Module 2: Classical AI

M. Vidyasagar

Distinguished Professor, IIT Hyderabad
Email: m.vidyasagar@iith.ac.in
Website: www.iith.ac.in/∼m_vidyasagar/

Outline

Outline

Role of Mathematical Logic in Classical AI

- "Classical" AI as practiced between 1955 to 1980 (roughly) was mostly aimed at programming machines to mimic human reasoning.
- Mathematical logic was employed in order to ensure that humans could "trust" the output of such reasoning machines.
- In this lecture we review some aspects of mathematical logic.
- The most basic is predicate logic, followed by propositional logic, and then "nonstandard" logics.

Outline

IIT Hyderabad

Boolean Variables

- In predicate logic, every variable is **Boolean**, in that it has only two values: True (T) or False (F); these could also be denoted by 1 and 0 respectively.
- There are also binary Boolean **operations**, such as $\wedge$ (and), $\vee$ (or), $\implies$ (implies).
- All binary operations can be expressed in terms of their truth tables (given on the next slide).
- Not all symbols are necessary; some can be expressed in terms of others. For example, $A \implies B$ is the same as $\neg A \vee B$, where $\neg$ is negation.

Truth Tables of Binary Boolean Operations

Function	$A \wedge B$			$A \vee B$			$A \implies B$	
$A \setminus B$	T	F	$A \setminus B$	T	F	$A \setminus B$	T	F
T	T	F	T	T	T	T	T	F
F	F	F	F	T	F	F	T	T

Some useful identities (distribution and negation laws):

$$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C),$$

$$A \vee (B \wedge C) = (A \vee C) \wedge (A \vee C).$$

$$\neg(A \wedge B) = \neg A \vee \neg B, \neg(A \vee B) = \neg A \wedge \neg B.$$

Using Predicate Logic to Test Tentative Conclusions

In using predicate logic to reason, we have a set of "clauses," which are expressions involving some Boolean variables $X_1, \ldots, X_n$, call them $\phi_1, \ldots, \phi_m$, and a "desired conclusion" which consists of another clause θ.

The objective is to determine whether the clauses together imply the desired conclusion.

In symbols, we wish to know whether

$$(\phi_1 \wedge \ldots \wedge \phi_m) \implies \theta.$$

Tautologies and Contradictions

A **tautology** is a statement that is always true. An example is $X \vee \neg X$.

A **contradiction** is a statement that is always false. An example is $X \wedge \neg X$.

Predicate logic is *refutation complete*. So the statement

$$(\phi_1 \wedge \ldots \wedge \phi_m) \implies \theta.$$

is true if and only if the statement

$$(\phi_1 \wedge \ldots \wedge \phi_m) \wedge \neg \theta.$$

is a contradiction.

A Toy Example

Two clauses: (i) If a patient has high fever, give Crocin, and (ii) patient has high fever. Desired conclusion: Give Crocin.

Define $A =$ patient has high fever, and $B =$ patient should be given Crocin. So the two clauses are $\phi_1 = (A \implies B)$, or equivalently $\phi_1 = \neg A \vee B$, and $\phi_2 = A$. The desired conclusion is $\theta = B$. Does θ follow from ϕ_1 and ϕ_2?

A Toy Example (Con'td)

The question is:

$$(\neg A \vee B) \wedge A \implies B?$$

Use the refutation argument. Is

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

a contradiction?

We could enumerate all four possible truth values of (A, B) and check that the statement is always false. But that approach does not "scale". With n Boolean variables, we would have to check 2^n combinations. Is there an easier way?

A Toy Example (Con'td)

Use the distribution law

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

is equivalent to

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

which is in turn equivalent to

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

The first clause contains $\neg A \wedge A$ while the second contains $B \wedge \neg B$. So both clauses are always false, so it is a contradiction. Hence the original implication is true.

Outline

The Satisfiability Problem

Suppose $\phi_1, \ldots, \phi_m$ are clauses in Boolean variables $X_1, \ldots, X_n$.

The **satisfiability problem (SAT)** is to determine whether there is a truth assignment of the variables that makes *each* of the clauses to be true.

