15.2 Bayesian network structure and conditional distributions describing the umbrella world. The transition model is $P(Rain_t \mid Rain_{t-1})$ and the sensor model is $P(Umbrella_t \mid Rain_t)$.

Additionally, we need to specify an intial state probability $P(X_0)$.

The complete joint probability distribution over all the variables is (for any t, with first order Markov assumption) given by

$$P(X_{0:t}, E_{1:t}) = P(X_0) \prod_{i=1}^{t} P(X_i|X_{i-1})P(E_i|X_i)$$

and consists of initial state probability, the transition model and the sensor model.

One problem with the first order Markov assumption is, that it might not model the real world apropriatly. Possible fixes:

- Increase order of Markov process
- Augment state, e.g. $Season_t$, $Temp_t$, $Pressure_t$

4.3 Inference in Temporal Models

Inference tasks:

- **Filtering:** computing the **belief state** - the posterior distribution over the most recent state - given all evidence to date. In our example, we wish to compute $P(X_t|e_{1:t})$. Filtering is what a rational agent does to keep track of the current state so that rational decisions can be made
- **Prediction:** computing the posterior distribution over the future state, given all evidence to date. That is, we wish to compute $P(X_{t+k}|e_{1:t})$ for some $k > 0$
- **Smoothing:** computing the posterior distribution over a past sate, given all evidence up to the present. That is, we wish to compute $P(X_k|e_{1:t})$ for some k such that $0 \leq k < t$. Smoothing provides a better esimate of the state than was available at the time, because it incorporates more evidence
- **Most likely explanation:** Given a sequence of observations, we might wish to find the sequence of states that is most likely to have generated those observations. That is, we wish to compute $argmax_{x:t} P(x_{1:t}|e_{1:t})$

4.3.1 Filtering and prediction

Given the filtering result up to time t, the agents needs to compute the result for $t+1$ from the new evidence e_{t+1},

$$P(X_{t+1}|e_{1:t+1}) = f(e_{t+1}, P(X_t|e_{1:t}))$$

for some function f. This process is called **recursive estimation**.

Two part process: first, the current state distribution is projected forward from $ttot + 1$; then it is updated using the new evidence e_{t+1}.

$$\begin{aligned} P(X_{t+1}|e_{1:t+1}) &= P(X_{t+1}|e_{1:t}, e_{t+1}) \\ &= \alpha P(e_{t+1}|X_{t+1}, e_{1:t})\ P(X_{t+1}|e_{1:t}) \\ &= \alpha P(e_{t+1}|X_{t+1})\ P(X_{t+1}|e_{1:t}) \end{aligned}$$

$P(e_{t+1}|X_{t+1})$ is given by the sensor model. We obtain the other term by conditioning on the current state X_t:

$$\begin{aligned} P(X_{t+1}|e_{1:t}) &= \sum_{x_t} P(X_{t+1}|x_t, e_{1:t}) P(x_t|e_{1:t}) \\ &= \sum_{x_t} P(X_{t+1}|x_t) P(x_t|e_{1:t}) \end{aligned}$$

so

$$P(X_{t+1}|e_{1:t+1}) = \alpha P(e_{t+1}|X_{t+1})\ \sum_{x_t} P(X_{t+1}|x_t) P(x_t|e_{1:t})$$

In the summation, the first factor comes from the transition model and the second comes from the current state distribution. Hence, we have the desired recursive formulation. We can think of the filtered estimate $P(X_t|e_{1:t})$ as a message $f_{1:t}$ that is propagated forward along the sequence, modified by each transition and updated by each new observation. The process is given by $f_{1:t+1} = \alpha FORWARD(f_{1:t}, e_{t+1})$.

Prediction is simply filtering without new evidence,

$$P(X_{t+k+1}|e_{1:t}) = \sum_{x_{t+k}} P(X_{t+k+1}|x_{t+k}) P(x_{t+k}|e_{1:t})$$

As $k \rightarrow \infty$ the predicted distribution tends to converge to a fixed point, after which it remains constant for all time. This is the stationay distribution of the Markov process defined by transition model.

4.3.2 Smoothing

We can again split the computation into two parts - the evidence up to k and the evidence from $k+1$ to t.

$$\begin{aligned} P(X_k|e_{1:t}) &= P(X_k|e_{1:k}, e_{k+1:t}) \\ &= \alpha P(X_k|e_{1:k}) P(e_{k+1:t}|X_k, e_1 : k) \\ &= \alpha P(X_j|e_{1:k}) P(e_{k+1:t}|X_k) \\ &= \alpha f_{1:k} \times b_{k+1:t} \end{aligned}$$

The backward message $b_{k+1:t}$ can be computed by a recursive process that runs *backward* from t.

$$\begin{aligned} P(e_{k+1:t}|X_k) &= \dots \\ &= \sum_{x_{k+1}} P(e_{k+1}|x_{k+1})P(e_{k+2:t}|x_{k+1})P(x_{k+1}|X_k) \end{aligned}$$

Using the message notation, we have $b_{k+1:t} = BACKWARD(b_{k+2:t}, e_{k+1})$.

