CSC 411: Artificial Intelligence (3 Units: LH 45)

COURSE CONTENT

- Introduction to artificial intelligence,
- Understanding natural languages,
- Knowledge representation,
- Expert systems,
- Pattern recognition,
- The language PROLOG.

UNIT 1: INTRODUCTION TO ARTIFICIAL INTELLIGENCE

Definition of AI

What is Artificial Intelligence?

Artificial Intelligence is a branch of *Science* which deals with helping machines find solutions to complex problems in a more human-like fashion. This generally involves borrowing characteristics from human intelligence, and applying them as algorithms in a computer friendly way. A more or less flexible or efficient approach can be taken depending on the requirements established, which influences how artificial the intelligent behaviour appears.

AI is generally associated with *Computer Science*, but it has many important links with other fields such as *Mathematics*, *Psychology*, *Cognition*, *Biology* and *Philosophy*, among many others. Our ability to combine knowledge from all these fields will ultimately benefit our progress in the quest of creating an intelligent artificial being

It is also concerned with the design of intelligence in an artificial device. The term was coined by McCarthy in 1956. There are two ideas in the definition.

1. Intelligence
2. Artificial device

What is intelligence?

Is it that which characterize humans? Or is there an absolutestandard of judgment? Accordingly there are two possibilities:

- A system with intelligence is expected to behave as intelligently as a human
- A system with intelligence is expected to behave in the bestpossible manner

Secondly what type of behavior are we talking about?

- Are we looking at the thought process or reasoning ability of thesystem?
- Or are we only interested in the final manifestations of the systemin terms of its actions?

Given this scenario different interpretations have been used by different researchers as defining the scope and view of Artificial Intelligence.

1. One view is that artificial intelligence is about designing systemsthat are as intelligent as humans. This view involves trying to

understand human thought and an effort to build machines that emulate the human thought process. This view is the cognitive science approach to AI.

2. The second approach is best embodied by the concept of the Turing Test. Turing held that in future computers can be programmed to acquire abilities rivaling human intelligence. As part of his argument Turing put forward the idea of an 'imitation game', in which a human being and a computer would be interrogated under conditions where the interrogator would not know which was which, the communication being entirely by textual messages. Turing argued that if the interrogator could not distinguish them by questioning, then it would be unreasonable not to call the computer intelligent. Turing's 'imitation game' is now usually called 'the Turing test' for intelligence.

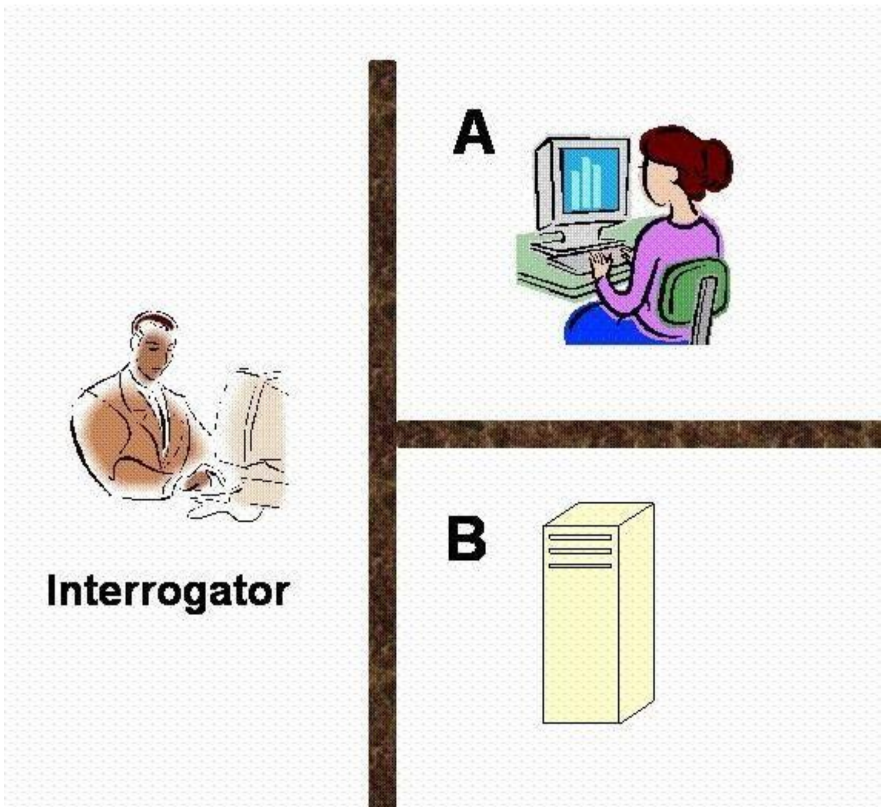


Figure 1: Turing Test

Turing Test

Consider the following setting. There are two rooms, A and B. One of the rooms contains a computer. The other contains a human. The interrogator is outside and does not know which one is a computer. Hecan ask questions through a teletype and receives answers from both A and B. The interrogator needs to identify whether A or B are humans.To pass the Turing test, the machine has to fool the interrogator into believing that it is human. For more details on the Turing test visit thesite http://cogsci.ucsd.edu/~asaygin/tt/ttest.html

1. Logic and laws of thought deals with studies of ideal or rational thought process and inference. The emphasis in this case is on the inferencing mechanism, and its properties. That is how the system arrives at a conclusion, or the reasoning behind its selection of actions is very important in this point of view. The soundness and completeness of the inference mechanisms areimportant here.

2. The fourth view of AI is that it is the study of rational agents. This view deals with building machines that act rationally. The focus is on how the system acts and performs, and not so much on the reasoning process. A rational agent is one that acts rationally, that is, is in the best possible manner.

3.1.2 Typical AI problems

While studying the typical range of tasks that we might expect an
―intelligent entity‖ to perform, we need to consider both ―common-place‖ tasks as well as expert tasks.
Examples of common-place tasks include

- *Recognizing* people, objects.
- Communicating (through *natural language*).
- *Navigating* around obstacles on the streets

These tasks are done matter of firstly and routinely by people and someother animals.

Expert tasks include:

- Medical diagnosis.
- Mathematical problem solving
- Playing games like chess

These tasks cannot be done by all people, and can only be performed byskilled specialists.

Now, which of these tasks are easy and which ones are hard? Clearly tasks of the first type are easy for humans to perform, and almost all are able to master them. The second range of tasks requires skill development and/or intelligence and only some specialists can perform them well. However, when we look at what computer systems have been able to achieve to date, we see that their achievements include performing sophisticated tasks like medical diagnosis, performing symbolic integration, proving theorems and playing chess.

On the other hand it has proved to be very hard to make computer systems perform many routine tasks that all humans and a lot of animals can do. Examples of such tasks include navigating our way without running into things, catching prey and avoiding predators. Humans and animals are also capable of interpreting complex sensory information. We are able to recognize objects and people from the visual image that we receive. We are also able to perform complex social functions.

Intelligent behaviour. This discussion brings us back to the question of what constitutes intelligent behaviour. Some of these tasks andapplications are:

- Perception involving image recognition and computer vision
- Reasoning
- Learning
- Understanding language involving natural language processing,speech processing
- Solving problems
- Robotics

3.1.3 Practical Impact of AI

AI components are embedded in numerous devices e.g. in copy machines for automatic correction of operation for copy quality improvement. AI systems are in everyday use for identifying credit card fraud, for advising doctors, for recognizing speech and in helping complex planning tasks. Then there are intelligent tutoring systems that provide students with personalized attention

Thus AI has increased understanding of the nature of intelligence and found many applications. It has helped in the understanding of human reasoning, and of the nature of intelligence. It will also help you understand the complexity of modeling human reasoning.

You can now look at a few famous AI systems.

1. ALVINN

Autonomous Land Vehicle in a Neural Network

In 1989, Dean Pomerleau at CMU created ALVINN. This is a system which learns to control vehicles by watching a person drive. It contains a neural network whose input is a 30x32 unit two dimensional camera image. The output layer is a representation of the direction the vehicle should travel.

The system drove a car from the East Coast of USA to the west coast, a total of about 2850 miles. Out of this about 50 miles were driven by a human being and the rest solely by the system.

2. Deep Blue

In 1997, the Deep Blue chess program created by IBM, beat the current world chess champion, Gary Kasparov.

3. Machine translation

A system capable of translations between people speaking different languages will be a remarkable achievement of enormous economic and cultural benefit. Machine translation is one of the important fields of endeavour in AI. While some translating systems have been developed, there is a lot of scope for improvement in translation quality.

4. Autonomous agents

In space exploration, robotic space probes autonomously monitor their surroundings, make decisions and act to achieve their goals.
NASA's Mars rovers successfully completed their primary three-month missions in April, 2004. The Spirit rover had been exploring a range of Martian hills that took two months to reach. It is finding curiously eroded rocks that may be new pieces to the puzzle of the region's past. Spirit's twin, Opportunity, had been examining exposed rock layers inside a crater.

5. Internet Agents

The explosive growth of the internet has also led to growing interest in internet agents to monitor users' tasks, seek needed information, and to learn which information is most useful

Approaches to AI

Strong AI aims to build machines that can truly reason and solve problems. These machines should be self aware and their overall intellectual ability needs to be indistinguishable from that of a human being. Excessive optimism in the 1950s and 1960s concerning strong AI has given way to an appreciation of the extreme difficulty of the problem. Strong AI maintains that suitably programmed machines are capable of cognitive mental states.

Weak AI deals with the creation of some form of computer-based artificial intelligence that cannot truly reason and solve problems, but can act as if it were intelligent. Weak AI holds that suitably programmedmachines can simulate human cognition.

Applied AI aims to produce commercially viable "smart" systems such as, security system that is able to recognise the faces of people who are permitted to enter a particular building. Applied AI has already enjoyed considerable success.

Cognitive AI: computers are used to test theories about how the human mind works--for example, theories about how we recognise faces and other objects, or about how we solve abstract problems.

Limits of AI Today

Today‘s successful AI systems operate in well-defined domains and employ narrow, specialized knowledge. Common sense knowledge is needed to function in complex, open-ended worlds. Such a system also needs to understand unconstrained natural language. However these capabilities are not yet fully present in today‘s intelligent systems.

✓ What can AI systems do?

Today‘s AI systems have been able to achieve limited success in some of these tasks.

- In Computer vision, the systems are capable of facerecognition
- In Robotics, we have been able to make vehicles that aremostly autonomous
- In Natural language processing, we have systems that arecapable of simple machine translation
- Today‘s Expert systems can carry out medical diagnosis ina narrow domain

- Speech understanding systems are capable of recognizingseveral thousand words continuous speech
- Planning and scheduling systems had been employed inscheduling experiments with the Hubble Telescope
- The Learning systems are capable of doing text categorization into about a 1000 topics
- In Games, AI systems can play at the Grand Master levelin chess (world champion), checkers, etc.

AI History

Intellectual roots of AI date back to the early studies of the nature of knowledge and reasoning. The dream of making a computer imitate humans also has a very early history.

The concept of intelligent machines is found in Greek mythology. There

is a story in the 8th century A.D about Pygmalion Olio, the legendary king of Cyprus. He fell in love with an ivory statue he made to represent his ideal woman. The king prayed to the goddess Aphrodite, and the goddess miraculously brought the statue to life. Other myths involve human-like artifacts. As a present from Zeus to Europa, Hephaestus created Talos, a huge robot. Talos was made of bronze and his duty was to patrol the beaches of Crete.

Aristotle (384-322 BC) developed an informal system of syllogistic logic, which is the basis of the first formal deductive reasoning system.

Early in the 17th century, Descartes proposed that bodies of animals are nothing more than complex machines.

Pascal in 1642 made the first mechanical digital calculating machine.

In the 19th century, George Boole developed a binary algebra representing (some) "laws of thought."

Charles Babbage & Ada Byron worked on programmable mechanical calculating machines.

In the late 19th century and early 20th century, mathematical philosophers like Gottlob Frege, Bertram Russell, Alfred North Whitehead, and Kurt Gödel built on Boole's initial logic concepts to develop mathematical representations of logic problems.

The advent of electronic computers provided a revolutionary advance in the ability to study intelligence.

In 1943 McCulloch & Pitts developed a Boolean circuit model of brain. They wrote the paper ―A Logical Calculus of Ideas Immanent in Nervous Activity‖, which explained how it is possible for neural networks to compute.

Marvin Minsky and Dean Edmonds built the SNARC in 1951, which is the first randomly wired neural network learning machine (SNARC stands for Stochastic Neural-Analog Reinforcement Computer).It was a neural network computer that used 3000 vacuum tubes and a network with 40 neurons.

In 1950 Turing wrote an article on ―Computing Machinery and Intelligence‖ which articulated a complete vision of AI. For more on Alan Turing see the site http://www.turing.org.uk/turing/ . Turing‗s paper talked of many things, of solving problems by searching through the space of possible solutions, guided by heuristics. He illustrated his ideas on machine intelligence by reference to chess. He even propounded the possibility of letting the machine alter its own instructions so that machines can learn from experience.

In 1956 a famous conference took place in Dartmouth. The conference brought together the founding fathers of artificial intelligence for the first time. In this meeting the term ―Artificial Intelligence‖ was adopted.

Between 1952 and 1956, Samuel had developed several programs for playing checkers. In 1956, Newell & Simon‗s Logic Theorist was published. It is considered by many to be the first AI program. In 1959, Gelernter developed a Geometry Engine. In 1961 James Slagle (PhD dissertation, MIT) wrote a symbolic integration program SAINT. It was written in LISP and solved calculus problems at the college freshman level. In 1963, Thomas Evan's program Analogy was developed which could solve IQ test type analogy problems.

In 1963, Edward A. Feigenbaum & Julian Feldman published Computers and Thought, the first collection of articles about artificial intelligence.