Examples:

$$\phi_1 = X_1 \vee \neg X_2, \phi_2 = X_2 \vee \neg X_3, \phi_3 = \neg X_2 \vee \neg X_3.$$

Can all these clauses be satisfied? Yes.

$$\phi_1 = X_1 \vee X_2 \vee \neg X_4, \phi_2 = \neg X_1 \vee X_2 \vee \neg X_4, \phi_3 = X_2 \vee X_4, \phi_4 = \neg X_2$$

Can all these clauses be satisfied? No.

Relation to Machine Reasoning

We have seen that the question:

$$\phi_1 \wedge \ldots \wedge \phi_m \implies \theta?$$

is equivalent to: Is the set of clauses

$$\phi_1 \wedge \ldots \wedge \phi_m \wedge \neg\theta$$

satisfiable?

Conjunctive Normal Form (CNF)

If each clause ϕ_i is a *disjunction* of variables or their negations, this is called a **conjunctive normal form (CNF)**.

The previous example

$$\phi_1 = X_1 \vee \neg X_2, \phi_2 = X_2 \vee \neg X_3, \phi_3 = \neg X_2 \vee \neg X_3.$$

is in CNF. It is satisfied by (for example) $X_1 = T$, $X_2 = T$, $X_3 = F$.

The CNF with

$$\phi_1 = X_1 \vee X_2 \vee \neg X_4, \phi_2 = \neg X_1 \vee X_2 \vee \neg X_4, \phi_3 = X_2 \vee X_4, \phi_4 = \neg X_2$$

is *not* satisfiable.

NP-Completeness of Satisfiability

We could always enumerate all 2^n truth assignments and check whether some assignment satisfies all clauses. Is there a better way?

In general, no! The Satisfiability Problem (SAT) is **NP-complete**.

1. Given a candidate answer, it is easy to check whether it really is an answer (NP), but finding an answer from scratch is difficult.
2. Moreover, among all NP problems, SAT is among the hardest.

Are We at a Dead End?

We have seen that

1. Checking whether a given set of clauses implies a desired conclusion is a SAT problem.
2. In general checking SAT is hard.

So what is to be done?

Are there *some* instances of SAT that are not so difficult?

We study the linear programming relaxation of SAT.

Outline

IIT Hyderabad

Converting SAT into an Integer Linear Program

With the *Boolean* variables $X_1, \ldots, X_n$, associate *binary* variables $x_1, \ldots, x_n$. X_i is T or F, $x_i = 0$ or 1, with $x_i = 1$ denoting $X_i = T$ and $x_i = 0$ denoting $X_i = F$. Then $\neg X_i$ corresponds to $(1 - x_i)$.

Convert each clause to a linear inequality. For example

$$\phi_1 = X_1 \vee X_2 \vee \neg X_4$$

becomes

$$x_1 + x_2 + (1 - x_4) \geq 1.$$

In this way SAT becomes a set of m linear inequalities.

An Example

The earlier example

$$\phi_1 = X_1 \vee X_2 \vee \neg X_4, \phi_2 = \neg X_1 \vee X_2 \vee \neg X_4, \phi_3 = X_2 \vee X_4, \phi_4 = \neg X_2$$

becomes the set of four inequalities

$$x_1 + x_2 + (1 - x_4) \geq 1, (1 - x_1) + x_2 + (1 - x_4) \geq 1, x_2 + x_4 \geq 1, (1 - x_2) \geq 1.$$

Check whether any *binary* variables $x_1, \ldots, x_4$ satisfy these inequalities.

The answer is: No. But to get this answer, in general we have to enumerate all 2^n combinations.

Converting SAT into an Linear Program

Testing whether the set of m linear inequalities is satisfied by n *binary* variables $x_1, \ldots, x_n$ is again NP-complete. So we replace the condition $x_i = 0$ or 1 by $0 \leq x_i \leq 1$.

1. It easy to check whether this set of inequalities has a solution.
2. The solution may not consist of binary values for the x_i.
3. However, if the solution has all binary values, then the original SAT problem has a solution.

When do ILP and LP have the same solutions?

Horn Clauses

A **Horn** clause is a clause in which at most one variables does not have a $\neg$ in front of it. Example: Suppose

$$\phi_1 = \neg X_1 \vee X_2 \vee \neg X_3, \phi_2 = X_1 \vee X_2 \vee \neg X_3.$$

Then ϕ_1 is a Horn clause but ϕ_2 is not a Horn clause.

Important Fact: If a set of Horn clauses is converted into a linear program (LP), then every solution of the LP (if any) is also a solution of the original SAT problem.