function FORWARD-BACKWARD(**ev**, *prior*) **returns** a vector of probability distributions
  inputs: **ev**, a vector of evidence values for steps $1, \dots, t$
    prior, the prior distribution on the initial state, $\mathbf{P}(\mathbf{X}_0)$
  local variables: **fv**, a vector of forward messages for steps $0, \dots, t$
    b, a representation of the backward message, initially all 1s
    sv, a vector of smoothed estimates for steps $1, \dots, t$

  fv$[0] \leftarrow$ *prior*
  for $i = 1$ **to** t **do**
    fv$[i] \leftarrow$ FORWARD(**fv**$[i-1]$, **ev**$[i]$)
  for $i = t$ **downto** 1 **do**
    sv$[i] \leftarrow$ NORMALIZE(**fv**$[i] \times$ **b**)
    b $\leftarrow$ BACKWARD(**b**, **ev**$[i]$)
  return sv

Figure 15.4 The forward–backward algorithm for smoothing: computing posterior probabilities of a sequence of states given a sequence of observations. The FORWARD and BACKWARD operators are defined by Equations (15.5) and (15.9), respectively.

4.3.3 Finding the most likely sequence

...

4.4 HMMs and DBNs

...

5 Making Complex decisions

Here, we deal with **sequential decision problems**, where the agent's utility depends on a sequence of decisions.

5.1 Sequential Decision Problems

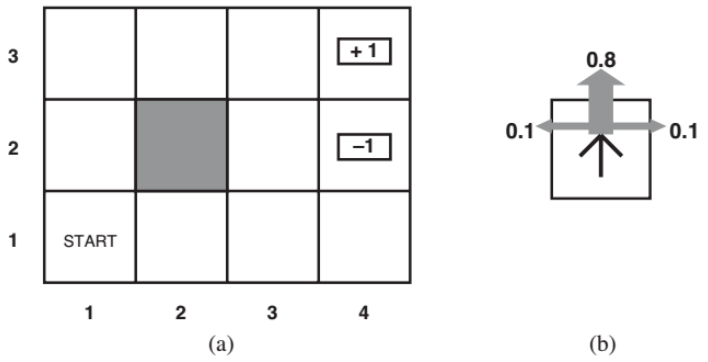


Figure 17.1 (a) A simple 4×3 environment that presents the agent with a sequential decision problem. (b) Illustration of the transition model of the environment: the "intended" outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. A collision with a wall results in no movement. The two terminal states have reward +1 and –1, respectively, and all other states have a reward of –0.04.

A sequential decision problem for a fully observable, stochastic environment with an Markovian transition model (the probability of reaching s' from s depends only on s and not on the history of earlier states) and additive rewards is called a **Markov decision process**, or **MDP**, and consists of

- a set of states (with an initial state s_0)
- a set $ACTIONS(s)$ in each state
- a transition model $P(s'|s, a)$
- and a reward function $R(s)$

Any fixed action sequence won't solve the problem, because the agent might end up in a state other than the goal. Therefore, a solution must specify what the agnt should do for *any* state that the agent might reach.

$\Rightarrow$ Policy π, where $\pi(s)$ is the action recommended by the policy π for state s.

Each time a given policy is executed starting from the initial state, the stochastic nature of the environment may lead to a different enviornment history. The quality of a policy is therefore measured by the expected utility of the possible environment histories generated by the policy. An **optimal policy** is a policy that yields the highest expected utility and is denoted as π^*.

The careful balancing of risk and reward is a characteristic of MDPs that does not arise in deterministic search problems.