In 1965, J. Allen Robinson invented a mechanical proof procedure, the **Resolution Method**, which allowed programs to work efficiently with formal logic as a representation language. In

1967, the Dendral program (Feigenbaum, Lederberg, Buchanan, Sutherland at Stanford) was demonstrated which could **interpret mass spectra on organic chemical compounds**. This was the first successful knowledge-based program for scientific reasoning. In 1969 the SRI robot, Shakey, demonstrated combining locomotion, perception and problem solving.
The years from 1969 to 1979 marked the early development of knowledge-based systems

In 1974, MYCIN demonstrated the power of rule-based systems for knowledge representation and inference in medical diagnosis and therapy. Knowledge representation schemes were developed. These included frames developed by Minski. Logic based languages like Prolog and Planner were developed.

We will now mention a few of the AI systems that were developed over the years.

The Meta-Dendral learning program produced new results in chemistry (rules of mass spectrometry)

In the 1980s, Lisp Machines developed and marketed.Around 1985, neural networks return to

popularity.

In 1988, there was a resurgence of probabilistic and decision-theoreticmethods.

The early AI systems used general systems, little knowledge. AI researchers realized that specialized knowledge is required for rich tasksto focus reasoning.

The 1990's saw major advances in all areas of AI including the following:

- Machine learning, data mining
- Intelligent tutoring,
- Case-based reasoning,
- Multi-agent planning, scheduling,
- Uncertain reasoning,
- Natural language understanding and translation,
- Vision, virtual reality, games, and other topics.

Rod Brooks' COG Project at MIT, with numerous collaborators, made significant progress in building a humanoid robot.

The first official Robo-Cup soccer match featuring table-top matches with 40 teams of interacting robots was held in 1997. For details, see the site http://murray.newcastle.edu.au/users/students/2002/c3012299/bg. html

In the late 90s, Web crawlers and other AI-based information extractionprograms become essential in widespread use of the world-wide-web.

Interactive robot pets ("smart toys") become commercially available,realizing the vision of the 18th century novelty toy makers.

In 2000, the Nomad robot explores remote regions of Antarctica lookingfor meteorite samples.

AI in the news http://www.aaai.org/AITopics/html/current.html

CONCLUSION

Artificial intelligence (AI) is the intelligence of machines and the branch of computer science that aims to create it. AI textbooks define the field as "the study and design of intelligent agents" where an intelligent agent is a system that perceives its environment and takes actions that maximize its chances of success. John McCarthy, who coined the term in 1956, defines it as "the science and engineering of making intelligent machines."

The field was founded on the claim that a central property of humans, intelligence—the sapience of *Homo sapiens*—can be so precisely described that it can be simulated by a machine. This raises philosophical issues about the nature of the mind and the ethics of creating artificial beings, issues which have been addressed by myth, fiction and philosophy since antiquity. Artificial intelligence has been the subject of optimism, but has also suffered setbacks and, today, has become an essential part of the technology industry, providing the heavy lifting for many of the most difficult problems in computer science.

CHAPTER 2 INTRODUCTION TO INTELLIGENT AGENTS

Introduction to Agent

An agent perceives its environment through sensors. The complete set of inputs at a given time is called a percept. The current percept or a sequence of percepts can influence the actions of an agent. The agent can change the environment through actuators or effectors. An operation involving an Effector is called an action. Actions can be grouped into action sequences. The agent can have goals which it tries to achieve.

Thus, an agent can be looked upon as a system that implements a mapping from percept sequences to actions.

A performance measure has to be used in order to evaluate an agent.
An autonomous agent decides autonomously which action to take in the current situation to maximize progress towards its goals.

Agent Performance

An agent function implements a mapping from perception history to action. The behaviour and performance of intelligent agents have to be evaluated in terms of the agent function.

The ideal mapping specifies which actions an agent ought to take at any point in time.

The performance measure is a subjective measure to characterize how successful an agent is. The success can be measured in various ways. It can be measured in terms of speed or efficiency of the agent. It can be measured by the accuracy or the quality of the solutions achieved by the agent. It can also be measured by power usage, money, etc.

Examples of Agents

1. **Humans can be looked upon as agents.** They have eyes, ears, skin, taste buds, etc. for sensors; and hands, fingers, legs, mouth for effectors.

2. **Robots are agents.** Robots may have camera, sonar, infrared, bumper, etc. for sensors. They can have grippers, wheels, lights, speakers, etc. for actuators. Some examples of robots are Xavier from CMU, COG from MIT, etc.

Figure 2: Xavier Robot (CMU)

Then we have the AIBO entertainment robot from SONY.

Figure 3: Aibo from SONY

3. We also have software agents or softbots that have some functions as sensors and some functions as actuators. Askjeeves.com is an example of a softbot.
4. Expert systems like the Cardiologist are an agent.
5. Autonomous spacecrafts.
6. Intelligent buildings.

Agent Faculties

The fundamental faculties of intelligence are

- Acting
- Sensing
- Understanding, reasoning, learning

Blind action is not a characterization of intelligence. In order to act intelligently, one must sense. Understanding is essential to interpret thesensory percepts and decide on an action. Many robotic agents stress sensing and acting, and do not have understanding.

Intelligent Agents

An Intelligent Agent must sense, must act, must be autonomous (tosome extent). It also must be rational.

AI is about building rational agents. An agent is something thatperceives and acts.

A rational agent always does the right thing.

1. What are the functionalities (goals)?
2. What are the components?
3. How do we build them?

Rationality

Perfect Rationality assumes that the rational agent knows all and will take the action that maximizes her utility. Human beings do not satisfy this definition of rationality.

Rational Action is the action that maximizes the expected value of the performance measure given the percept sequence to date.

However, a rational agent is not omniscient. It does not know the actual outcome of its actions, and it may not know certain aspects of its environment. Therefore rationality must take into account the limitations of the agent. The agent has too select the best action to the best of its knowledge depending on its percept sequence, its background knowledge and its feasible actions. An agent also has to deal with the expected outcome of the actions where the action effects are not deterministic.

Bounded Rationality

―Because of the limitations of the human mind, humans must use approximate methods to handle many tasks.‖ Herbert Simon, 1972 Evolution did not give rise to optimal agents, but to agents which are insome senses locally optimal at best. In 1957, Simon proposed the notionof Bounded Rationality: that property of an agent that behaves in amanner that is nearly optimal with respect to its goals as its resourceswill allow.

Under these promises an intelligent agent will be expected to actoptimally to the best of its abilities and its resource constraints.

Agent Environment

Environments in which agents operate can be defined in different ways. It is helpful to view the following definitions as referring to the way the environment appears from the point of view of the agent itself.

Observability

In terms of observability, an environment can be characterized as fully observable or partially observable.

In a fully observable environment, the entire environment relevant to the action being considered is observable. In such environments, the agent does not need to keep track of the changes in the environment. A chess playing system is an example of a system that operates in a fully observable environment.

In a partially observable environment, the relevant features of the environment are only partially observable. A bridge playing program is an example of a system operating in a partially observable environment.

Determinism

In deterministic environments, the next state of the environment is completely described by the current state and the agent‘s action. Image analysis

If an element of interference or uncertainty occurs then the environment is stochastic. Note that a deterministic yet partially observable environment will *appear* to be stochastic to the agent.

If the environment state is wholly determined by the preceding state and the actions of *multiple* agents, then the environment is said to be strategic. Example: Chess, Ludo

Episodicity

An episodic environment means that subsequent episodes do not depend on what actions occurred in previous episodes.

In a sequential environment, the agent engages in a series of connected episodes.

Dynamism

Static Environment: does not change from one state to the next while the agent is considering its course of action. The only changes to the environment are those caused by the agent itself.

• A static environment does not change while the agent is thinking.

- The passage of time as an agent deliberates is irrelevant.
- The agent doesn‘t need to observe the world during deliberation.

A Dynamic Environment changes over time independent of the actions of the agent -- and thus if an agent does not respond in a timely manner, this counts as a choice to do nothing

Continuity

If the number of distinct percepts and actions is limited, the environment is discrete, otherwise it is continuous.

Presence of Other agents

Single agent/ Multi-agent

A multi-agent environment has other agents. If the environment contains other intelligent agents, the agent needs to be concerned about strategic, game-theoretic aspects of the environment (for either cooperative *or* competitive agents)

Most engineering environments do not have multi-agent properties, whereas most social and economic systems get their complexity from the interactions of (more or less) rational agents.

Agent architectures

Table Based Agent

In table based agent the action is looked up from a table based on information about the agent‘s percepts. A table is simple way to specify a mapping from percepts to actions. The mapping is implicitly defined by a program. The mapping may be implemented by a rule basedsystem, by a neural network or by a procedure.

There are several disadvantages to a table based system. The tables may become very large. Learning a table may take a very long time, especially if the table is large. Such systems usually have little autonomy, as all actions are pre-determined.

Percept based agent or reflex agent

In percept based agents,

1. information comes from sensors - percepts
2. changes the agents current state of the world
3. triggers actions through the effectors

Such agents are called reactive agents or stimulus-response agents. Reactive agents have no notion of history. The current state is as the sensors see it right now. The action is based on the current percepts only.

The following are some of the characteristics of percept-based agents.

- Efficient
- No internal representation for reasoning, inference.
- No strategic planning, learning.
- Percept-based agents are not good for multiple, opposing, goals.

Subsumption Architecture

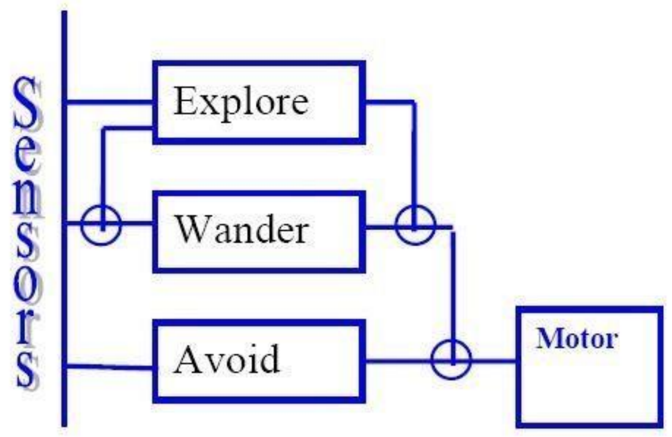
We will now briefly describe the subsumption architecture (Rodney Brooks, 1986). This architecture is based on reactive systems. Brooks notes that in lower animals there is no deliberation and the actions are based on sensory inputs. But even lower animals are capable of many complex tasks. His argument is to follow the evolutionary path and buildsimple agents for complex worlds.

The main features of Brooks‘ architecture are.

- There is no explicit knowledge representation
- Behaviour is distributed, not centralized
- Response to stimuli is reflexive
- The design is bottom up, and complex behaviours are fashionedfrom the combination of simpler underlying ones.
- Individual agents are simple

The Subsumption Architecture built in layers. There are different layers of behaviour. The higher layers can override lower layers. Each activity is modeled by a finite state machine.

The subsumption architecture can be illustrated by Brooks‘ MobileRobot example.



Subsumption Architecture

Figure 4: Subsumption ArchitectureThe system is built in three

layers.

1. **Layer 0:** Avoid Obstacles
2. **Layer1:** Wander behaviour
3. **Layer 2:** Exploration behavior

Layer 0 (Avoid Obstacles) has the following capabilities:

- **Sonar:** generate sonar scan
- **Collide:** send HALT message to forward
- **Feel force:** signal sent to run-away, turn

Layer1 (Wander behaviour)

- Generates a random heading
- Avoid reads repulsive force, generates new heading, feeds to turnand forward

Layer 2 (Exploration behaviour)

- Whenlook notices idle time and looks for an interesting place.
- Pathplan sends new direction to avoid.
- Integrate monitors path and sends them to the path plan.

State-Based Agent or Model-Based Reflex Agent

State based agents differ from percept based agents in that such agents maintain some sort of state based on the percept sequence received so far. The state is updated regularly based on what the agent senses, and the agent‘s actions. Keeping track of the state requires that the agent has knowledge about how the world evolves, and how the agent‘s actions affect the world.

Thus a state based agent works as follows:

- information comes from sensors – percepts
- based on this, the agent changes the current state of the world
- based on state of the world and knowledge (memory), it triggersactions through the effectors

Goal-based Agent

The goal based agent has some goal which forms a basis of its actions. Such agents work as follows:

- information comes from sensors - percepts

- changes the agents current state of the world
- based on state of the world and knowledge (memory) and goals/intentions, it chooses actions and does them through the effectors.

Goal formulation based on the current situation is a way of solving many problems and search is a universal problem solving mechanism in AI. The sequence of steps required to solve a problem is not known a priori and must be determined by a systematic exploration of the alternatives.

Utility-based Agent

Utility based agents provide a more general agent framework. In case that the agent has multiple goals, this framework can accommodate different preferences for the different goals.
Such systems are characterized by a utility function that maps a state or a sequence of states to a real valued utility. The agent acts so as to maximize expected utility.

Learning Agent

Learning allows an agent to operate in initially unknown environments. The learning element modifies the performance element. Learning is required for true autonomy

CONCLUSION

In conclusion, an intelligent agent (IA) is an autonomous entity which observes and acts upon an environment . Intelligent agents may also learn or use knowledge to achieve their goals. They may be very simple or very complex: a reflex machine such as a thermostat is an intelligent agent, as is a human being, as is a community of human beings working together towards a goal.

CHAPTER 3 KNOWLEDGE REPRESENTATION

Overview of Knowledge Representation

Knowledge Representation (KR) research involves analysis of how to accurately and effectively reason and how best to use a set of symbols to represent a set of facts within a knowledge domain.

- A symbol vocabulary and a system of logic are combined to enable inferences about elements in the KR to create new KR sentences.
- Logic is used to supply formal semantics of how reasoning functions should be applied to the symbols in the KR system.
- Logic is also used to define how operators can process and reshape the knowledge. Examples of operators and operations include negation, conjunction, adverbs, adjectives, quantifiers and modal operators. Interpretation theory is this logic.
- These elements--symbols, operators, and interpretation theory-- are what give sequences of symbols meaning within a KR.