Interpretation of Horn Clauses

Take two medical rules:

1. If a patient has high fever and vomiting and diarrhea, then s/he has cholera.
2. If a patient has high fever but no vomiting and no diarrhea, then s/he does not have cholera.

How do we represent these rules?

Interpretation of Horn Clauses (Cont'd)

Let $F =$ high fever, $V =$ vomiting, $D =$ diarrhea, and $C =$ cholera. Then (remembering that $A \implies B$ is equivalent to $\neg A \vee B$, we can write

$$\phi_1 = (F \wedge V \wedge D) \implies C, \text{ or } \phi_1 = \neg F \vee \neg V \vee \neg D \vee C.$$

$$\phi_2 = (F \wedge \neg V \wedge \neg D) \implies C, \text{ or } \phi_2 = \neg F \vee V \vee D \vee \neg C.$$

The first one is a Horn clause, the second one is not.

In a Horn clause, the *presence* of a symptom is informative, while the *absence* of a symptom is not.

Outline

Forward and Backward Chaining

Expert systems combined rules, facts, and logic to arrive at conclusions.

- In forward chaining, one starts with observations (symptoms) and proceeds towards conclusions (diagnoses).
- It is very useful in analyzing cascading failures caused by a single failure. In backward chaining, one starts with the conclusions, an identifies what set of symptoms can lead to that conclusion.
- It is very useful in identifying all possible causes of a failure or a disease.

For more details, see this Link

Desirable Attributes of Rule-Based Expert Systems

Two desirable attributes of automated reasoning:

- **Consistency:** Any conclusion reached by the computer must be a logical consequence of the facts. This is essential to give confidence to the user.
- **Completeness:** The computer must discover *every* logical consequence of the facts.
    - A fault-diagnosis expert system must identify *every* fault that could have caused the failures.
    - A medical diagnosis expert system must identify *every* disease that could be causing the symptoms.

Consistency and Completeness of Rule-Based Expert Systems

- If reasoning is based on predicate logic, it is *always* consistent – *all* conclusions are logical consequences of the facts.
- If the rules consist of Horn clauses, then both forward chaining and backward chaining are *complete*.
  - In forward chaining, *all* logical consequences of the facts will be found.
  - In backward chaining, *all* facts that could have led to the conclusions are found.
- If the rules do not consist of Horn clauses, in general neither forward chaining nor backward chaining is complete.

For more details, see this Link

Outline

IIT Hyderabad

The Need for Probabilistic Reasoning

A rule such as: "If a patient has high fever and vomiting and diarrhea, then s/he has cholera" is not very realistic.

A more realistic rule would be: "If a patient has high fever and vomiting and diarrhea, then s/he has cholera with probability 70%."

So we need a way of reasoning *under uncertainty*.

One such approach is indicated here. It can sometimes lead to counter-intuitive results.

A Linear Programming Approach

We study the following toy example. Suppose $A \implies B$ with probability 0.6, and A is true with probability 0.8. What is the probability that B is true?

A naive answer is $0.6 \times 0.8 = 0.48$. But this is not correct. We need to study a linear programming problem instead.

Setting up the Linear Programming Problem

Set up the following table of probabilities.

$A \setminus B$	T	F
T	p_{11}	p_{12}
F	p_{21}	p_{22}

So p_{11} is the probability that A is true and also B is true, and so on. The data consists of two statements

$$p_{11} + p_{21} + p_{22} = 0.6, p_{11} + p_{12} = 0.8.$$

We seek to find the maximum and the minimum values of $p_{11} + p_{21}$ subject to these constraints.

Solving the Linear Programming Problem

A problem of the type

$$\min_{\mathbf{x}} \mathbf{f}^\top \mathbf{x} \text{ s.t. } A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}$$

is called a **linear programming problem**.

Equality constraints of the form $A\mathbf{x} = \mathbf{b}$ are handled by setting

$$A\mathbf{x} \leq \mathbf{b}, -A\mathbf{x} \leq -\mathbf{b},$$

or equivalently

$$\left[\begin{array}{r} A \\ -A \end{array} \right] \mathbf{x} \leq \left[\begin{array}{r} \mathbf{b} \\ -\mathbf{b} \end{array} \right].$$

Solution to Example

The solution can be found using the `Matlab` command `linprog`.
▸ Diary

The solution for the maximum and minimum are, respectively

$$[p_{11} \;\; p_{12} \;\; p_{21} \;\; p_{22}] = [0.4 \;\; 0.4 \;\; 0 \;\; 0.2]^\top, \text{ and } [0.4 \;\; 0.4 \;\; 0.2 \;\; 0]^\top.$$

So $0.4 \leq \Pr(B) \leq 0.6$.