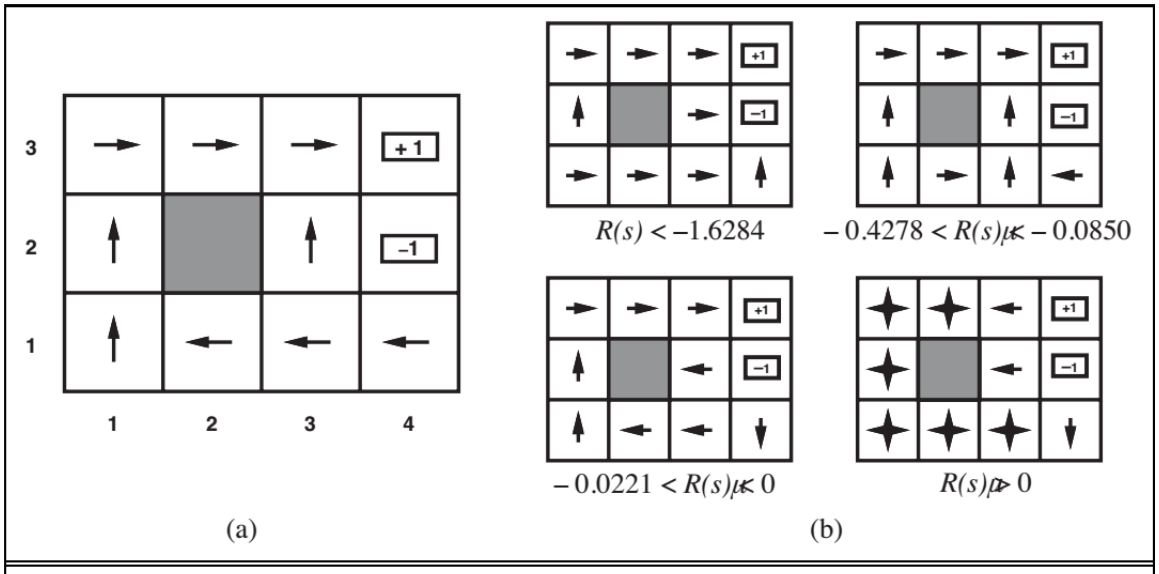


Figure 17.2 (a) An optimal policy for the stochastic environment with $R(s) = -0.04$ in the nonterminal states. (b) Optimal policies for four different ranges of $R(s)$.

5.1.1 Utilities over time

Utility function on environmental histories: $U_h([s_0, s_1, \ldots, s_n])$

Finite horizon: there is a fixed time N after which nothin matters - the game is over so to speak ($U_h([s_0, \ldots, s_{N+k}]) = U_h([s_0, \ldots, s_N])$). With a finite horizon, the optimal action in a given state could change over time (because for example, the time is running out and I have to take more risk in order to achieve a goal). We say that the optimal policy for a finite horizon is **nonstationary**.

Inifinte horizon: With no fixed time limit, there is no reason to behave differently in the same state at different times. Hence, the optimal action depends only on the current state, and the optimal policy is **stationary**.

We assume that preferences on state sequences are **stationary**,

$$([s, s_0, s_1, s_2 \ldots] \succ [s, s'_0, s'_1, s'_2, \ldots]) \Leftrightarrow ([s_0, s_1, s_3 \ldots] \succ [s'_0, s'_1, s'_2, \ldots])$$

"If you prefere one future to another starting tomorrow, then you should still prefer that future if it were to start today instead."

Under this assumption, there are only two coherent ways to assign utilities to sequences:

- **Additive rewards:** The utility of a state sequence is

$$U_h(s_0, s_1, s_2, \ldots) = R(s_0) + R(s_1) + R(s_2) + \ldots$$

- **Discounted reward:** The utility of a state sequence is

$$U_h(s_0, s_1, s_2, \ldots) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots$$

  where the **discount factor** γ is a number between 0 and 1. The discount factor decribes the preference of an agent for current rewards over future rewards ($\gamma = 1$: additive rewards).

When we choose an infinite horizon, environmental histories might be infinitely long, and utilities with undiscounted rewards might be infinte. Comparing state sequences with utility $+\infty$ is difficult. Three solutions:

1. With discounted rewards, the utility of an infinite sequence is finite. If $\gamma < 1$ and rewards are bounded by $\pm R_{max}$, we have

$$U_h([s_0, s_1, s_2, \ldots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = R_{max}/(1-\gamma)$$

2. If the environment contains terminal states and if the agent is guaranteed to get to one eventually (has **proper policy**), we will never need to compare infinite sequences
3. Infinite sequences can be compared in terms of the **average reward** per time step

5.1.2 Optimal policies and the utilities of states

We assume the agent is in some initial state s and define S_t to be the state the agent reaches at time t when executing a particular policy π.

The expected utility obtained by executing π starting in s is given by

$$U^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t R(S_t)]$$

where the expectation is with respect to the probability distribution over state sequences determined by s and π.

Optimal policies: $\pi_s^* = argmax_{\pi} U^{\pi}(s)$

When using discounted utilities with infinite horizons, the optimal policy is independent of the starting state.

The utility function $U(s)$ $(= U^{\pi^*(s)})$ allows the agent to choose the action that maximizes the expected utility of the subsequent state:

$$\pi^*(s) = argmax_{a \in A(s)} \sum_{s'} P(s'|s,a) U(s')$$

5.2 Value Iteration

The basic idea is to calculate the utility of each state and then use the state utilities to select an optimal action in each state.

5.2.1 The Bellman equation for utilities

The utility of a state is the immediate reward for that state plus the expected discounted utility for the next state, assuming that the agent chooses the optimal action. That is, the utility of a state is given by

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U(s')$$

called **Bellman equation**.

5.2.2 The value iteration algorithm

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    rewards $R(s)$, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states in S, initially zero
    δ, the maximum change in the utility of any state in an iteration

  repeat
    $U \leftarrow U'$; $\delta \leftarrow 0$
    for each state s **in** S **do**
      $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$
  return U

Figure 17.4 The value iteration algorithm for calculating utilities of states. The termination condition is from Equation (17.8).