A key parameter in choosing or creating a KR is its *expressivity*. The more expressive a KR, the easier and more compact it is to express a fact or element of knowledge within the semantics and grammar of that KR.
However, more expressive languages are likely to require more complex logic and algorithms to construct equivalent inferences.
A highly expressive KR is also less likely to be complete and consistent.
Less expressive KRs may be both complete and consistent.
Auto epistemic temporal modal logic is a highly expressive KR system, encompassing meaningful chunks of knowledge with brief, simple symbol sequences (sentences). Propositional logic is much less expressive but highly consistent and complete and can efficiently produce inferences with minimal algorithm complexity.

Nonetheless, only the limitations of an underlying knowledge base affect the ease with which inferences may ultimately be made (once the appropriate KR has been found). This is because a knowledge set may be exported from a knowledge model or knowledge base system (KBS) into different KRs, with different degrees of expressiveness, completeness, and consistency.
If a particular KR is inadequate in some way, that set of problematic KR elements may be transformed by importing them into a KBS, modified and operated on to eliminate the problematic elements or augmented with additional knowledge imported from other sources, and then exported into a different, more appropriate KR.

In applying KR systems to practical problems, the complexity of the problem may exceed the resource constraints or the capabilities of the KR system. Recent developments in KR include the concept of the Semantic Web, and development of XML-based knowledge representation languages and standards, including Resource Description Framework (RDF), RDF Schema, Topic Maps, DARPA Agent Mark-up Language (DAML), Ontology Inference Layer (OIL), and WebOntology Language (OWL).

There are several KR techniques such as frames, rules, tagging, and semantic networks which originated in Cognitive Science. Since knowledge is used to achieve intelligent behaviour, the fundamental goal of knowledge representation is to facilitate reasoning, drawingconclusions. A good KR must be both declarative and procedural knowledge.
What is knowledge representation can best be understood in terms of five distinct roles it plays, each crucial to the task at hand:

- A knowledge representation (KR) is most fundamentally a surrogate, a substitute for the thing itself, used to enable an entity to determine consequences by thinking rather than acting, i.e., by reasoning about the world rather than taking action in it.
- It is a set of ontological commitments, i.e., an answer to the question: In what terms should I think about the world?
- It is a fragmentary theory of intelligent reasoning, expressed in terms of three components:
   - (i) the representation's fundamental conception of intelligent reasoning;
   - (ii) the set of inferences the representation sanctions; and
   - (iii) the set of inferences it recommends.
- It is a medium for pragmatically efficient computation, i.e., the computational environment in which thinking is accomplished. One contribution to this pragmatic efficiency is supplied by the guidance a representation provides for organizing information so as to facilitate making the recommended inferences.
- It is a medium of human expression, i.e., a language in which we say things about the world."

Some issues that arise in knowledge representation from an AI perspective are:

- How do people represent knowledge?
- What is the nature of knowledge?
- Should a representation scheme deal with a particular domain orshould it be general purpose?
- How expressive is a representation scheme or formal language?
- Should the scheme be declarative or procedural?

There has been very little top-down discussion of the knowledge representation (KR) issues and research in this area is a well aged quillwork. There are well known problems such as "spreading activation" (this is a problem in navigating a network of nodes), "subsumption" (this is concerned with selective inheritance; e.g. an ATV can be thought of as a specialization of a car but it inherits only particular characteristics) and "classification." For example a tomatocould be classified both as a fruit and a vegetable.

In the field of artificial intelligence, problem solving can be simplified by an appropriate choice of *knowledge representation*. Representing knowledge in some ways makes certain problems easier to solve. For example, it is easier to divide numbers represented in Hindu-Arabic numerals than numbers represented as Roman numerals.

Characteristics

A good knowledge representation covers six basic characteristics:

1. Coverage, which means the KR covers a breath and depth of information. Without a wide coverage, the KR cannot determine anything or resolve ambiguities.
2. Understandable by humans. KR is viewed as a natural language, so the logic should flow freely. It should support modularity and hierarchies of classes (Polar bears are bears, which are animals).

3. It should also have simple primitives that combine in complex forms.
4. Consistency. If John closed the door, it can also be interpreted as the door was closed by John. By being consistent, the KR can eliminate redundant or conflicting knowledge.
5. Efficient: Easiness for modifying and updating.
6. Supports the intelligent activity which uses the knowledge base

To gain a better understanding of why these characteristics represent a good knowledge representation, think about how an encyclopaedia (e.g. Wikipedia) is structured. There are millions of articles (coverage), and they are sorted into categories, content types, and similar topics (understandable). It redirects different titles but same content to the same article (consistency). It is efficient, easy to add new pages or update existing ones, and allows users on their mobile phones and desktops to view its knowledge base.

History of Knowledge Representation and Reasoning

In computer science, particularly artificial intelligence, a number of representations have been devised to structure information.

KR is most commonly used to refer to representations intended for processing by modern computers, and in particular, for representations consisting of explicit objects (the class of all elephants, or Clyde a certain individual), and of assertions or claims about them ('Clyde is an elephant', or 'all elephants are grey'). Representing knowledge in such explicit form enables computers to draw conclusions from knowledge already stored ('Clyde is grey').

Many KR methods were tried in the 1970s and early 1980s, such as heuristic question-answering, neural networks, theorem proving, and expert systems, with varying success. Medical diagnosis (e.g., Mycin) was a major application area, as were games such as chess.

In the 1980s formal computer knowledge representation languages and systems arose. Major projects attempted to encode wide bodies of general knowledge; for example the "Cyc" project (still ongoing) went through a large encyclopaedia, encoding not the information itself, but the information a reader would need in order to understand the encyclopaedia: naive physics; notions of time, causality, motivation; commonplace objects and classes of objects.

Through such work, the difficulty of KR came to be better appreciated. In computational linguistics, meanwhile, much larger databases of language information were being built, and these, along with great increases in computer speed and capacity, made deeper KR more feasible.

Several programming languages have been developed that are oriented to KR. Prolog developed in 1972, but popularized much later, represents propositions and basic logic, and can derive conclusions from known premises. KL-ONE (1980s) is more specifically aimed at knowledge representation itself. In 1995, the Dublin Core standard of metadata was conceived.

In the electronic document world, languages were being developed to represent the structure of documents, such as SGML (from which HTML descended) and later XML. These facilitated information retrieval and data mining efforts, which have in recent years begun to relate to knowledge representation.

Development of the Semantic Web, has included development of XML- based knowledge representation languages and standards, including RDF, RDF Schema, Topic Maps, DARPA Agent Markup Language (DAML), Ontology Inference Layer (OIL), and Web Ontology Language (OWL).

Knowledge Representation Languages

William Woods defines the properties of a KR Language as follows:
A KR language must unambiguously represent any interpretation of a sentence (logical adequacy), have a method for translating from natural language to that representation, and must be usable for reasoning [Woods, 1975].

Wood's definition is merely a simplification of the KR Hypothesis where "reasoning" is the only method of "engendering the behavior that manifests that knowledge." Reasoning is essential to KR, and especially to KR languages, yet even simple reasoning capabilities can lead to serious tractability problems [Brachman and Levesque, 1987], and thus must be well understood and used carefully.

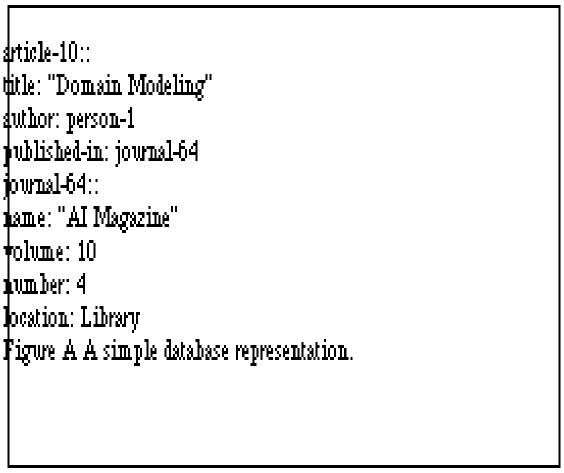
One of the most important developments in the application of KR in the past 20 years has been the proposal [Minsky, 1981], study [Woods, 1975] [Brachman, 1977] [Brachman, 1979], and development [Brachman and Schmolze, 1985] [Fox, Wright, and Adam, 1985] [Bobrow and Winograd, 1985] of frame-based KR languages. While frame-based KR languages differ in varying degrees from each other, the central tenet of these systems is a notation based on the specification of objects (concepts) and their relationships to each other. The main features of such a language are:

1. *Object-orientedness*. All the information about a specific concept is stored with that concept, as opposed, for example, to rule- based systems where information about one concept may be scattered throughout the rule base.

2. *Generalization/Specialization*. Long recognized as a key aspect of human cognition [Minsky, 1981], KR languages provide a natural way to group concepts in hierarchies in which higher level concepts represent more general, shared attributes of the concepts below.

3. *Reasoning*. The ability to state in a formal way that the existence of some piece of knowledge implies the existence of some other, previously unknown piece of knowledge is important to KR. Each KR language provides a different approach to reasoning.

4. *Classification*. Given an abstract description of a concept, most KR languages provide the ability to determine if a concept fits that description, this is actually a common special form of reasoning.



```
article-10::
title: "Domain Modeling"
author: person-1
published-in: journal-64
journal-64::
name: "AI Magazine"
volume: 10
number: 4
location: Library
```

Figure A A simple database representation.

Object orientation and generalization help to make the represented knowledge more understandable to humans; reasoning and classification help make a system behave as if it knows what is represented. Frame- based systems thus meet the goals of the KR Hypothesis.

It is important to realize both the capabilities and limitations of frame- based representations, especially as compared to other formalisms.

To begin with, all symbolic KR techniques are derived in one way or another from First Order Logic (FOL), and as a result are suited for representing knowledge that doesn't change.(Figure one Simple database representation) . Different KR systems may be able to deal with non-monotonic changes in the knowledge being represented, but the basic assumption has been that change, if present, is the exception rather than the rule.

Two other major declarative KR formalisms are *production systems* and *database systems*.
Production systems allow for the simple and natural expression of *if-then rules*. However, these systems have been shown to be quite restrictive when applied to large problems, as there is no ordering of the rules, and inferences cannot be constrained away from those dealing only with the objects of interest.

Production systems are subsumed by frame-based systems, which additionally provide natural inference capabilities like classification and inheritance, as well as knowledge-structuring techniques such as generalization and object orientation.

Database systems provide only for the representation of simple assertions, without inference. Rules of inference are important pieces of knowledge about a domain. For example, consider the bibliographic database in Figure 1. If someone were interested in article-10 and wanted to know where it was, that person would have to be smart enough to realize that *an article can be found in the location of the journal in which it is published*. That sentence is a rule, it is *knowledge*. It is knowledge that cannot be expressed in a database system. The person doing the retrieval of the information in the database must have that knowledge in order to type the SQL statement that will get the proper location. In a frame-based system that knowledge can be expressed as a rule that will fire when article-10 is accessed, thus the user is not required to know it.

Frame-based systems are currently severely limited when dealing with procedural knowledge [Winograd, 1975]. An example of procedural knowledge would be Newton's Law of Gravitation - the attraction between two masses is inversely proportional to the square of their distances from each other. Given two frames representing the two bodies, with slots holding their positions and mass, the value of the gravitational attraction between them cannot be (Figure 2-A framework for Domain Modelling).

Inferred declaratively using the standard reasoning mechanisms available in frame-based KR languages, though a function or procedure in a programming language could represent the mechanism for performing this "inference" quite well. Frame-based systems that can deal with this kind of knowledge do so by adding a procedural language to its representation. The knowledge is not being represented in a frame- based way, it is being represented as C or (more commonly) LISP code which is accessed through a slot in the frame [Bobrow and Winograd, 1985]. This is an important distinction - there is knowledge being.

Encoded in those LISP functions that is not fully accessible. The system can reason *with* that knowledge, but not *about* it; in other words we can use some attached procedure to compute (or infer) the value of one slot based on some others, but we cannot ask how that value was obtained.

Domain *Modeling*

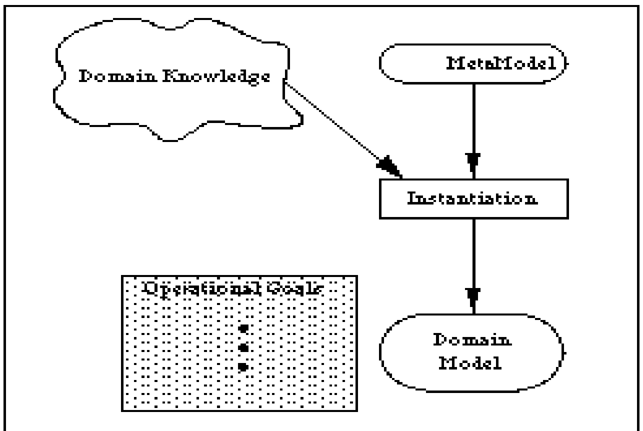


Figure A A framework for Domain Modeling

Domain modeling is the field in which the application of KR to specific domains is studied and performed. Figure 2 shows a framework for discussing domain modeling that seems to map well onto most examples[Iscoe, Tam, and Liu, 1991].

The amorphous shape labelled *Domain Knowledge* refers to the knowledge possessed by the domain expert that must be encoded in some fashion. This knowledge is not well defined and is fairly difficult for others to access. The box labelled *Meta-Model* refers to the KR formalism, typically a KR language that will be used as the *symbol level* [Newell, 1982] for the machine representation of this knowledge. The box labelled *instantiation* refers to the *process* of taking the domain knowledge and physically representing it using the meta-model, this process is sometimes referred to as *knowledge acquisition* [Schoen, 1991]. The box labelled *domain model* refers to the knowledge-base that results from the instantiation, and the *operational goals* are typically not represented formally, but refer to the reason the domain model was built and what it will be used for.