The lower bound is lower than $0.6 \times 0.8 = 0.48$. This is counter-intuitive.

Outline

DENDRAL

DENDRAL was probably the first rule-based expert system, whose function was to predict the structure of organic chemical compounds. ▸ Link

One more ▸ Link

MYCIN

"MYCIN was an early backward chaining expert system that used artificial intelligence to identify bacteria causing severe infections, such as bacteremia and meningitis, and to recommend antibiotics, with the dosage adjusted for patient's body weight." Link

MYCIN incorporated probabilistic reasoning, though in somewhat simple form.

Outline

Outline

Directed Acyclic Graphs (DAGs)

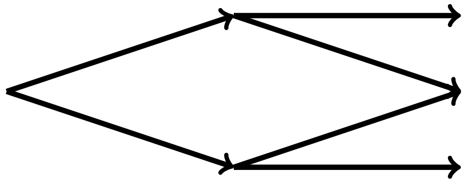
A **graph** consists of a set of vertices $\mathcal{V} = \{V_1, \ldots, V_n\}$ and a set of edges $\mathcal{E}$ where each edge is of the form (V_i, V_j). In this case we say that the edge is *from* vertex V_i *to* vertex V_j.

If, for every edge (V_i, V_j), there is also an edge (V_j, V_i), then the graph is said to be **undirected**. Otherwise it is said to be **directed**.

A **path** is a sequence of edges such that the second vertex of one edge is the first vertex of the next. A **cycle** is a path from a vertex to itself.

A **Directed Acyclic Graph (DAG)** is a directed graph in which there are no cycles.

Example of a DAG

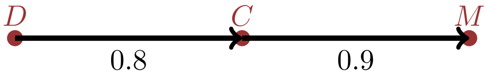


In every DAG, there is at least one **source** that does not have any incoming edges, and at least one **sink** that does not have any outgoing edges.

Bayesian Networks

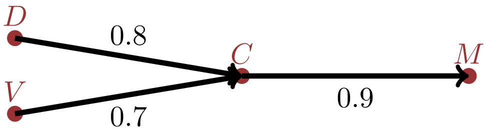
A **Bayesian network** is a DAG in which the vertices represent random variables. The edges have weights that denote probabilities.

Example: We may have two rules: (1) If a patient has diarrhea, then *with probability* 0.8 s/he has cholera. (2) If a patient has cholera, then *with probability* 0.9 the right medicine to administer is M. This is represented as shown below:



Independence Assumption

It is assumed that in a path of length two or more, the probabilities can simply be multiplied. This is called the "independence assumption."


$$\begin{tikzcd} D \arrow[dr, "0.8"] & & \\ & C \arrow[r, "0.9"] & M \\ V \arrow[ur, "0.7"'] & & \end{tikzcd}$$

If a patient comes in with diarrhea, we administer M with probability $0.8 \times 0.9 = 0.72$. If a patient comes in with vomiting, we administer M with probability $0.7 \times 0.9 = 0.63$

Note: The probability of administering M for suspected cholera (C) does not depend on the probable cause of diagnosing cholera.

Computing Probabilities in a Bayesian Networks

- Start at all source vertices, and assign probabilities to them.
- Add probabilities when two edges arrive at the same vertex.

This procedure assigns a probability to each node in the network and is quite efficient. ▸ Examples

Outline

Dempster-Shafer Theory

It is a "theory of belief" (also called "theory of evidence") where the weights of the beliefs need not add up to one are thus not probabilities.

"Real" probabilistic rule: If a patient has high fever, then s/he has malaria with probability 0.5, cholera with probability 0.4, and ...

These probabilities are determined using *statistics* and add up to one.

But a doctor might say: "If I see a patient with high fever, then I am 80% sure s/he has malaria, 70% sure that s/he has cholera, ..."

Dempster-Shafer theory gives a systematic method for pooling this kind of reasoning and arriving at conclusions. ▸ Link