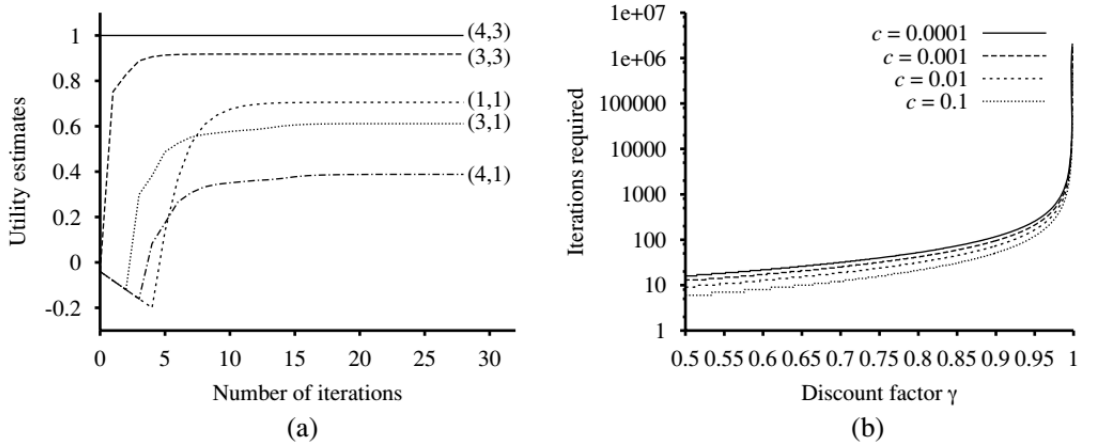


Figure 17.5 (a) Graph showing the evolution of the utilities of selected states using value iteration. (b) The number of value iterations k required to guarantee an error of at most $\epsilon = c \cdot R_{\max}$, for different values of c, as a function of the discount factor γ.

For n possible states, there are n Bellman equations, one for each state. The n (non-linear) equations contain n unknowns. If we apply the Bellman update infinetly often, we are guaranteed to reach an equilibrium, in which case the final utility values must be solutions to the Bellman equations. In fact, they are also the *unique* solutions, and the corresponding policy is optimal.

5.3 Policy Iteration

The policy iteration algorithm alternates the following two steps, beginning from some initilial policy π_0:

1. **Policy evaluation**: given a policy π_i, calculate $U_i = U^{\pi_i}$, the utility for each state if π_i were

to be executed

$$U_i(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) U_i(s')$$

2. **Policy improvement**: calculate a new MEU policy π_{i+1}, using one-step look-ahead based on U_i

$$\pi(s) = argmax_{a \in A(s)} \sum_{s'} P(s'|s, a) U(s')$$

The algorithm terminates when the policy improvement yields no change in the maximum expected utilities. The important point is that these equations are linear, because the max operator has been removed. For n states, we have n linear equations with n unknowns, which can be solved exactly in time $O(n^3)$.

5.4 Partially Observable MDPs

In an **partially observable** environment, the agent does not necessarily know which state it is in, so it cannot execute the action $\pi(s)$ recommended for that state. Furthermore, the utility of a state s and the optimal action in s depend not just on s, but also on how much the agent knows when it is in s. We cannot avoid POMDPs, because the real world is one.

5.4.1 Definition of POMDPs

A partially observable MDP is a MDP (transition model, actions, reward function) together with an **observation model** O that is stationary and has the sensor Markov property: $O(s, e) = P(e|s)$.

The optimal policy in an POMDP is a function $\pi(b)$ where b is the **belief state** (probability distribution over states). The optimal action depends only on the agent's current belief state, it does not depend on the actual state.

If $b(s)$ (the probability/belief degree that agent is in s) is the previous belief state and the agent does an action a and then perceives e, then the new belief state is

$$b'(s') = \alpha P(e|s') \sum_{s} P(s'|s, a) b(s)$$

POMDP decision cycle:

- Given the current belief state b, execute the action $a = \pi^*(b)$
- Receive percept e
- Update the current belief state and repeat

... Sehr viel exotischer Stuff. Think I'll skip this, sorry Mr. Prof. No hate.

6 Learning from Examples

An agent is **learning** if its improving its performance on future tasks after making observations about the world.

Reasons for Learning:

- The designer might not anticipate all possible situations that the agent might find itself in
- The designer can not anticipate all changes over time
- Sometimes human programmers have no idea how to program a solution themselves

6.1 Forms of Learning

Any component of an agent can be improved by learning from data. Improvements, and the techniques used to make them, depend on four major factors:

- Which component is to be improved
- What prior knowledge the agent already has
- What representation is used for the data and the component
- What feedback is available to learn from ((semi-)supervised, unsupervised or reinforcement)

6.2 Supervised Learning

Given a **training set** of N example input-output pairs

$$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$$

where each y_j was generated by an unknown function $y = f(x)$ discover a function h that approximates the true function f, called *hypothesis*. Learning is a search through the space of possible hypotheses for one that will perfom well, even on new examples beyond the training set. In this case we say the hypotesis **generalizes** well. Sometimes the function f is stochastic, why we have to learn a conditional probability $P(Y|x)$. In general, there is trade-off between complex hypotheses that fit the training data well and simpler hypotheses that may generalize better.

6.3 Learning Decision Trees

6.3.1 The decision tree representaion

For the moment we restrict ourselfs to a Boolean classification.

A decision tree reaches its decision by performing a sequence of tests. Each internal node corresponds to a test of the value of one of the input attributes, A_i, and the branches from the node are labled with the possible values of the attributes, $A_i = v_{ik}$.

6.3.2 Inducing decision trees from examples

Examples: pairs of (x, y), where x is a set of values for the input attributes and y is a single Boolean output value.