Specific examples of real-world domain modelling efforts and how they fit into this framework can be found in [Iscoe, 1991], and it has become clear that the most prevalent operational goal across modelling efforts today is understanding the domain of a large software system [Arango, 1989].

One thing that seems to be universally lacking in efforts with this operational goal is the realization that a software system operating within a domain *is a part of that domain*, and deserves as much attention and detail in the model as any other part. The main reason for this oversight is that there is a historical reason for distinguishing *procedural* from *declarative* knowledge [Winograd, 1975], and as a result the two are typically represented differently: domain models are represented with frame based KR languages and programs are represented withprogramming languages.

This traditional separation between programs and domain models causes problems during the instantiation of a domain model that includes not only knowledge of the objects and attributes, but knowledge of the procedural aspects of the processes associated with the domain as well. The problems stem from the fact that domain modelling is a discipline in which advances are made incrementally, by building upon previous systems [Simon, 1991]. Some of the most significant results are in the form of methodologies which help other domain modellers to avoid pitfalls and use techniques that work [Gruber, 1993].

The predominant methodologies for domain modelling clearly indicate that the instantiation of the model is the most time consuming part, and that the most important part of instantiation is *ontological analysis* [Alexander, Freiling, and Shulman, 1986] (which is more fully described in the next section).

Ontologies for general taxonomies of objects are abundant, and there seem to be clear guidelines for developing new ones.

The problem is that for the knowledge required to represent procedural knowledge and reason about it (not with it); there are few guidelines, especially when the procedures requiring representation are implemented as software. There is not much background to draw upon, other than software information systems, as far as ontologies and methodologies for modelling what software does. Ontological analysis ended up being a large part of the effort for this research, since it had never been done before.

Ontological Analysis

The word *ontology* means "the study of the state of being." *Ontology* describes the states of being of a particular set of things. This description is usually made up of axioms that define each thing. In knowledge representation, ontology has become the defining term for the part of a domain model that excludes the *instances*, yet describes what they can be. Ontological analysis is the process of defining this part of the model.

What makes up a specific domain ontology is restricted by the representational capabilities of the *meta-model* - the language used to construct the model. Each knowledge representation language differs in its manner and range of expression. In general, ontology consists of three parts: *concept* definitions, *role* definitions, and further inference definitions.

CONCEPT DEFINITIONS

The concept definitions set up all the *types* of objects in the domain. In object oriented terms this is called the class definitions, and in database terms these are the entities. There can be three parts to the concept definitions:

- Concept taxonomy. The taxonomy is common to most knowledge representation languages, and through it is specified the nature of the categories in terms of generalization and specialization.
- Role defaults which specify for each concept what the default values are for any attributes.

- Role restrictions which specify for a concept any constraints on the values in a role, such as what types the values must be, how many values there can be, etc.

A role is an attribute of an object. In object-oriented terms it is a slot, in database terms (and even some KR languages) it is a relation. In the simplest case, a role for an object just has a value; the object mailbox-4 might have a role number-of-messages, for example, that would have a value which is a number.

ROLE DEFINITION

Roles also express relationships between objects. The same object might have a role called owner which relates mailbox-4 to the object person-2. Roles which represent relationships are unidirectional. A role definition may have up to three parts as well:

- The role taxonomy which specifies the generalization/specialization relationship *between* roles. For example, ontology for describing cars might include roles called has-engine, has-seats, and has-headlights, which relate objects that represent cars to objects that represent engines, seats, and headlights, resp. The role has-parts, then, could be expressed as the generalization of all these roles, and the result is that all the values of all the more specialized roles would also be values of the more general role.
- Role inverses which provide a form of inference that allows the addition of a role in the opposite direction when the forward link is made. For example, if the inverse of has-engine was engine-of, then when the has-engine link between the object that represents the car and the object that represents the engine is made, the engine-of link will automatically be added between the engine object and the car object.
- Role restrictions. The role itself may be defined such that it can only appear between objects of certain types (domain/range restrictions), or can only appear a specified number of times (cardinality restriction). This is the same information specified in role restriction for concepts, some representation languages consider this information to be part of the role, and some considerit to be part of the concept.

INFERENCE DEFINITION

The final part of ontology is the specification of additional inference that the language provides. Examples of this are forward and/or backward chaining rules, path grammars, subsumption and/or classification, demons, etc. An explanation of the inference mechanisms used in this research will be given in the next section.

Classic

Classic is a frame-based knowledge representation language that belongs to the family of *description logics* [Brachman, et al., 1991]. It is descended from KL-ONE [Brachman and Schmolze, 1985], and has been specifically designed to be mindful of the tractability of its own inferences [Brachman, et al., 1989].

Classic knowledge-bases are composed of four kinds of objects: *concepts*, *roles*, *rules*, and *individuals*. Ontology in Classic consists of concept taxonomy, role taxonomy, role inverses, role restrictions and defaults. The role restrictions and defaults are specified as part of the concept definitions. Classic rules are forward chaining rules.

The Classic Language

Throughout this document, it has been necessary to make explicit use of the notation and terminology of Classic to explain and describe some of the more detailed aspects of this research. This section contains a brief introduction to the language of Classic in order to make it clear precisely how one goes about describing objects. This introduction only presents the subset of Classic which was used in this research. The Classic language is specifically designed to make it possible to describe objects in such a way that it is possible to determine automatically whether one object is subsumed by another. The peculiarities of the language arise from the goal of making this determination not only possible, but tractable.

To begin with, a typical concept description looks like this:(defconcept information-filter

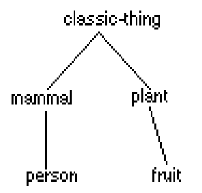
```
(and kbeds-object
(all information-filter-of valid-mail-recipient)(at-least one information-filter-of)
(all has-filter-condition kbeds-mail-message)(at-least one has-filter-condition)
(at-most one has-filter-condition)
(all has-filter-action kbeds-filter-action)(at-least one has-filter-action))
:disjoint kbeds-thing)
```

This says an information-filter is subsumed by (or is more specialized than, or is a subclass of) kbeds-object (and therefore is also described by that concept), and that all the *fillers* (in Classic a filler is the value of a particular role on a particular object) for its information-filter-of role must be individuals of valid- mail-recipient, and that there must be at least one filler for that role, and similarly for the role has-filter-condition except that there can also be at most one filler (in other words, there can be only one filler), and so on. The disjoint specification identifies this concept as a member of a disjoint set, which means that an individual cannot be subsumed by more than one concept in that set. The most obvious example of a disjoint set would be the gender set, containing the concepts "male person" and "female person."

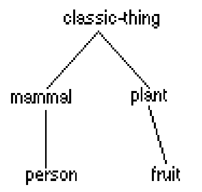
In the terminology of Classic, the description above is *told* information. Told information is precisely the information that is explicitly typed into the knowledge base. This is opposed to *derived* information, which is all the information that Classic derives or infers through its various mechanisms. (Figure 3 – A Simple taxonomy of primitive concepts).

This description actually shows a *primitive* concept - one which Classic will not automatically try to classify. Classic will also not automatically try to find which individuals are subsumed by a primitive concept, this information must be told. This subtle notion may seem irrelevant, but it is the norm in most representation languages (when a concept is created the modeller explicitly names the parent concepts, and when an
individual is created, the modeller explicitly names the concept that the new individual is an instance of). It is important in Classic because thereis another kind of concept, the *defined* concept, which Classic actually does automatically classify and find individuals of. For

example, in figure 3 a simple taxonomy of primitive concepts is shown. Let ussuppose we create a defined concept called vegetarian-mammal as follows: (and mammal (all food plant)). Next we create another defined concept called fruit-eating-person: (andperson (all food fruit)). Classic will derive thatvegetarian-mammal *subsumes* fruit-eating-person (why? because mammal subsumes person and plant subsumes fruit). If we created two individuals, joe and apple, and tell Classic that they are instances of person and fruit, resp., and further tell Classic that apple is a filler for joe's food role, Classic will derive that joe is an instance of fruit-eating-person (and therefore also vegetarian-mammal). Again, Classic will never derive that an individual is an instance of a primitive concept, it must always be told that.



igure A A simple taxonomy of primitive concepts.



igure A A simple taxonomy of primitive concepts.

This automatic classification of individuals of defined concepts through subsumption is a simple, yet extremely powerful process. It is the key to several significant advances in software information systems described in later sections.

Another important point about Classic is the *Open World Assumption*. Classic does not assume that the information it knows is all the information there is. Returning to the example above, we have the following *told* information about two individuals, joe: (and person(fills food apple)), and apple: fruit. Classic will then add all the *derived* information it can to these individuals, yielding joe: (and person mammal classic-thing (fills food apple)), and apple: (and fruit plant classic-thing). Where is the information about joe being a fruit- eating-person? The truth is that Classic cannot derive this yet. The definition of fruit-eating-person specifies that *all* the fillers for the food role of an individual must be instances of fruit. Classic does not assume that because it knows one (or two, or zero, etc.) fillers for an individual's role, that it knows them all. In fact, Classic assumes the opposite: it assumes it does not know them all.

There are two ways for Classic to figure out that it knows all the fillers for an individual's role. The first way is for it to be told, by *closing* the role. When a role is closed on an individual, it tells Classic that there can be no more filler. In the above example, the user would have to close the food role on joe in order for Classic to derive that joe is an instance of fruit-eating-person (Classic always tries to reclassify individuals when their roles are closed). The second way is for Classic to derive that a role is closed on an individual if there is an *at-most* restriction on the role. For example, if the concept person (or mammal) additionally had (at-most one food) in its description, then since joe is told to be an instance of person that restriction would apply to him, and since he already has one filler in his food role, *and* since he can have at most one filler in his food role, he can have no more and the role is derived to be closed.

The final part of ontology in Classic is the rules. Classic rules come in two forms, *description* rules and *filler* rules. All classic rules have as their antecedent a named concept, and are fired on an individual when the individual is classified as an instance of the concept.

The consequent of a classic description rule is a classic description which, when the rule fires on an individual, is merged into the description of the individual. For example, if we had a description rule like: vegetarian-mammal --> *(at-most 0 has-prey)*, the rule would fire on joe when he is classified as a fruit-eating- person and would add the at-most restriction to the description of joe. Classic would then also derive that joe's has-prey role is closed as well.

The consequent of a classic filler rule is the name of a role and a LISP function that will be invoked when the rule fires. The function is passed the individual the rule fired on and the role named in the consequent, and returns a list of new fillers for that role and individual. One useful application for filler rules is to create inter-role dependencies. For example, the concept rectangle has three roles: length, width, and area. We could define a function for calculating the area in LISP as follows:

```
(defun calculate-area (rect role)


(let ((length (car (cl-fillers rect @length)))(width (car (cl-fillers rect @width))))
(* length width)))
```

And then define a filler rule: rectangle --> *area calculate-area*, the filler for the area role would automatically be generated based on the fillers inthe length and width roles.

Enhancements to Classic

It was necessary to extend Classic in several ways in order to support this research. Each extension had a different motivation, which may not be entirely clear until that aspect of the research is discussed. These extensions are explained here, however, so that the sections involvingthe research do not need to sidetrack into explanations of the underlying support.

The first extension to Classic was a facility for supporting what some other representation languages call *path grammars* or role *transitivity* [Fox, Wright, and Adam, 1985]. A very common form of inference in frame-based representations is one in which the fillers for one role in a class of individuals can always be found by following the same *role path*. For example, an individual representing an article in a journal might have a role called published-in which is filled with an individual representing a journal. Journal individuals could have a role called location which is filled with some string indicating the place where the journal is physically located. It makes sense that the article individual should also have a location that is *the same as the location of the journal it is published in*, and this can be represented as a path rule. A path rule, like all Classic rules, has for its antecedent a concept name, and for its consequent a role and a role path (an ordered list of roles). When a rule fires on an individual, classic follows the role path to the end, and fills the role specified in the rule with all the values it finds at the end of the path. In the journal article example, the path rule would be: article --> *location (published-in location)*. An individual of article might be described: (and article (fills published-in journal-10)), and the journal individual journal-10: (and journal (fills location "Shelf 10")). Classic would fire the path rule on the individual ofarticle and follow the path: the first part of the path is published-in, which gets us to article-10, and the next part of the path is location which gets us to the string "Shelf 10." This value is then derived to be the filler for the location role of the individual of article. If a particular path ends "early," that is, the path leads to an intermediate individual that has no fillers for the next role in the path, no fillers are returned for that particular branch of the path.

The path rule facility was further expanded to allow for the expression of *specialization overrides*. A specialization override is a type of inference in which a value that would normally be derived to fill a role is blocked if and only if there is already a value filling the role that is more specialized. The most common example of this is in object- oriented languages, a class inherits all the methods of its superclass, except the ones that are already defined by the class.
The next enhancement to Classic was a facility for dumping individuals into a file; in essence there is no way to save the current state of a classic knowledge-base. While this may not sound like a significant extension, there is one aspect of dumping (or, more accurately, of loading) a knowledge-base that is very intricate: the order in which roles are closed. When a role is told to be closed on an individual, it means there can be no more filler for that role - told or derived. However, when derived role filler depends on filler or fillers in other individuals, the role cannot be closed until the fillers it depends on are closed. There is an implicit ordering of role closing based on all the ways Classic can derive information.

The most significant enhancement to Classic was the addition of a facility for representing spanning objects [Welty and Ferrucci, 1994]. In reality, this enhancement is in the process of being made to Classic by its support team, and the spanning object facility used for this research was actually applied to the "dumped" knowledge-based - that is, the spanning functions worked by generating a text file containing Classic descriptions, then the knowledge-base was cleared and the text file could be loaded in. Until support for multiple universes of discourse is added to Classic (which will happen in the next major release), this was the only way to proceed. The only limitation this presented was an inability to change the first universe from the second.

CONCLUSION

Knowledge representation (KR) is an area of artificial intelligence research aimed at representing knowledge in symbols to facilitate inferencing from those knowledge elements, creating new elements of knowledge. The KR can be made to be independent of the underlying knowledge model or knowledge base system (KBS) such as a semantic network.

