CS50: Introduction to Artificial Inteligence Notes

Noah Peters

May 2, 2023

Abstract

CS50: Introduction to Artificial Inteligence notes. Note template by Pingbang Hu.

Contents

Chapter 1

Knowledge

1.1 Propositional Logic

We use standard logic notation:

- $\neg p$
- $p \vee q$
- $p \wedge q$
- $p \Rightarrow q$:

p	q	$p \Rightarrow q$
false	false	true
false	true	true
true	false	false
true	true	true

- $p \Leftrightarrow q$:

p	q	$p \Leftrightarrow q$
false	false	true
false	true	true
true	false	false
true	true	true

Now we must establish *what* is considered to be "true" in our world by defining a **model**. We need to represent that knowledge. We do so by defining it via a **knowledge base**.

Definition 1.1.1 (Model)**.** Assignnment of a truth value to every propositional symbol.

Definition 1.1.2 (Knowledge Base)**.** A set of sentences known by a knowledge-based agent.

Definition 1.1.3 (Enatilment)**.**

$$\alpha \models \beta \text{ "}\alpha \textit{ entails } \beta\text{"}$$

In every model in which sentence α is true, sentence β is also true.

1.2 Inference

Our aim is to see if our knowledge base, KB, entails some query about the world, α:

$$KB \models \alpha?$$

We first define a **model checking algorithm** to determinne if $KB \models \alpha$. We can deermine this by doing the following:

- enumerate all possible models
- if in every model where KB is true, α is also true, then $KB \models \alpha$

1.3 Inference By Resolution

To determine if $KB \models \alpha$ via knowledge resolution:

- Check if $KB \wedge \neg\alpha$ is a contradiction
  - Conver $KB \wedge \neg\alpha$ tto Conjunctive Normal Form
  - Keep checking to see if we can use resolution to produce new clause
  - If we ever produce the empty clause (equivalen to False), we have a contradiction and so $KB \models \alpha$
- If so, then $KB \models \alpha$
- Otherwise, no entailment

Problem 1.3.1. Does $(A \vee B) \wedge (\neg B \vee C) \wedge (\neg C)$ entail A ?

Answer. First, we convert to CNF:

$$(A \vee B) \wedge (\neg B \vee C) \wedge (\neg C) \wedge (\neg A)$$

We can resolve $(\neg B \vee C)$ and $(\neg C)$ by concluding that $\neg B$. With the knowledge of $\neg B$ we now see that, considering $A \vee B$, we can conclude A. We see that

$$A \wedge \neg A \Rightarrow False$$

and so we can conclude that the clause entails A. ⊛

Chapter 2

Probability

2.1 Introduction to Probability

We can represent a **possible world** using ω where all possible worlds is the set Ω . Thus, we can define total probability as:

$$\sum_{\omega \in \Omega} p(\omega).$$

We build machine models to predict outcomes based on data using **conditional probability**. In other words, we have probability based on some evidence. We can calculate conditional probability using the following formula:

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

2.2 Random Variables

In probability we define a variable with a set of possible values as a **random variable**:

Definition 2.2.1 (Random Variable)**.** A variable possessing a distribution of probabilities for various "states".

2.3 Baye's Rule

From the above equation we have:

$$P(a \wedge b) = P(b)P(a|b) = P(a)P(b|a)$$

Definition 2.3.1 (Independence)**.** The knowledge that one event occurs does not affect the probability of the other event. So we have

$$P(a \wedge b) = P(a)P(b)$$

since $P(b) = P(b|a)$ if a and b are independent.

Thus, we can derive Baye's rule, which relates the probability of one event on the condition of another event, to the reverse relationship.

Definition 2.3.2 (Baye's Rule)**.**

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

What Baye's rule allows us to do is that given:

$$P(\text{visible effect} \mid \text{unknown cause})$$

we can calculate

$$P(\text{unknown cause} \mid \text{visible effect}).$$

2.4 Bayesian Networks

There are a number of different probabilistic models. The first we discuss are **Bayesian Networks**.

Definition 2.4.1 (Bayesian Network). A data structure that represents the dependencies among random variables. They have the following characteristics:

- directed graph
- each node represents a random variable
- arrow from X to Y means X is a parent of Y
- each node X has probability distribution $P(X|\text{Parents}(X))$

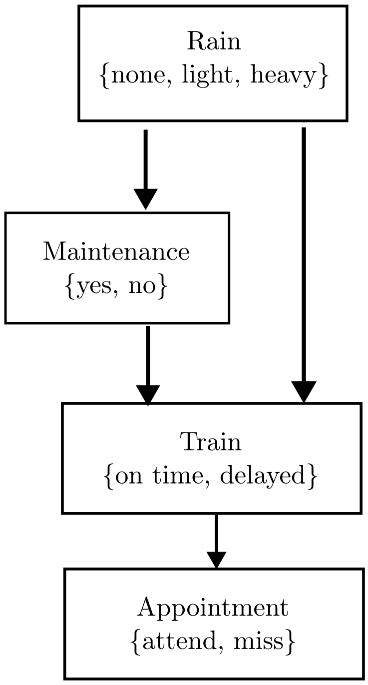

Figure 2.1: A basic example of a Bayesian Network.

We now aim to make an *inference* using the Bayesian Network. Given the following:

- Query X: variable for which to compute distribution
- Evidence variables E: observed variables for event e
- Hidden variables Y: non-evidence, non-query variable

our goal is to calculate $P(X|e)$.