It is an intractable problem to find the smallest consistent tree. With some simple heuristic however, we can find a good approximate solution: a small, but not smallest consistent tree.

The Decision-Tree-Learning algorithm adopts a greedy divide-and-conquer strategy: always test the most important attribute first (which makes the most difference to the classification of an example). This test divides the problem up into smaller subproblems that can then be solved recursively.

Four cases for recursion:

1. If the remaining examples are all positive or negative, we are done.
2. If there are some positive and some negative examples, then choose the best attribute to split them
3. If there are no examples left, it means that no example has been observed for this combination of attribute values, and we return plurality classification of all the examples that were used in constructing the node's parant
4. If there are no attributes left, but both positive and negative examples, it means that these examples have exactly the same description, but different classifications → error or noise in data. Return plurality

function DECISION-TREE-LEARNING(*examples*, *attributes*, *parent_examples*) **returns** a tree

if *examples* is empty **then return** PLURALITY-VALUE(*parent_examples*)
else if all *examples* have the same classification **then return** the classification
else if *attributes* is empty **then return** PLURALITY-VALUE(*examples*)
else
  $A \leftarrow \text{argmax}_{a \in attributes}$ IMPORTANCE(a, *examples*)
  tree ← a new decision tree with root test A
  for each value v_k of A **do**
    $exs \leftarrow \{e : e \in examples \textbf{ and } e.A = v_k\}$
    subtree ← DECISION-TREE-LEARNING(exs, $attributes - A$, *examples*)
    add a branch to *tree* with label $(A = v_k)$ and subtree *subtree*
  return *tree*

Figure 18.5 The decision-tree learning algorithm. The function IMPORTANCE is described in Section 18.3.4. The function PLURALITY-VALUE selects the most common output value among a set of examples, breaking ties randomly.

6.3.3 Choosing attribute tests

A perfect attribute divides the examples into sets, each of which are all positive or all negative and thus will be leaves of the tree. We will use the notion of information gain, which is defined in terms of **entropy**.

Entropy is a measure of the uncertainty of a random variable; acquisition of information corresponds to reduction in entropy. In general, the entropy of a random variable V with values v_k, each with probability $P(v_k)$, is defined as

$$H(V) = \sum_k P(v_k) log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) log_2 P(v_k).$$

Entropy for boolean variable, which is true with probability q:

$$B(q) = -(q\ log_2(q) + (1-q) log_2(1-q))$$

If a training set contains p positive and n negative examples, then the entropy of the goal attribute on the whole set is

$$H(Goal) = B(\frac{p}{p+n})$$

An attribute A with d distinct values divides the training set E into $E_1, \ldots, E_d$. Each subset has p_k positive examples, so if we go along that branch, we will need an additional $B(p_k/(p_k + n_k))$ bits of information to answer the question. A randomly chosen example from the training set has the kth value for the attribute with probability $(p_k + n_k)/(p + n)$, so the expected entropy remaining after testing value A is

$$Remainder(A) = \sum_{k=1}^{d} \frac{p_k + n_k}{p+n} B(\frac{p_k}{p_k + n_k})$$

The **information gain** from the attribute test on A is the expected reduction in entropy:

$$Gain(A) = B(\frac{p}{p+n}) - Remainder(A)$$

6.3.4 Generalization and overfitting

Possible problem: **overfitting**

Solution: **decision tree pruning**

Look at test node that has only leaf nodes as descendant. If the test appears to be irrelevant-detecting only noise in the data-then we eliminiate the test, replacing it with a leaf node ($\rightarrow$ good indicator: information gain).

...

6.4 Evaluating and Choosing the Best Hypothesis

We want to learn a hypothesis that fits the **future data best**.

Future data: We make the **stationary** assumption over future data: that there is a probability distribution over examples that remain stationary over time. Each examples data point is a random variable E_j whose observed value ($e_j = (x_j, y_j)$ is sample from that distribution, and is independent of the previous examples:

$$P(E_j|E_{j-1}, \ldots) = P(E_j)$$

and each example has an identical prior probability distribution:

$$P(E_j) = P(E_{j-1}) = \ldots$$

(Identisch verteilte Zufallsvariablen bestizen alle dieselbe Verteilung, nehmen also mit gleicher Wahrscheinlichkeit gleiche Werte an, beinflussen sich aber nicht gegenseitig)

Best: We define the **error rate** of a hypothesis as the proportion of mistakes it makes - the proportion of times $h(x) \neq y$ for an example (x, y).

holdout cross-validation: Take one portion of the data for training, and the other portion for testing.

k-fold-cross-validation: We perfom k rounds of learning, where $\frac{1}{k}$ portion is used for testing.

leave-one-out-cross-validation: k = number of data samples

It is necessary to further take a part of your test set as a **validation set** to tune the parameters of your model, because doing so with your test set may make the hypothsis not genalize well.

6.4.1 Model selection: Complexity versus goodness of fit

Finding the best hypothesis are two tasks: **model selection** (define hypothsis space) and **optimization** (find best hypothesis within that space). Use the validation set for optimization.