CHAPTER 4 NATURAL LANGUAGE PROCESSING

INTRODUCTION

Natural language processing (NLP) is a field of computer science and linguistics concerned with the interactions between computers and human (natural) languages; it began as a branch of artificial intelligence. In theory, natural language processing is a very attractive method of human–computer interaction.
Natural language understanding is sometimes referred to as an AI-complete problem because it seems to require extensive knowledge about the outside world and the ability to manipulate it.

History of natural language processing

The history of NLP generally starts in the 1950s, although work can be found from earlier periods. In 1950, Alan Turing published his famous article "Computing Machinery and Intelligence" which proposed what is now called the Turing test as a criterion of intelligence. This criterion depends on the ability of a computer program to impersonate a human in a real-time written conversation with a human judge, sufficiently well that the judge is unable to distinguish reliably — on the basis of the conversational content alone — between the program and a real human. The Georgetown experiment in 1954 involved fully automatic translation of more than sixty Russian sentences into English. The authors claimed that within three or five years, machine translation would be a solved problem. However, real progress was much slower, and after the ALPAC report in 1966, which found that ten years long research had failed to fulfill the expectations, funding for machine translation was dramatically reduced. Little further research in machine translation was conducted until the late 1980s, when the first statistical machine translation systems were developed.

Some notably successful NLP systems developed in the 1960s were SHRDLU, a natural language system working in restricted "blocks worlds" with restricted vocabularies, and ELIZA, a simulation of a Rogerian psychotherapist, written by Joseph Weizenbaum between 1964 to 1966. Using almost no information about human thought or emotion, ELIZA sometimes provided a startlingly human-like interaction. When the "patient" exceeded the very small knowledge base, ELIZA might provide a generic response, for example, responding to "My head hurts" with "Why do you say your head hurts?‖

During the 70's many programmers began to write 'conceptual ontologies', which structured real-world information into computer- understandable data. Examples are MARGIE (Schank, 1975), SAM (Cullingford, 1978), PAM (Wilensky, 1978), TaleSpin (Meehan, 1976), QUALM (Lehnert, 1977), Politics (Carbonell, 1979), and Plot Units (Lehnert 1981). During this time, many chatterbots were written including PARRY, Racter, and Jabberwacky.

Up to the 1980s, most NLP systems were based on complex sets of hand-written rules. Starting in the late 1980s, however, there was a revolution in NLP with the introduction of machine learning algorithms for language processing. This was due both to the steady increase in computational power resulting from Moore's Law and the gradual lessening of the dominance of Chomskyan theories of linguistics (e.g. transformational grammar), whose theoretical underpinnings discouraged the sort of corpus linguistics that underlies the machine- learning approach to language processing. Some of the earliest-used machine learning algorithms, such as decision trees, produced systems of hard if-then rules similar to existing hand-written rules. Increasingly, however, research has focused on statistical models, which make soft, probabilistic decisions based on attaching real-valued weights to the features making up the input data. Such models are generally more robust when given unfamiliar input, especially input that contains errors (as is very common for real-world data), and produce more reliable results when integrated into a larger system comprising multiple subtasks.

Many of the notable early successes occurred in the field of machine translation, due especially to work at IBM Research, where successively more complicated statistical models were developed. These systems were able to take advantage of existing multilingual textual corpora that had been produced by the Parliament of Canada and the European Union as a result of laws calling for the translation of all governmental proceedings into all official languages of the corresponding systems of government. However, most other systems depended on corpora specifically developed for the tasks implemented by these systems, which was (and often continues to be) a major limitation in the success of these systems. As a result, a great deal of research has gone into methods of more effectively learning from limited amounts of data.

Recent research has increasingly focused on unsupervised and semi- supervised learning algorithms. Such algorithms are able to learn from data that has not been hand-annotated with the desired answers, or using a combination of annotated and non-annotated data. Generally, this task is much more difficult than supervised learning, and typically produces less accurate results for a given amount of input data. However, there is an enormous amount of non-annotated data available (including, among other things, the entire content of the World Wide Web), which can often make up for the inferior results.

NLP Using Machine Learning

As described above, modern approaches to natural language processing (NLP) are grounded in machine learning. The paradigm of machine learning is different from that of most prior attempts at language processing. Prior implementations of language-processing tasks typically involved the direct hand coding of large sets of rules. The machine-learning paradigm calls instead for using general learning algorithms — often, although not always, grounded in statistical inference — to automatically learn such rules through the analysis of large corpora of typical real-world examples. A corpus (plural, "corpora") is a set of documents (or sometimes, individual sentences) that have been hand-annotated with the correct values to be learned.

As an example, consider the task of part of speech tagging, i.e. determining the correct part of speech of each word in a given sentence, typically one that has never been seen before. A

typical machine- learning-based implementation of a part of speech tagger proceeds in two steps, a training step and an evaluation step. The first step — the training step — makes use of a corpus of training data, which consists of a large number of sentences, each of which has the correct part of speech attached to each word. (An example of such a corpus in commonuse is the Penn Treebank. This includes (among other things) a set of
500 texts from the Brown Corpus, containing examples of various genres of text, and 2500 articles from the Wall Street Journal.) This corpus is analyzed and a learning model is generated from it, consisting of automatically created rules for determining the part of speech for a word in a sentence, typically based on the nature of the word in question, the nature of surrounding words, and the most likely part of speech for those surrounding words. The model that is generated is typically the best model that can be found that simultaneously meets two conflicting objectives: To perform as well as possible on the training data, and to be as simple as possible (so that the model avoids over fitting the training data, i.e. so that it generalizes as well as possible to new data rather than only succeeding on sentences that have already been seen). In the second step (the evaluation step), the model that has been learned is used to process new sentences. An important part of the development of any learning algorithm is testing the model that has been learned on new, previously unseen data. It is critical that the data used for testing is not the same as the data used for training; otherwise, the testing accuracy will be unrealistically high.

Many different classes of machine learning algorithms have been applied to NLP tasks. In common to all of these algorithms is that they take as input a large set of "features" that are generated from the input data. As an example, for a part-of-speech tagger, typical features might be the identity of the word being processed, the identity of the words immediately to the left and right, the part-of-speech tag of the word to the left, and whether the word being considered or its immediate neighbors are content words or function words. The algorithms differ, however, in the nature of the rules generated. Some of the earliest-used algorithms, such as decision trees, produced systems of hard if-then rules similar to the systems of hand-written rules that were then common. Increasingly, however, research has focused on statistical models, which make soft, probabilistic decisions based on attaching real- valued weights to each input feature. Such models have the advantage that they can express the relative certainty of many different possible answers rather than only one, producing more reliable results when such a model is included as a component of a larger system. In addition, models that make soft decisions are generally more robust when given unfamiliar input, especially input that contains errors (as is very common for real-world data).

Systems based on machine-learning algorithms have many advantages over hand-produced rules:

- The learning procedures used during machine learning automatically focus on the most common cases, whereas when writing rules by hand it is often not obvious at all where the effortshould be directed.
- Automatic learning procedures can make use of statistical inference algorithms to produce models that are robust to unfamiliar input (e.g. containing words or structures that have not been seen before) and to erroneous input (e.g. with misspelled words or words

accidentally omitted). Generally, handling such input gracefully with hand-written rules — or more generally, creating systems of hand-written rules that make soft decisions
— is extremely difficult and error-prone.

 Systems based on automatically learning the rules can be made more accurate simply by supplying more input data. However, systems based on hand-written rules can only be made more accurate by increasing the complexity of the rules, which is a much more difficult task. In particular, there is a limit to the complexity of systems based on hand-crafted rules, beyond which the systems become more and more unmanageable. However, creating more data to input to machine-learning systems simply requires a corresponding increase in the number of man-hours worked, generally without significant increases in the complexity of the annotation process.

How Natural Language Processing and Machine Learning is Applied

Natural language processing-based solutions comprise language translation, digital speech recognition, emotive or sentiment analysis, question and answer online systems, smart chatbot systems, automatic text summarization capabilities, financial market intelligence, automatic text categorization, and automatic language grammar scanning.
These technologies help both individuals and organizations to analyze their data, uncover new insights, automate time and labor-consuming processes and gain competitive advantages.

Language Translation

Translating languages is a far more intricate process than simply translating using word-to-word replacement techniques. Each language has its own unique grammar rules and limitations. The challenge of translating any language passage or digital text is to perform this process without changing the underlying style or meaning. As computer systems cannot explicitly understand grammar, they require a specific program to dismantle a sentence, then reassemble using another language in a manner that makes sense to humans.
Google Translate is such a tool, a well-known online language translation service. Previously Google Translate used a Phrase-Based Machine Translation, which scrutinized a passage for similar phrases between dissimilar languages. Presently, Google Translate uses the Google Neural Machine Translation instead, which uses machine learning and natural language processing algorithms to search for language patterns.

Speech Recognition Activities

Speech recognition capabilities are a smart machine's capability to recognize and interpret specific phrases and words from a spoken language and transform them into machine-readable formats. It uses natural language processing algorithms to allow computers to imitate human interactions, and machine language methods to reply, therefore mimicking human responses.
Google Now, Siri, and Alexa are a few of the most popular models utilizing speech recognition technology. By simply saying 'call Fred', a smartphone mobile device will recognize what that personal command represents and will then create a call to the personal contact saved as Fred.

Emotion and Sentiment Analysis

Sentiment or emotive analysis uses both natural language processing and machine learning to decode and analyze human emotions within subjective data such as news articles and

influencer tweets. Positive, adverse, and impartial viewpoints can be readily identified to determine the consumer's feelings towards a product, brand, or a specific service. Automatic sentiment analysis is employed to measure public or customer opinion, monitor a brand's reputation, and further understand a customer's overall experience.
Financial markets are sensitive domains heavily influenced by human sentiment and emotion. Negative presumptions can lead to stock prices dropping, while positive sentiment could trigger investors to purchase more of a company's stock, thereby causing share prices to rise.

Online Smart Chatbots
Online chatbots are computer programs that provide 'smart' automated explanations to common consumer queries. They contain automated pattern recognition systems with a rule-of-thumb response mechanism. They are used to conduct worthwhile and meaningful conversations with people interacting with a particular website. Initially, chatbots were only used to answer fundamental questions to minimize call center volume calls and deliver swift customer support services.
However, nowadays, AI-powered chatbots are developed to manage more complicated consumer requests making conversational experiences somewhat intuitive. For example, chatbots within healthcare systems can collect personal patient data, help patients evaluate their symptoms, and determine the appropriate next steps to take. Additionally, these healthcare chatbots can arrange prompt medical appointments with the most suitable medical practitioners, and even suggest worthwhile treatments to partake.

Intelligent Question and Answer Systems
Question and answer computer systems are those intelligent systems used to provide specific answers to consumer queries. Besides chatbots, question and answer systems have a large array of stored knowledge and practical language understanding algorithms - rather than simply delivering 'pre-canned' generic solutions. These systems can answer questions like 'When did Winston Churchill first become the British Prime Minister?' or 'How do I proceed on foot to Paris Gare du Nord?'. These intelligent responses are created with meaningful textual data, along with accompanying audio, imagery, and video footage.
Question and answer smart systems are found within social media chatrooms using intelligent tools such as IBM's Watson.

Automatic Text Condensing and Summarisation
Automatic text condensing and summarization processes are those tasks used for reducing a portion of text to a more succinct and more concise version. This process happens by extracting the main concepts and preserving the precise meaning of the content. This application of natural language processing is used to create the latest news headlines, sports result snippets via a webpage search and newsworthy bulletins of key daily financial market reports.

Financial Market Intelligence
Financial market intelligence gathers valuable insights covering economic trends, consumer spending habits, financial product movements along with their competitor information. Such extractable and actionable information is used by senior business leaders for strategic decision-making and product positioning. Market intelligence systems can analyze current

financial topics, consumer sentiments, aggregate, and analyze economic keywords and intent. All processes are within a structured data format that can be produced much quicker than traditional desk and data research methods.
By utilizing market intelligence services, organizations can identify those end-user search queries that are both current and relevant to the marketplace, and add contextually appropriate data to the search results. As a result, it can provide meaningful information to help those organizations decide which of their services and products to discontinue or what consumers are currently targeting.

Automatic Text Classification
The process required for automatic text classification is another elemental solution of natural language processing and machine learning. It is the procedure of allocating digital tags to data text according to the content and semantics. This process allows for immediate, effortless data retrieval within the searching phase. This machine learning application can also differentiate spam and non-spam email content over time.

Automatic Grammar Checking and Suggestive Alternatives
Automatic grammar checking, which is the task of noticing and remediating grammatical language errors and spelling mistakes within the text, is another prominent component of NLP-ML systems. Auto-grammar checking processes will visually warn stakeholders of a potential error by underlining an identified word in red.

Major tasks in NLP

The following is a list of some of the most commonly researched tasks in NLP. Note that some of these tasks have direct real-world applications, while others more commonly serve as subtasks that are used to aid in solving larger tasks. What distinguishes these tasks from other potential and actual NLP tasks is not only the volume of research devoted to them but the fact that for each one there is typically a well-defined problem setting, a standard metric for evaluating the task, standard corpora on which the task can be evaluated, and competitions devoted to the specific task.

- **Automatic summarization:** Produce a readable summary of a chunk of text. Often used to provide summaries of text of a known type, such as articles in the financial section of a newspaper.
- **Co reference resolution:** Given a sentence or larger chunk of text, determine which words ("mentions") refer to the same objects ("entities"). Anaphora resolution is a specific example of this task, and is specifically concerned with matching up pronouns with the nouns or names that they refer to. The more general task of co reference resolution also includes identify so- called "bridging relationships" involving referring expressions. For example, in a sentence such as "He entered John's house through the front door", "the front door" is a referring expression and the bridging relationship to be identified is the fact that the door being referred to is the front door of John's house (rather than of some other structure that might also be referred to).
- **Discourse analysis:** This rubric includes a number of related tasks. One task is identifying the discourse structure of connected text, i.e. the nature of the discourse

relationships between sentences (e.g. elaboration, explanation, contrast). Another possible task is recognizing and classifying the speech acts in a chunk of text (e.g. yes-no question, content question, statement, assertion, etc.).