Problem 2.4.1. Calculate $P(\text{Appointment}|\text{light, no})$

Answer. Here, the *evidence* is that there is light rain and no maintenance. The *query* is the status of Appointment. The *hidden layer* is the status of the train, since you are not given the train's status and you are not querying it; it's just a confounding variable.

We note that

$$P(\text{Appointment}|\text{light, no}) = \alpha P(\text{Appointment, light, no})$$

and by the marginalization technique:

$$= \alpha[P(\text{Appointment, light, no, on time}) + P(\text{Appointment, light, no, delayed})].$$

⊛

2.5 Inference by Enumeration

The above is an example of **inference by enumeration**. More formally, for the following:

- X: the query variable
- e: the evidence
- y: ranges over values of hidden variables
- α: normalizes the result

we have:

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

2.6 Sampling

Rather than attempting to calculate an exact probability, we can approximate a probability instead via **sampling**. By randomly generating samples for $n = 1000$ or $n = 10000$, we can get fairly useful results.

Definition 2.6.1 (Rejection Sampling). **Rejection sampling** is the process of simulating numerous examples from a distribution while considering only the samples that possess the attributes of the desired query.

Remark. Rejection sampling is not particularly effective when the evidence you are looking for is fairly unlikely, since you are rejecting a lot of samples. This is inefficient since you are throwing away a large portion of your samples.

Definition 2.6.2 (Likelihood Weighting). Rather than sampling everything, in **likelihood weighting**, we start by *fixing* the values for evidence variables. We then sample the non-evidence variables using conditional probabilities in the Bayesian Network. Finally we weight each sample by its *likelihood*.

2.7 Markov Models

As opposed to assigning one random variable to a value, but rather an array of random variables to a value over a timescale. This leads to a lot more data, and so we must make some assumptions.

Definition 2.7.1 (Markov Assumption). The assumption that the current state depends only on a finite fixed number of previous states.

Definition 2.7.2 (Markov Chain). A sequence of random variables where the distribution of each variable follows the Markov assumption.

For example, consider the basic Markov Chain

$$X_t \to X_{t+1}$$

where X_t represents the weather today, and X_{t+1} represents the weather tomorrow. We can construct a **transition model** to describe the relationship as follows:

	X_{t+1}: Sunny	X_{t+1}: Rainy
X_t: Sunny	0.8	0.2
X_t: Rainy	0.3	0.7

We can refine our definition of the Markov Model further, by introducing *hidden* states.

Definition 2.7.3 (Hidden Markov Model)**.** A Markov model for a system with hidden states that generate some observed event.

Definition 2.7.4 (Filtering)**.** Given observations from start until now, calculate distribution for current state.

Definition 2.7.5 (Prediction)**.** Given observations from start until now, calculate the distribution for a future state.

Definition 2.7.6 (Smoothing)**.** Given observations from start until now, calculate the distribution for a past state.

Definition 2.7.7 (Most Likely Explanation)**.** Given observations from start until now, calculate most likely sequence of states.

Chapter 3

Optimization

3.1 Hill Climbing

The most basic form of optimization we explore is that of the local search.

Definition 3.1.1 (Local Search)**.** Search algorithms that maintain a single node and searches by moving to a neighboring node.

We can implement a local searching using **hill climbing** in which we do the following:

- Check values of neighbors of current best value
- If a neighbor has a value closer to the desired value, set that value as current best

Algorithm 3.1: HillClimb

```
Data: problem
cur = initial state of problem
repeat
    neighbor = best val neighbor of cur
    if neighbor not better than current then
        return cur
    cur = neighbor
until better neighbor does not exist
```

The problem with this naive approach (**steepest ascent**) is that we may get stuck at a local extremum rather than the global extremum. Further, we may encounter a "shoulder", where several neighboring values are the same, and get stuck.

3.2 Simulated Annealing

Although there are a variety of hill climbing algorithms (stochastic, first-choice, beam, etc.). They have their flaws but the theme is that we never go from a good value to a worse value. Thus, we must overcome local extrema. We can tackle this with **simulated annealing**.

Definition 3.2.1 (Simulated Annealing)**.** Simulated annealing is akin to a "cooling" physical process.

- Early on, higher "temperature": more likely to accept neighbors that are worse than current state
- Later on, lower "temperature": less likely to accept neighbors that are worse than current state

Algorithm 3.2: SimulatedAnnealing

Data: problem, max

cur = initial state of problem
for $t = 1$ *to max* **do**
  T = Temperature(t)
  neighbor = random neighbor of cur
  ΔE = how much better neighbor is than cur
  if $\Delta E > 0$ **then**
    cur = neighbor
  with probability $e^{\Delta E / T}$, set cur = neighbor

3.3 Linear Programming

In the context where we are trying to optimize for some function or when we have real values we are often trying to minimize or maximize a cost function given a variety of constraints. This is where **linear programming** becomes useful.

Definition 3.3.1. Linear programming problems often entail the following:

- Minimize a cost function $c_1x_1 + c_2x_2 + \ldots + c_nx_n$
- With constraints of form $a_1x_1 + a_2x_2 + \ldots + a_nx_n \leq b$ or of form $a_1x_1 + a_2x_2 + \ldots + a_nx_n = b$
- With bounds for each variable $l_i \leq x_i \leq u_i$

Appendix