6.4.2 From error rates to loss

- Absolute value loss: $L_1(y, \hat{y}) = |y - \hat{y}|$
- Squared error loss: $L_1(y, \hat{y}) = (y - \hat{y})^2$
- 0/1 loss: $L_{0/1}(y, \hat{y}) = 0 \; or \; 1$

Generalization Loss:

$$GenLoss_L(h) = \sum_{(x,y)\in\eta} L(y, h(x))P(x, y)$$

Beause $P(x, y)$ is not known, the learning agent can only estimate generalization loss with **empirical loss**:

$$EmpLoss_{L,E}(h) = \frac{1}{N} \sum_{(x,y)\in E} L(y, h(x))$$

6.5 The Theory of Learning

Computational learning theory: Any hypothesis that is seriously wrong will almost certainly be found out with high probability after a small number of examples, because it will make an incorrect prediction. Thus, any hypothesis that is consistent wit ha sufficiently large set of training examples is unlikely to be seriouly wrong: that is, it must be **probably approximately corret**.

Any learning algorithm that return hypotheses that are probably approximately corret is called a **PAC learning** algorithm.

Axioms for PAC learning:

- Stationarity: $P(E) = P(X, Y)$
- True function f is deterministic and a member of the hypothesis class $\mathcal{H}$ that is being considered

Error rate of a hyposthesis h is defined as

$$error(h) = GenLoss_{L_{0/1}}(h) = \sum_{x,y} L_{0/1}(y, h(x))P(x, y)$$

In other words, $error(h)$ is the probability that h missclassifies a new example.

A hypothesis h is **approximately correct** if $error(h) \leq \epsilon$, where ϵ is a small constant. The hypothesis space containing all the bad ones is called $\mathcal{H}_{bad}$.

The probability, that a "seriously wrong" hypothesis h_b (with $error(h_b) > \epsilon$) is consistent with the first N examples is

$$P(h_b \text{ agrees with } N \text{ examples}) \leq (1 - \epsilon)^N$$

and

$$P(\mathcal{H}_{bad} \text{ constains a consisten hypothesis}) \leq |\mathcal{H}_{bad}|(1 - \epsilon)^N \leq |\mathcal{H}|(1 - \epsilon)^N$$

How many samples do we need, in order to be safe?

We would like to reduce the probability of this event below some small numer δ:

$$|\mathcal{H}|(1-\epsilon)^N \leq \delta.$$

It follows that

$$N \geq \frac{1}{\epsilon}(ln\frac{1}{\delta} + ln|\mathcal{H}|)$$

Thus, if a learning algorithm return a hypothesis that is consistent with this many examples, then with probability at least $1-\delta$ it has error at most ϵ. N is called the **sample complexity** of the hypothesis space.

6.6 Regression and Classification with Linear Models

6.6.1 Univatiate linear regression

Goal: find $h_w(x) = w_1 x + w_0$ that best fits the data
Squared loss function: $Loss(h_w) = \sum_{j=1}^{N} L_2(y_j, h_w(x_j)) = \sum_{j=1}^{N}(y_j - (w_1 x_j + w_0)$
We would like to find $w^* = argmin_w Loss(h_w)$, which has a closed form solution.

If there is no closed form solution, we usually do a **gradient descent**:

1. $w \leftarrow$ any point in the parameters space
2. for each w_i in w do: $w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(w)$
3. Repeat 2 until convergence

6.6.2 Multivariate linear regression

...

6.6.3 Linear classifiers with a hard threshold

A decision boundary is a line (or surface) that separates two classes.
The decision function is given by $h_w(x) = 1\ if\ w \cdot x \geq 0$ and 0 otherwise.
Perceptron learning rule: $w_i \leftarrow \alpha \cdot (y - h_w(x)) \cdot x_i$

6.6.4 Linear Classification with logistic regression

Idea: softening the threshold function - approximating the hard threshold with a continous, differentiable function.

$$Logistic(z) = \frac{1}{1+e^{-z}}$$

and

$$h_w(x) = Logisitic(w \cdot x) = \frac{1}{1+e^{-w \cdot x}}$$

The process of fitting the weights of this model to minimize loss on a data set is called **logistic regression**.

6.7 Artificial Neural Networks

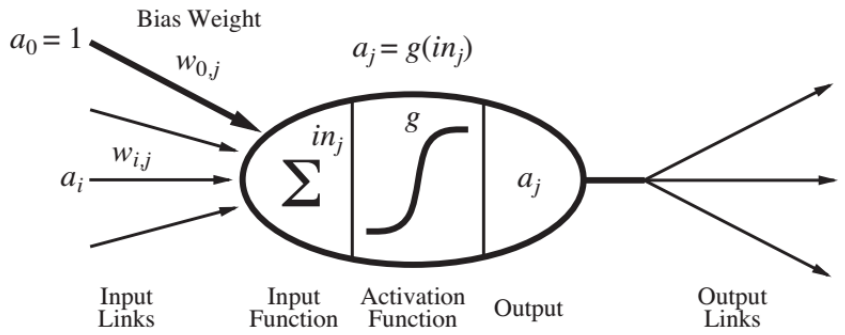


Figure 18.19 A simple mathematical model for a neuron. The unit's output activation is $a_j = g(\sum_{i=0}^{n} w_{i,j} a_i)$, where a_i is the output activation of unit i and $w_{i,j}$ is the weight on the link from unit i to this unit.