- **Machine translation:** Automatically translate text from one human language to another. This is one of the most difficult problems, and is a member of a class of problems colloquially termed "AI-complete", i.e. requiring all of the different types of knowledge that humans possess (grammar, semantics, facts about the real world, etc.) in order to solve properly.
- **Morphological segmentation:** Separate words into individual morphemes and identify the class of the morphemes. The difficulty of this task depends greatly on the complexity of the morphology (i.e. the structure of words) of the language being considered. English has fairly simple morphology, especially inflectional morphology, and thus it is often possible to ignore this task entirely and simply model all possible forms of a word (e.g. "open, opens, opened, and opening") as separate words. In languages such as Turkish, however, such an approach is not possible, as each dictionary entry has thousands of possible word forms.
- **Named entity recognition (NER):** Given a stream of text, determine which items in the text map to proper names, such as people or places, and what the type of each such name is (e.g. person, location, organization). Note that, although capitalization can aid in recognizing named entities in languages such as English, this information cannot aid in determining the type of named entity, and in any case is often inaccurate or insufficient. For example, the first word of a sentence is also capitalized, and named entities often span several words, only some of which are capitalized. Furthermore, many other languages in non-Western scripts (e.g. Chinese or Arabic) do not have any capitalization at all, and even languages with capitalization may not consistently use it to distinguish names. For example, German capitalizes all nouns, regardless of whether they refer to names, and French and Spanish do not capitalize names that serve as adjectives.
- **Natural language generation:** Convert information from computer databases into readable human language.
- **Natural language understanding:** Convert chunks of text into more formal representations such as first-order logic structures that are easier for computer programs to manipulate. Natural language understanding involves the identification of the intended semantic from the multiple possible semantics which can be derived from a natural language expression which usually takes the form of organized notations of natural languages concepts. Introduction and creation of language metamodel and ontology are efficient however empirical solutions. An explicit formalization of natural languages semantics without confusions with implicit assumptions such as closed world assumption (CWA) vs. open world assumption, or subjective Yes/No vs. objective True/False is expected for the construction of a basis of semantics formalization.
- **Optical character recognition (OCR):** Given an image representing printed text, determine the corresponding text.
- **Part-of-speech tagging:** Given a sentence, determine the part of speech for each word. Many words, especially common ones, can serve as multiple parts of speech. For example, "book" can be a noun ("the book on the table") or verb ("to book a flight"); "set" can be a noun, verb or adjective; and "out" can be any of at least five different parts of speech. Note that some languages have more such ambiguity than others. Languages with little

inflectional morphology, such as English are particularly prone to such ambiguity. Chinese is prone to such ambiguity because it is a tonal language during verbalization. Such inflection is not readily conveyed via the entities employed within the orthography to convey intended meaning.

- **Parsing:** Determine the parse tree (grammatical analysis) of a given sentence. The grammar for natural languages is ambiguous and typical sentences have multiple possible analyses. In fact, perhaps surprisingly, for a typical sentence there may be thousands of potential parses (most of which will seem completely nonsensical to a human).
- **Question answering:** Given a human-language question, determine its answer. Typical questions have a specific right answer (such as "What is the capital of Canada?"), but sometimesopen-ended questions are also considered (such as "What is the meaning of life?").
- **Relationship extraction:** Given a chunk of text, identify the relationships among named entities (e.g. who is the wife of whom).
- **Sentence breaking (also known as sentence boundary disambiguation):** Given a chunk of text, find the sentence boundaries. Sentence boundaries are often marked by periods or other punctuation marks, but these same characters can serve other purposes (e.g. marking abbreviations).
- **Sentiment analysis:** Extract subjective information usually from a set of documents, often using online reviews to determine "polarity" about specific objects. It is especially useful for identifying trends of public opinion in the social media, for the purpose of marketing.
- **Speech recognition:** Given a sound clip of a person or people speaking, determine the textual representation of the speech. This is the opposite of text to speech and is one of the extremely difficult problems colloquially termed "AI-complete" (see above). In natural speech there are hardly any pauses between successive words, and thus speech segmentation is a necessary subtask of speech recognition (see below). Note also that in most spoken languages, the sounds representing successive letters blend into each other in a process termed coarticulation, so the conversion of the analog signal to discrete characters can be a very difficult process.
- **Speech segmentation:** Given a sound clip of a person or people speaking, separate it into words. A subtask of speech recognition and typically grouped with it.
- Topic segmentation and recognition: Given a chunk of text, separate it into segments each of which is devoted to a topic, and identify the topic of the segment.
- **Word segmentation:** Separate a chunk of continuous text into separate words. For a language like English, this is fairly trivial, since words are usually separated by spaces. However, some written languages like Chinese, Japanese and Thai do not mark word boundaries in such a fashion, and in those languages text segmentation is a significant task requiring knowledge of thevocabulary and morphology of words in the language.
- **Word sense disambiguation:** Many words have more than one meaning; we have to select the meaning which makes the most sense in context. For this problem, we are typically given a list of words and associated word senses, e.g. from a dictionary or from an online resource such as WordNet.

In some cases, sets of related tasks are grouped into subfields of NLP that are often considered separately from NLP as a whole. Examples include:

- **Information retrieval (IR):** This is concerned with storing, searching and retrieving information. It is a separate field within computer science (closer to databases), but IR relies on some NLP methods (for example, stemming). Some current research and applications seek to bridge the gap between IR and NLP.
- **Information extraction (IE):** This is concerned in general with the extraction of semantic information from text. This covers tasks such as named entity recognition, coreference resolution, relationship extraction, etc.
- **Speech processing:** This covers speech recognition, text-to- speech and related tasks.

Other tasks include:

- Stemming
- Text simplification
- Text-to-speech
- Text-proofing
- Natural language search
- Query expansion
- Truecasing

Statistical Natural Language Processing

Statistical natural-language processing uses stochastic, probabilistic and statistical methods to resolve some of the difficulties discussed above, especially those which arise because longer sentences are highly ambiguous when processed with realistic grammars, yielding thousands or millions of possible analyses. Methods for disambiguation often involve the use of corpora and Markov models. Statistical NLP comprises all quantitative approaches to automated language processing, including probabilistic modeling, information theory, and linear algebra. The technology for statistical NLP comes mainly from machine learning and data mining, both of which are fields of artificial intelligence that involve learning from data.

7.0 7-42.

CHAPTER 5
EXPERT SYSTEM

INTRODUCTION

Expert system is a computer program that uses artificial intelligence to solve problems within a specialized domain that ordinarily requires human expertise. The first expert system was developed in 1965 by Edward Feigenbaum and Joshua Lederberg of Stanford University in California, U.S. Dendral, as their expert system was later known, was designed to analyze chemical compounds. Expert systems now have commercial applications in fields as diverse as medical diagnosis, petroleum engineering, financial investing make financial forecasts and schedule routes for delivery vehicles.

What is an Expert System?

An expert system is a computer program that is designed to solve complex problems and to provide decision-making ability like a human expert. It performs this by extracting knowledge from its knowledge base using the reasoning and inference rules according to the user queries.

The expert system is a part of AI, and the first ES was developed in the year 1970, which was the first successful approach of artificial intelligence. It solves the most complex issue as an expert by extracting the knowledge stored in its knowledge base. The system helps in decision making for compsex problems using both facts and heuristics like a human expert. It is called so because it contains the expert knowledge of a specific domain and can solve any complex problem of that particular domain. These systems are designed for a specific domain, such as medicine, science, etc.

The performance of an expert system is based on the expert's knowledge stored in its knowledge base. The more knowledge stored in the KB, the more that system improves its performance. One of the common examples of an ES is a suggestion of spelling errors while typing in the Google search box.

Below are some popular examples of the Expert System:

- **DENDRAL:** It was an artificial intelligence project that was made as a chemical analysis expert system. It was used in organic chemistry to detect unknown organic molecules with the help of their mass spectra and knowledge base of chemistry.
- **MYCIN:** It was one of the earliest backward chaining expert systems that was designed to find the bacteria causing infections like bacteraemia and meningitis. It was also used for the recommendation of antibiotics and the diagnosis of blood clotting diseases.

- **PXDES:** It is an expert system that is used to determine the type and level of lung cancer. To determine the disease, it takes a picture from the upper body, which looks like the shadow. This shadow identifies the type and degree of harm.
- **CaDeT:** The CaDet expert system is a diagnostic support system that can detect cancer at early stages.

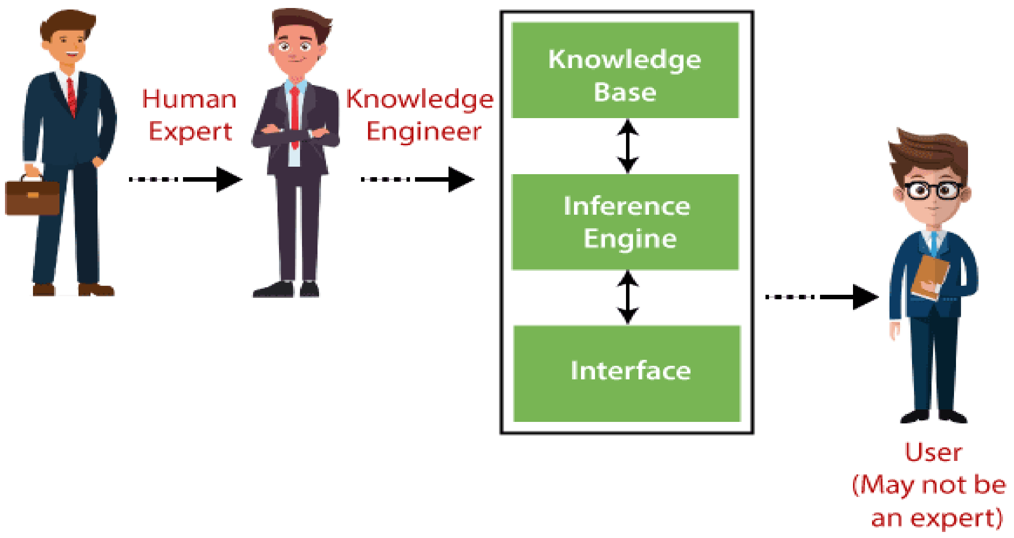
Characteristics of Expert System

- **High Performance:** The expert system provides high performance for solving any type of complex problem of a specific domain with high efficiency and accuracy.
- **Understandable:** It responds in a way that can be easily understandable by the user. It can take input in human language and provides the output in the same way.
- **Reliable:** It is much reliable for generating an efficient and accurate output.
- **Highly responsive:** ES provides the result for any complex query within a very short period of time.

Components of Expert System

An expert system mainly consists of three components:

- User Interface
- Inference Engine
- Knowledge Base



1. User Interface

With the help of a user interface, the expert system interacts with the user, takes queries as an input in a readable format, and passes it to the inference engine. After getting the response from the inference engine, it displays the output to the user. In other words, **it is an interface that helps a non-expert user to communicate with the expert system to find a solution**.

2. Knowledge Base

A knowledge base (abbreviated KB, kb or Δ) is a special kind of database for knowledge management, providing the means for the computerized collection, organization, and retrieval of knowledge. Also, it is a collection of data representing related experiences which their results is related to their problems and solutions.

Facts for a knowledge base must be acquired from human experts through interviews and observations. This knowledge is then usually represented in the form of ―if-then‖ rules (production rules): ―If some condition is true then the following inference can be made (or some action taken).‖ The knowledge base of a major expert system includes thousands of rules. A probability factor is often attached to the conclusion of each production rule, because the conclusion is not a certainty. For example, a system for the diagnosis of eye diseases might indicate, based on information supplied to it, a 90 percent probability that a person has glaucoma, and it might also list conclusions with lower probabilities. An expert system may display the sequence of rules through which it arrived at its conclusion; tracing this flow helps the

user to appraise the credibility of its recommendation and is useful as a learning tool for students.

Human experts frequently employ heuristic rules, or ―rules of thumb,‖ in addition to simple production rules. For example, a credit manager might know that an applicant with a poor credit history, but a clean record since acquiring a new job, might actually be a good credit risk. Expert systems have incorporated such heuristic rules and increasingly have the ability to learn from experience. Nevertheless, expert systems remain supplements, rather than replacements, for human experts.

Types Of Knowledge Base

Knowledge bases are essentially closed or open information repositories and can be categorized under two main headings:

 Machine-readable knowledge bases store knowledge in a computer-readable form, usually for the purpose of having automated deductive reasoning applied to them. They contain a set of data, often in the form of rules that describe the knowledge in a logically consistent manner. An ontology can define the structure of stored data - what types of entities are recorded and what their relationships are. Logical operators, such as *And* (conjunction), *Or* (disjunction), *material implication* and *negation* may be used to build it up from simpler pieces of information. Consequently, classical deduction can be used to reason about the

knowledge in the knowledge base. Some machine-readable knowledge bases are used with artificial intelligence, for example as part of an expert system that focuses on a domain like prescription drugs or customs law. Such knowledge bases are also used by the semantic web.
 Human-readable knowledge bases are designed to allow people to retrieve and use the knowledge they contain. They are commonly used to complement a help desk or for sharing information among employees within an organization. They might store troubleshooting information, articles, white papers, user manuals, knowledge tags, or answers to frequently asked questions. Typically, a search engine is used to locate information in the system, or users may browse through a classification scheme.

A text based system that can include groups of documents including hyperlinks between them is known as Hypertext Systems. Hypertext systems support the decision process by relieving the user of the significant effort it takes to relate and remember things." Knowledge bases can exist on both computers and mobile phones in a hypertext format. Knowledge base analysis and design (also known as KBAD) is an approach that allows people to conduct analysis and design in a way that result in a knowledge base, which can later be used to make informative decisions. This approach was first implemented by Dr. Steven H. Dam

3 Inference Engine

In computer science, and specifically the branches of knowledge engineering and artificial intelligence, an inference engine is a computer program that tries to derive answers from a knowledge base. It is the "brain" that expert systems use to reason about the information in the knowledge base for the ultimate purpose of formulating new conclusions. Inference engines are considered to be a special case of reasoning engines, which can use more general methods of reasoning.

Architecture

The separation of inference engines as a distinct software component stems from the typical production system architecture. This architecture relies on a data store:

1. An interpreter. The interpreter executes the chosen agenda items by applying the corresponding base rules.
2. A scheduler. The scheduler maintains control over the agenda by estimating the effects of applying inference rules in light of item priorities or other criteria on the agenda.
3. A consistency enforcer. The consistency enforcer attempts to maintain a consistent representation of the emerging solution

The Recognize-Act Cycle

The inference engine can be described as a form of finite state machine with a cycle consisting of three action states: *match rules*, *select rules*, and *execute rules*. Rules are represented in the system by a notation called predicate logic.

In the first state, match rules, the inference engine finds all of the rules that are satisfied by the current contents of the data store. When rules are in the typical *condition-action* form, this means testing the conditions against the working memory. The rule matching that are found are all candidates for execution: they are collectively referred to as the *conflict set*. Note that the same rule may appear several times in the conflict set if it matches different subsets of data items. The pair of a rule and a subset of matching data items are called an *instantiation* of the rule.
In many applications, where large volumes of data are concerned and/or when performance time considerations are critical, the computation of the conflict set is a non-trivial problem.

Data-Driven Computation versus Procedural Control

The inference engine control is based on the frequent re - evaluation of the data store states, not on any static control structure of the program. The computation is often qualified as *data-driven* or *pattern-directed* in contrast to the more traditional procedural control. Rules can communicate with one another only by way of the data, whereas in traditional programming languages procedures and functions explicitly call one another. Unlike instructions, rules are not executed sequentially and it is not always possible to determine through inspection of a set of rules which rule will be executed first or cause the inference engine to terminate.

In contrast to a procedural computation, in which knowledge about the problem domain is mixed in with instructions about the flow of control—although object-oriented programming languages mitigate this entanglement—the inference engine model allows a more complete separation of the knowledge (in the rules) from the control (the inference engine).

Inference Rules

An inference rule is a conditional statement with two parts namely; if clause and a then clause.

This rule is what gives expert systems the ability to find solutions to diagnostic and prescriptive problems. An example of an inference rule is:

If the restaurant choice includes French and the occasion is romantic, Then the restaurant choice is definitely Paul Bocuse.

An expert system's rule base is made up of many such inference rules. They are entered as separate rules and it is the inference engine that uses them together to draw conclusions. Because each rule is a unit, rules may be deleted or added without affecting other rules - though it should affect which conclusions are reached. One advantage of inference rules over traditional programming is that inference rules use reasoning which more closely resembles human reasoning.

Thus, when a conclusion is drawn, it is possible to understand how this conclusion was reached. Furthermore, because the expert system uses knowledge in a form similar to the that of the expert, it may be easier to retrieve this information directly from the expert.

Chaining

Two methods of reasoning when using inference rules are forward chaining and backward chaining.

Forward chaining starts with the data available and uses the inference rules to extract more data until a desired goal is reached. An inference engine using forward chaining searches the inference rules until it finds one in which the if clause is known to be true . It then concludes the then clause and adds this information to its data. It continues to do this until a goal is reached. Because the data available determines which inference rules are used, this method is also classified as data driven.

Backward chaining starts with a list of goals and works backwards to see if there is data which will allow it to conclude any of these goals. An inference engine using backward chaining would search the inference rules until it finds one which has a then clause that matches a desired goal. If the if clause of that inference rule is not known to be true, then it is added to the list of goals. For example, suppose a rule base contains:

(1) IF X is green THEN X is a frog. (Confidence Factor: +1%)
(2) IF X is NOT green THEN X is NOT a frog. (Confidence Factor: +99%)
(3) IF X is a frog THEN X hops. (Confidence Factor: +50%)
(4) IF X is NOT a frog THEN X does NOT hop. (Confidence Factor +50%)

Suppose a goal is to conclude that Fritz hops. Let X = "Fritz". The rule base would be searched and rule (3) would be selected because its conclusion (the then clause) matches the goal. It is not known that Fritz is a frog, so this "if" statement is added to the goal list. The rule base is again searched and this time rule (1) is selected because its then clause matches the new goal just added to the list. This time, the if clause (Fritz is green) is known to be true and the goal that Fritz hops is concluded. Because the list of goals determines which rules are selected and used, this method is called goal driven.

However, note that if we use confidence factors in even a simplistic fashion - for example, by multiplying them together as if they were like soft probabilities - we get a result that is known with a confidence factor of only one-half of 1%. (This is by multiplying 0.5 x 0.01 = 0.005). This is useful, because without confidence factors, we might erroneously conclude with certainty that a sea turtle named Fritz hops just by virtue of being green. In Classical logic or Aristotelian term logic systems, there are no probabilities or confidence factors; all facts are regarded as certain. An ancient example from Aristotle states, "Socrates is a man. All men are mortal. Thus Socrates is mortal."

In real world applications, few facts are known with absolute certainty and the opposite of a given statement may be more likely to be true ("Green things in the pet store are not frogs, with the probability or confidence factor of 99% in my pet store survey"). Thus it is often useful when building such systems to try and prove both the goal and the opposite of a given goal to see which is more likely.

Certainty Factors

One method of operation of expert systems is through a quasi- probabilistic approach with certainty factors: A human, when reasoning, does not always make statements with 100% confidence: he might venture, "If Fritz is green, then he is probably a frog" (after all, he might be a chameleon). This type of reasoning can be imitated using numeric values called confidences. For example, if it is known that Fritz is green, it might be concluded with 0.85 confidence that he is a frog; or, if it is known that he is a frog, it might be concluded with 0.95 confidence that he hops. These Certainty factor (CF) numbers quantify uncertainty in the degree to which the available evidence supports a hypothesis. They represent a degree of confirmation, and are not probabilities in a Bayesian sense. The CF calculus, developed by Shortliffe & Buchanan, increases or decreases the CF associated with a hypothesis as each new piece of evidence becomes available. It can be mapped to a probability update, although degrees of confirmation are not expected to obey the laws of probability. It is important to note, for example, that evidence for hypothesis H may have nothing to contribute to the degree to which Noth is confirmed or disconfirmed (e.g., although a fever lends some support to a diagnosis of infection, fever does not disconfirm alternative hypotheses) and that the sum of CFs of many competing hypotheses may be greater than one (i.e., many hypotheses may be well confirmed based on available evidence).

The CF approach to a rule-based expert system design does not have a widespread following, in part because of the difficulty of meaningfully assigning CFs a priori. (The above example of green creatures being likely to be frogs is excessively naive.) Alternative approaches to quasi- probabilistic reasoning in expert systems involve fuzzy logic, which has a firmer mathematical foundation. Also, rule-engine shells such as Drools and Jess do not support probability manipulation: they use an alternative mechanism called salience, which is used to prioritize the order of evaluation of activated rules.

In certain areas, as in the tax-advice scenarios discussed below, probabilistic approaches are not acceptable. For instance, a 95% probability of being correct means a 5% probability of being wrong. The rules that are defined in such systems have no exceptions: they are only a means of achieving software flexibility when external circumstances change frequently. Because rules are stored as data, the core software does not need to be rebuilt each time changes to federal and state tax codes are announced.

Real-Time Adaption

Industrial processes, data networks, and many other systems change their state and even their structure over time. Real time expert systems are designed to reason over time and change conclusions as the monitored system changes. Most of these systems must respond to constantly changing input data, arriving automatically from other systems such as process control systems or network management systems.

Representation includes features for defining changes in belief of data or conclusions over time. This is necessary because data becomes stale. Approaches to this can include decaying belief functions, or the simpler validity interval that simply lets data and conclusions expire after specified time period, falling to "unknown" until refreshed. An often- cited example (attributed to real time expert system pioneer Robert L. Moore) is a hypothetical expert system that might be used to drive a car. Based on video input, there might be an intermediate conclusion that a stop light is green and a final conclusion that it is OK to drive through the intersection. But that data and the subsequent conclusions have a very limited lifetime. You would not want to be a passenger in a car driven based on data and conclusions that were, say, an hour old.

The inference engine must track the times of each data input and each conclusion, and propagate new information as it arrives. It must ensure that all conclusions are still current. Facilities for periodically scanning data, acquiring data on demand, and filtering noise, become essential parts of the overall system. Facilities to reason within a fixed deadline are important in many of these applications.

An overview of requirements for a real-time expert system shell is given in. Examples of real time expert system applications are given in and.

Ability to Make Relevant Inquiries

An additional skill of an expert system is the ability to give relevant inquiries based on previous input from a human user, in order to give better replies or other actions, as well as working faster, which also pleases an impatient or busy human user - it allows a priori volunteeringof information that the user considers important.

Also, the user may choose not to respond to every question, forcing the expert system to function in the presence of partial information.

Commercially viable systems will try to optimize the user experience by presenting options for commonly requested information based on a history of previous queries of the system using technology such as forms, augmented by keyword-based search. The gathered information may be verified by a confirmation step (e.g., to recover from spelling mistakes), and now act as an input into a forward-chaining engine. If confirmatory questions are asked in a subsequent phase, based on the rules activated by the obtained information, they are more likely to be specific and relevant. Such abilities can largely be achieved by control flow structures.

In an expert system, implementing the ability to learn from a stored history of its previous use involves employing technologies considerably different from that of rule engines, and is considerably more challenging from a software-engineering perspective. It can, however, make the difference between commercial success and failure. A large part of the revulsion that users felt towards Microsoft's Office Assistant was due to the extreme naivete of its rules ("It looks like you are typing a letter: would you like help?") and its failure to adapt to the user's level of expertise over time (e.g. a user who regularly uses features such as Styles, Outline view, Table of Contents or cross-references is unlikely to be a beginner who needs help writing a letter).

Explanation System

Another major distinction between expert systems and traditional systems is illustrated by the following answer given by the system when the user answers a question with another question, "Why", as occurredin the above example. The answer is:

A. I am trying to determine the type of restaurant to suggest. So far Indian is not a likely choice. It is possible that French is a likely choice. If I know that if the diner is a wine drinker, and the preferred wine is French, then there is strong evidence that the restaurant choice should include French.

It is very difficult to implement a general explanation system (answering questions like "Why" and "How") in a traditional computer program. An expert system can generate an explanation by retracing the steps of its reasoning. The response of the expert system to the question "Why" exposes the underlying knowledge structure. It is a rule; a set of antecedent conditions which, if true, allow the assertion of a consequent. The rule references values, and tests them against various constraints or asserts constraints onto them. This, in fact, is a significant part of the knowledge structure. There are values, which may be associated with some organizing entity. For example, the individual diner is an entity with various attributes (values) including whether they drink wine and the kind of wine. There are also rules, which associate the currently known values of some attributes with assertions that can be made about other attributes. It is the orderly processing of these rules that dictates the dialogue itself.

Knowledge Engineering

The building, maintaining and development of expert systems are known as knowledge engineering. Knowledge engineering is a "discipline that involves integrating knowledge into computer systems in order to solve complex problems normally requiring a high level of human.

There are generally three individuals having an interaction in an expert system. Primary among these is the end-user, the individual who uses the system for its problem solving assistance. In the construction and maintenance of the system there are two other roles: the problem domain expert who builds the system and supplies the knowledge base, and a knowledge engineer who assists the experts in determining the representation of their knowledge, enters this knowledge into an explanation module and who defines the inference technique required to solve the problem. Usually the knowledge engineer will represent the problem solving activity in the form of rules. When these rules are created from domain expertise, the knowledge base stores the rules of the expert system.

General Types of Problems Solved

Expert systems are most valuable to organizations that have a high-level of know-how experience and expertise that cannot be easily transferred to other members. They are designed to carry the intelligence and information found in the intellect of experts and provide this knowledge to other members of the organization for problem-solving purposes.

Typically, the problems to be solved are of the sort that would normally be tackled by a professional, such as a medical professional in the case of clinical decision support systems. Real experts in the problem domain (which will typically be very narrow, for instance "diagnosing skin conditions in teenagers") are asked to provide "rules of thumb" on how they evaluate the problem — either explicitly with the aid of experienced systems developers, or sometimes implicitly, by getting such experts to evaluate test cases and using computer programs to examine the test data and derive rules from that (in a strictly limited manner). Generally, expert systems are used for problems for which there is no single "correct" solution which can be encoded in a conventional algorithm — one would not write an expert system to find the shortest paths through graphs, or to sort data, as there are simpler ways to do these tasks.

Simple systems use simple true/false logic to evaluate data. More sophisticated systems are capable of performing at least some evaluation, taking into account real-world uncertainties, using such methods as fuzzy logic. Such sophistication is difficult to develop and still highly imperfect.