6.7.1 Neural network structures

- Neural networks are composed of nodes or **units** connected by directed **links**.
- A link from unit i to j serves to propagate the **activation** a_i from i to j
- Each link has a numeric weigth $w_{i,j}$ associated with it
- Each unit has a dummy input a_0 with an associated weight $w_{0,j}$ (**bias term**)
- Each unit j computes a weighted sum of its inputs:

$$in_j = \sum_{i=1}^{n} w_{i,j} a_i$$

- Then it applies an **activation function** g to this sum to derive

$$a_j = g(in_j) = g(\sum_{i=1}^{n} w_{i,j} a_i)$$

- feed-forward network form a directed acyclic graph, recurrent network feeds its outputs back into its own inputs

6.7.2 Single-layer feed-forward neural networks

- A perceptron with m outputs is really m seperate networks, because each weight affects only one of the outputs
- Single perceptron can not learn XOR, because not linearily seperable

6.7.3 Multilayer feed-forward neural networks

Question: How do we train such networks?

Think of the network as a function $h_w(x)$ parameterized by the weights w. As long as we can calculate the derivatives of this function with respect to the weights, we can use the gradient-descent loss-minimization method to train a network.

It is possible to represent any continous function of the inputs with arbitraty accuracy.

6.7.4 Learning in multilayer networks

For the L_2 loss, we have, for any weight w (because it is additive across the components of the error vector $y - h_w(x)$:

$$\frac{\partial}{\partial w} Loss(w) = \frac{\partial}{\partial w}|y - h_w(x)|^2 = \frac{\partial}{\partial w}\sum_k (y_k - a_k)^2 = \sum_k \frac{\partial}{\partial w}(y_k - a_k)^2$$

The major complication comes from the addition of hidden layers to the networks. Whereas the error $y - h_w$ at the output layer is clear, the error at the hidden layers seems mysterious because the training data do not say whay value the hidden nodes should have $\rightarrow$ **back-propagate** the error from the output layer to the hidden layers.

Update rule at output layer:

$$w_{j,k} \leftarrow w_{j,k} + \alpha \times a_j \times \Delta_k$$
$$\Delta_k = Err_k \times g'(in_k),\ Err_k = (y - h_w)_k$$

The idea is that hidden node j is "responsible" for some fraction of the error Δ_k in each of the output nodes to which it connects. Thus, the Δ_k values are divided according to the strength of the connection between the hidden node and the output node and a backpropagated to provide the $delta_j$ values for the hidden layer:

$$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta_j$$
$$\Delta_j = g'(in_j)\sum_k w_{j,k}\Delta_k$$

The backpropagation process can be summarized as follows:

- Compute the Δ values for the output units, using the observed error
- Starting with outer layer, repeat the following for each layer in the network, until the earliest hidden layer is reached:
  - Propagate the Δ values back to the previous layer
  - Update the weights between the two layers

6.8 Support Vector Machines

Properties:

1. SVMs construct a **maximum margin seperator** - a decision boundary with the largest possible distance to example point
2. SVMs can embed the data into a higher-dimensional space, using the so-called kernel-trick. The high-dimensional linear seperator is actually nonlinear in the original space

The seperator is deined as the set of points $\{x : w \cdot x + b = 0\}$. Solution can be found by solving

$$argmax_\alpha \sum_j \alpha_j - \frac{1}{2}\sum_{j,k}\alpha_j\alpha_k y_j y_k (x_j \cdot x_k)$$

subject to the constraints $\alpha_j \geq 0$ and $\sum_j \alpha_j y_j = 0$.
Our w then is $w = \sum_j \alpha_j x_j$ and $h(x) = sign(\sum_j \alpha_j y_j (x \cdot x_j) - b)$.

We see that the data enter the expression only in the form of dot products of pairs of points. We can use the **kernel trick**: A dot product in a higher dimensional space can be evaluated via a kernel function, without explicitly transforming the data into a higher dimensional space.

Mercers theorem tells us that any "reasonable" kernel function (matrix $K_{jk} = K(x_j, x_k)$ is positive definite) corresponds to some feature space.

7 Knowledge in Learning

In which we examine the problem of learning when you know something already

7.1 A Logical Formulation of Learning

Look at examples described by attribues, e.g. $Alternate, Bar, Fri/Set...$

Let us generically call the ith exmample X_i. We will use the notation $D_i(X_i)$ to refer to the descprition of X_i, where D_i can be any logical expression, e.g. $Alternatve(X_i) \wedge \neg Bar(X_i) \ldots$. The classification of the example is given by a literal using the goal predicate, e.g. $WillWait(X_i)$.

The aim of inductive learning in general is to find a hypothesis that classifies the examples well to new exmaples:

$$\forall Goal(x) \Leftrightarrow C_j(x)$$

where $C_j(x)$ is a candidate definition - some expression involving the attribute predicates.