Different Types of Expert System are

- Rule-Based expert system
- Frames-Based expert system
- Hybird system
- Model-based expert system
- Ready-made system
- Real-Time expert system

Examples of Applications

Expert systems are designed to facilitate tasks in the fields of accounting, medicine, process control, financial service, production, human resources among others. Typically, the problem area is complex enough that a more simple traditional algorithm cannot provide a proper solution, The foundation of a successful expert system depends on a series of technical procedures and development that may be designed by technicians and related experts. As such, expert systems do not typically provide a definitive answer, but provide probabilistic recommendations. An example of the application of expert systems in the financial field is expert systems for mortgages. Loan departments are interested in expert systems for mortgages because of the growing cost of labour, which makes the handling and acceptance of relatively small loans less profitable. They also see a possibility for standardised, efficient handling of mortgage loan by applying expert systems, appreciating that for the acceptance of mortgages there are hard and fast rules which do not always exist with other types of loans. Another common application in the financial area for expert systems is in trading recommendations in various marketplaces. These markets involve numerous variables andhuman emotions which may be impossible to deterministically characterize, thus expert systems based on the rules of thumb from experts and simulation data are used. Expert system of this type can range from ones providing regional retail recommendations, like Wishabi, to ones used to assist monetary decisions by financial institutions and governments. Another 1970s and 1980s application of expert systems, which we today would simply call AI, was in computer games. For example, the computer baseball games Earl Weaver Baseball and Tony La Russa Baseball each had highly detailed simulations of the game strategies of those two baseball managers. When a human played the game against the computer, the computer queried the Earl Weaver or Tony La Russa Expert System for a decision on what strategy to follow. Even those choices where some randomness was part of the natural system (such as when to throw a surprise pitch-out to try to trick a runner trying to steal a base) were decided based on probabilities supplied by Weaver or La Russa. Today we would simply say that "thegame's AI provided the opposing manager's strategy."

Advantages

- Compared to traditional programming techniques, expert-system approaches provide the added flexibility (and hence easier modifiability) with the ability to model rules as data rather than as code. In situations where an organization's IT department is overwhelmed by a software-development backlog, rule-engines, by facilitating turnaround, provide a means that can allow organizations to adapt more readily to changing needs.
- In practice, modern expert-system technology is employed as an adjunct to traditional programming techniques, and this hybrid approach allows the combination of the strengths of both approaches. Thus, rule engines allow control through programs (and user interfaces) written in a traditional language, and also incorporate necessary functionality such as inter-operability with existing database technology.

Disadvantages

The Garbage In, Garbage Out (GIGO) phenomenon: A system that uses expert-system technology provides no guarantee about the quality of the rules on which it operates. All self-designated "experts" are not necessarily so, and one notable challenge in expert system design is in getting a system to recognize the limitsto its knowledge.

Expert systems are notoriously narrow in their domain of knowledge— as an amusing example, a researcher used the "skin disease" expert system to diagnose his rust bucket car as likely to have developed measles — and the systems are thus prone to making errors that humans would easily spot. Additionally, once some of the mystique had worn off, most programmers realized that simple expert systems were essentially just slightly more elaborate versions of the decision logic they had already been using. Therefore, some of the techniques of expert systems can now be found in most complex programs without drawing much recognition.

- An expert system or rule-based approach is not optimal for all problems, and considerable knowledge is required so as to not misapply the systems.
- Ease of rule creation and rule modification can be double-edged. A system can be sabotaged by a non-knowledgeable user who can easily add worthless rules or rules that conflict with existing ones. Reasons for the failure of many systems include the absence of (or neglect to employ diligently) facilities for system audit, detection of possible conflict, and rule lifecycle management (e.g. version control, or thorough testing before deployment). The problems to be addressed here are as much technological as organizational.

An example and a good demonstration of the limitations of an expert system is the Windows operating system troubleshooting software located in the "help" section in the taskbar menu. Obtaining technical operating system support is often difficult for individuals not closely involved with the development of the Operating System. Microsoft has designed their expert system to provide solutions, advice, and suggestions to common errors encountered while using their operating systems.

CONCLUSION

Expert systems now have commercial applications in fields as diverse as medical diagnosis, petroleum engineering, financial investing make financial forecasts and schedule routes for delivery vehicles.

CHAPTER 5
PATERN RECOGNITION

What Is Pattern Recognition?

Pattern recognition is a data analysis process that uses machine learning algorithms to classify input data into objects, classes, or categories based on recognized patterns, features, or regularities in data. It has several applications in the fields of astronomy, medicine, robotics, and satellite remote sensing, among others. Pattern recognition involves two primary classification methods:

- **Supervised classification:** In a supervised pattern recognition method, a human trains a computer algorithm to recognize patterns based on predefined labeled datasets. Once the pattern is identified, the method subsequently classifies new data.

- **Unsupervised classification:** In unsupervised classification, the model learns independently without any direct guidance from a human. The computer algorithm identifies correlations between multiple data elements (inputs) based on their similarity score and performs data classification.

Pattern recognition is implemented via several approaches. While it is difficult to decide on one particular approach to perform recognition tasks, we'll discuss six popular methods commonly used by professionals and businesses for pattern recognition.

Methods of Pattern Recognition

1. Statistical pattern recognition: This pattern recognition approach uses historical statistical data that learns from patterns and examples. The method collects observations and processes them to define a model. This model then generalizes over the collected observations and applies the rules to new datasets or examples.

2. Syntactic pattern recognition: Syntactic pattern recognition involves complex patterns that can be identified using a hierarchical approach. Patterns are established based on the way primitives (e.g., letters in a word) interact with each other. An example of this could be how primitives are assembled in words and sentences. Such training samples will enable the development of grammatical rules that demonstrate how sentences will be read in the future.

3. Neural pattern recognition: This method uses artificial neural networks (ANN) and learns from complex and non-linear input/output relations, adapts to data, and detects patterns. The most popular and effective method in neural networks is the feed-forward method. In this method, learning happens by giving feedback to input patterns. This is much like humans learning from their past experiences and mistakes. The ANN-based model is rated as the most expensive pattern recognition method compared to other methods due to the computing resources involved in the process.

4. Template matching: Template matching is one of the simplest of all pattern recognition approaches. Here, the similarity between two entities is determined by matching the sample with the reference template. Such methods are typically used in digital image processing, where small sections of an image are matched to a stored template image. Some of its real-world examples include medical image processing, face recognition, and robot navigation.

5. Fuzzy-based approach: In the fuzzy approach, a set of patterns are partitioned based on the similarity in the features of the patterns. When the unique features of a pattern are correctly detected, data can be easily classified into that known feature space. Even the human visual system sometimes fails to recognize certain components despite scanning objects for a long time. The same holds true for the digital world, where algorithms cannot figure out the exact nature of an object. Hence, the fuzzy approach aims to classify objects based on several similar features in the detected patterns.

6. Hybrid approach: A hybrid approach employs a combination of the above methods to take advantage of all these methods. It employs multiple classifiers to detect patterns where each classifier is trained on a specific feature space. A conclusion is drawn based on the results accumulated from all the classifiers.

How Does Pattern Recognition Work?

Pattern recognition is applied for data of all types, including image, video, text, and audio. As the pattern recognition model can identify recurring patterns in data, predictions made by such models are quite reliable.

Pattern recognition involves three key steps: analyzing the input data, extracting patterns, and comparing it with the stored data. The process can be further divided into two phases:

1. **Explorative phase:** In this phase, computer algorithms tend to explore data patterns.
2. **Descriptive phase:** Here, algorithms group and attribute identified patterns to new data.

These phases can be further subdivided into the following modules:

1. Data collection: Data collection is the first step of pattern recognition. The accuracy of recognition is largely dependent on the quality of the datasets. As such, using open-source datasets is preferable and can save time instead of manual data collection processes. Thus, receiving data from the real world kick-starts the recognition process.

2. Pre-processing: Once the data is received as input, the algorithms start the pre-processing step, where data is cleaned, and impurities are fixed to produce comprehensive datasets that yield good predictions. Pre-processing involves data segmentation. For instance, when you look at a group photograph posted by a friend on social media, you realize that you’re familiar with some of the faces in the picture, which attracts your attention. This is what pre-processing means.

Pre-processing is coupled with enhancement. For example, consider you are viewing the same photograph, but it is ten years older. Now, just to be sure that the familiar faces are real, you start comparing their eyes, skin tone, and other physical traits. This is where enhancement happens. It involves a smoothing and normalization process that tries to correct the image from strong variations. As a result, data becomes easy to interpret for the models.

3. Feature extraction: Next, features are extracted from the pre-processed input data. Here, the input data is converted into a feature vector, representing a reduced version of a set of features. This step solves the problem of the high dimensionality of the input dataset. This means that only relevant features are extracted rather than using the entire dataset.

Once the features are extracted, you should select features with the highest potential of delivering accurate results. Upon shortlisting such features, they are sent for further classification.

4. Classification: Extracted features are then compared to a similar pattern stored in the database. Here, learning can happen in a supervised and unsupervised manner. The supervised method has prior knowledge of each pattern category, while unsupervised method learning happens on the fly. As patterns are eventually matched to the stored data, the classification of input data happens.

5. Post-processing: Classification is followed by a post-processing step, which makes decisions on the best ways to utilize the results to guide the system efficiently. Moreover, it involves analyzing each segment of the identified or classified data to derive further insights. These extracted insights are then implemented in practice for future pattern recognition tasks.

Applications of Pattern Recognition

Pattern recognition uses several tools, such as statistical data analysis, probability, computational geometry, machine learning, and signal processing, to draw inferences from data. As the recognition model is used extensively across industries, its applications vary from computer vision, object detection, and speech and text recognition to radar processing.

1. Image recognition: Today, image recognition tools are employed by security and surveillance systems across sectors. These devices capture and monitor multiple video streams at a time. This helps detect potential intruders. The same image recognition tech is used at business centers, IT firms, and production facilities as face ID systems.

Another corollary of the same application is presented by the 'emotion detection system.' Here, pattern recognition is applied to images and video footage to analyze and detect the human emotions of an audience in real-time. The objective of such systems is to identify the mood, sentiment, and intent of users. Thus, deep learning models are used to detect the patterns of facial expressions and body language of people. This data can then be used by organizations to fine-tune their marketing campaigns and thereby improve customer experience.

Another use case of image recognition is that of ‘object detection.’ This is a key tool for visual search applications. In this case, objects within an image or video segment are identified and labeled. It forms the basis of visual search wherein users can search and compare labeled images.

2. Text pattern recognition and NLP: Recognition algorithms are typically used to identify patterns in text data, which is then used in applications such as text translation, grammar correction, plagiarism detection, etc. Some machine learning-based pattern recognition algorithms are used to classify documents and detect sensitive text passages automatically. This applies to the finance and insurance sectors, where text pattern recognition is used for fraud detection.

3. Fingerprint scanning: Today, almost all smartphones and laptops have a fingerprint identification feature to protect the device from unauthorized access. This is because these smart devices have used pattern analysis to learn the features of your fingerprint and decide whether to allow or deny the user access request.

4. Seismic activity analysis: When observing how earthquakes and other natural calamities disturb the Earth’s crust, pattern recognition is an effective tool to study such earthly parameters. For instance, researchers can study seismic records and identify recurring patterns to develop disaster-resilient models that can mitigate seismic effects on time.

5. Audio and voice recognition: Personal assistants and speech-to-text converters are voice and audio recognition systems that run on pattern recognition principles. For instance, Apple’s Siri and Samsung’s Alexa are tools that perceive and analyze audio and voice signals to understand the meaning of words and phrases and accomplish the associated tasks.

6. Healthcare: The relevance of pattern recognition in the medical field was highlighted by a recent paper published by Nature Communications in February 2021. It was assumed that COVID-19 affected the older age group more than younger people, and researchers at MIT opined that it was not only due to the aging of the immune system but also due to lung changes that come along with growing age.

The scientific community at MIT studied lung images of the elderly and used pattern recognition to identify a change in the lung patterns of older groups. The study established that aging caused stiffening of the lung tissues and showed different gene expressions than the ones seen in younger individuals.

Such pattern recognition techniques are also used to detect and forecast cancer. For instance, clinical decision support systems (CDSS) use pattern recognition methods to diagnose patients based on their symptoms, while computer-aided detection systems

(CAD) assist doctors in interpreting medical images. CAD applications include breast cancer, lung cancer, and so on.

7. Social media: Pattern recognition can be employed on social media platforms as a security tool. It can be used to find offensive posts, detect suspected religious activists, identify criminals, or zero in on tweets that cause civil unrest. It can also be used to identify posts or comments that indicate self-harm and suicidal thoughts.

While social media already generates enormous amounts of data every day, AI can turn this data into actionable information. For example, Facebook is known to employ pattern recognition to detect fake accounts by using an individual's profile pics.

8. Cybersecurity: Organizational networks can use pattern recognition-based security systems that detect activity trends and respond to changing user behavior to block potential hackers. If cybersecurity teams have instant access to malware patterns, they can take appropriate action before an attack or threat hits the network. For instance, intrusion detection systems are AI filters that sit inside a corporate network and look for potential threats on the network.

9. Robotics: In modern times, robotic task forces have become common across industries. Robots are being increasingly employed to perform dangerous tasks. For instance, the detection of radioactive material is nowadays performed by robots. These machines use pattern recognition to complete the task. In this case, the robot's camera captures images of a mine, extracts the discriminative features, and uses classification algorithms to segregate images into dangerous or non-dangerous based on the detected features.

10. Optical character recognition: Optical character recognition OCR converts scanned images of text, photos, and screenshots into editable documents. The character recognition process eliminates the need to write documents manually, saving time and increasing efficiency. For example, PDF document editors and digital libraries refer to such programs with built-in character recognition features.

11. Coding: Coding is another field where pattern recognition is widely used. Pattern recognition assists developers in identifying errors in codes. Some of the popular examples include:

- **GitHub Copilot:** This AI-driven tool works alongside an editor and suggests some lines of code or entire functions. It is a developer-friendly tool designed by GitHub and OpenAI and supports multiple programming languages such as JavaScript, TypeScript, Python, Go, and Ruby. It is an advanced coding tool that not only recommends code but also suggests algorithms or entire unit tests. The tool is trained on billions of lines of code from public sources.
- **Tabnine:** This is another AI assistant for programmers. Its models offer suggestions for each keystroke made by a user. Moreover, recommendations are in the form of entire lines or functions in IDEs. It supports different IDEs such as VSCode, Android Studio, IntelliJ, Eclipse, and Webstorm.

- **Clever-Commit:** This is a coding assistant designed by Mozilla in partnership with Ubisoft. The AI tool is based on a 2018 research study that proposes a CLEVER (Combining Levels of Bug Prevention and Resolution Techniques) approach. The assistant not only identifies incorrect code but also suggests potential changes to it.