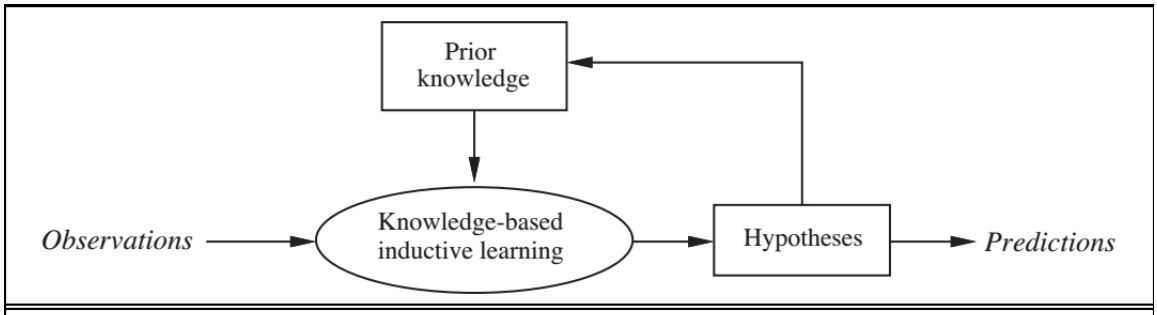


Figure 19.6 A cumulative learning process uses, and adds to, its stock of background knowledge over time.

7.2 Explanation-Based Learning

Is a method for extraction general rules from individual observations.

Example: We would like to be able to extract the general rule that for any arithmetic unknown u, the derivative of u^2 w.r.t u is $2u$:

$$ArithmeticUnknown(u) \Rightarrow Derivative(u^2, u) = 2u$$

7.2.1 Extracting general rules from examples

The basic idea is first to construct an explanation of the observation unsing prior knowledge, and then establish a definition of the class of cases for which the same explanation structure can be used.

... Ich check garnix, scheiss auf Logik!

8 Learning Probabilistic Models

8.1 Statistical Learning

Bayesian learning simply calculates the probability of each hypothesis, given the data, and makes predictions on that basis. Let D represent the data, with observed value d; then the probability of each hypothesis is obtained by Bayes' rule:

$$P(h_i|d) = \alpha P(d|h_i)P(h_i)$$

where the observations are assumed to be independent and identically distributed (i.i.d), and so

$$P(d|h_i) = \prod_j P(d_j|h_i)$$

where $p(h_i)$ is called the **hypothesis prior** and $P(d|h_i)$ the **likelihood** of the data under each hypothesis.

Now, suppose we want to make a prediction about an unknown quantity X. Then we have

$$P(X|d) = \sum_i P(X, h_i|d) = \sum_i (P(X|h_i, d) \cdot P(h_i|d) = \sum_i P(X|h_i)P(h_i|d)$$

where we assume that each hypothesis determies a probability over X.

Because of computational cost it is often infeasible to compute all the posterior probabilities for each hypotesis. A very common approximation is to make predictions based on a single *most probable* hypothesis - that is, an h_i that maximizes $P(h_i|d)$. This is often called a **maximum a posteriori** or MAP hypothesis with the assumption $P(X|d) \approx P(X|h_{MAP})$.

$$h_{MAP} = \operatorname*{argmax}_h P(h|d) = \operatorname*{argmax}_h \frac{p(d|h)p(h)}{p(d)} = \operatorname*{argmax}_h p(d|h)p(h)$$

Another simplification is provided by assuming **uniform** prior over the space of hypotheses. In that case, MAP learning reduces to choosing an h_i that maximized $P(d|h_i)$. This is called a **maximum-likelihood** (ML) hypothsis with

$$h_{ML} = argmax_h p(h|d)$$

8.2 Learning with Complete Data

The general task of learning a probability model, given data that are assumed to be generated from that model, is called **density estimation**. Here: **complete** data.

8.2.1 Maximum-likelihood parameters learning: Discrete models

We are given a dataset d of N candies. Under the hypothesis h_θ, the likelihood of this particular data set is

$$P(d|h_\theta) = \prod_{j=1}^{N} P(d_j|h_\theta)$$

The maximum-likelihood hypothesis is given by the value of θ that maximized this expression The same value is obtained by maximizing the **log-likelihood**,

$$L(d|h_\theta) = logP(d|h_\theta) = \sum_{j=1}^{N} logP(d_j|h_\theta)$$

We differentiate L with respect to θ and set the resulting expression to zero ($\frac{\partial L(d|h_\theta)}{\partial \theta} = 0$)

8.2.2 Naive Bayes models

The different features (attributes) $x_1, \ldots, x_n$ of an observation are assumed to be statistically independent:

$$P(C_k, x_1, x_2, \ldots, x_n) = p(C_k) \prod_{i=1}^{n} p(x_i|C_k)$$

and

$$P(C_k|x_1, x_2, \ldots, x_n) = \frac{p(C_k) \prod_{i=1}^{n} p(x_i|C_k)}{P(x_1, \ldots, x_n} = \alpha \; p(C_k) \prod_{i=1}^{n} p(x_i|C_k)$$

A deterministic prediction can be obtained by choosing the most likely class.