compiled by: Dr. Mohsen Kahani
# Contents

## Articles

<table>
<thead>
<tr>
<th>Article</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expert system</td>
<td>1</td>
</tr>
<tr>
<td>Knowledge representation and reasoning</td>
<td>8</td>
</tr>
<tr>
<td>Reasoning system</td>
<td>16</td>
</tr>
<tr>
<td>Forward chaining</td>
<td>19</td>
</tr>
<tr>
<td>Rete algorithm</td>
<td>20</td>
</tr>
<tr>
<td>Backward chaining</td>
<td>27</td>
</tr>
<tr>
<td>Backward induction</td>
<td>29</td>
</tr>
<tr>
<td>Production system</td>
<td>31</td>
</tr>
<tr>
<td>Production Rule Representation</td>
<td>34</td>
</tr>
<tr>
<td>Inference engine</td>
<td>35</td>
</tr>
<tr>
<td>Fuzzy logic</td>
<td>37</td>
</tr>
<tr>
<td>Fuzzy control system</td>
<td>43</td>
</tr>
<tr>
<td>Artificial neural network</td>
<td>55</td>
</tr>
<tr>
<td>Types of artificial neural networks</td>
<td>66</td>
</tr>
<tr>
<td>Feedforward neural network</td>
<td>72</td>
</tr>
<tr>
<td>Self-organizing map</td>
<td>76</td>
</tr>
<tr>
<td>Hybrid intelligent system</td>
<td>81</td>
</tr>
<tr>
<td>Neuro-fuzzy</td>
<td>82</td>
</tr>
<tr>
<td>Genetic fuzzy systems</td>
<td>84</td>
</tr>
<tr>
<td>Knowledge engineering</td>
<td>85</td>
</tr>
<tr>
<td>Knowledge retrieval</td>
<td>88</td>
</tr>
<tr>
<td>Knowledge acquisition</td>
<td>90</td>
</tr>
<tr>
<td>Knowledge management</td>
<td>91</td>
</tr>
<tr>
<td>Data warehouse</td>
<td>99</td>
</tr>
<tr>
<td>Extract, transform, load</td>
<td>106</td>
</tr>
<tr>
<td>Star schema</td>
<td>113</td>
</tr>
<tr>
<td>Snowflake schema</td>
<td>115</td>
</tr>
<tr>
<td>Data mining</td>
<td>118</td>
</tr>
<tr>
<td>Cross Industry Standard Process for Data Mining</td>
<td>131</td>
</tr>
<tr>
<td>Statistical classification</td>
<td>133</td>
</tr>
<tr>
<td>Cluster analysis</td>
<td>137</td>
</tr>
<tr>
<td>Association rule learning</td>
<td>148</td>
</tr>
<tr>
<td>Sequence mining</td>
<td>155</td>
</tr>
<tr>
<td>Anomaly detection</td>
<td>156</td>
</tr>
</tbody>
</table>
References

Article Sources and Contributors 159
Image Sources, Licenses and Contributors 163

Article Licenses

License 164
In artificial intelligence, an **expert system** is a computer system that emulates the decision-making ability of a human expert.[1] Expert systems are designed to solve complex problems by reasoning about knowledge, like an expert, and not by following the procedure of a developer as is the case in conventional programming.[2][3][4] The first expert systems were created in the 1970s and then proliferated in the 1980s.[5] Expert systems were among the first truly successful forms of AI software.[6][7][8][9][10][11]

An expert system has a unique structure, different from traditional programs. It is divided into two parts, one fixed, independent of the expert system: the inference engine, and one variable: the knowledge base. To run an expert system, the engine reasons about the knowledge base like a human.[12] In the 80s a third part appeared: a dialog interface to communicate with users.[13] This ability to conduct a conversation with users was later called "conversational".[14][15]

**History**

Expert systems were introduced by researchers in the Stanford Heuristic Programming Project, including the "father of expert systems" with the Dendral and Mycin systems. Principal contributors to the technology were Bruce Buchanan, Edward Shortliffe, Randall Davis, William vanMelle, Carli Scott and others at Stanford. Expert systems were among the first truly successful forms of AI software.[6][7][8][9][10][11]

Research is also very active in France, where researchers focus on the automation of reasoning and logic engines. The French Prolog computer language, designed in 1972, marks a real advance over expert systems like Dendral or Mycin: it is a shell,[16] that is to say a software structure ready to receive any expert system and to run it. It integrates an engine using First-Order logic, with rules and facts. It's a tool for mass production of expert systems and was the first operational declarative language,[17] later becoming the best selling AI language in the world.[18] However Prolog is not particularly user friendly and is an order of logic away from human logic.[19][20][21]

In the 1980s, expert systems proliferated as they were recognized as a practical tool for solving real-world problems. Universities offered expert system courses and two thirds of the Fortune 1000 companies applied the technology in daily business activities.[5][22] Interest was international with the Fifth Generation Computer Systems project in Japan and increased research funding in Europe. Growth in the field continued into the 1990s.

The development of expert systems was aided by the development of the symbolic processing languages Lisp and Prolog. To avoid re-inventing the wheel, expert system shells were created that had more specialized features for building large expert systems.[23]

In 1981 the first IBM PC was introduced, with MS-DOS operating system. Its low price started to multiply users and opened a new market for computing and expert systems. In the 80's the image of AI was very good and people believed it would succeed within a short time[15]. Many companies began to market expert systems shells from universities, renamed "generators" because they added to the shell a tool for writing rules in plain language and thus, theoretically, allowed to write expert systems without a programming language nor any other software[16]. The best known: Guru (USA) inspired by Mycin[17][18], Personal Consultant Plus (USA)[19][20], Nexpert Object (developed by Neuron Data, company founded in California by three French)[21][22], Genesia (developed by French public company Electricité de France and marketed by Steria)[23], VP Expert (USA)[24]. But eventually the tools were only used in research projects. They did not penetrate the business market, showing that AI technology was not mature.

In 1986, a new expert system generator for PCs appeared on the market, derived from the French academic research: Intelligence Service,[24][25] sold by GSI-TECSI software company. This software showed a radical innovation: it used propositional logic ("Zeroth order logic") to execute expert systems, reasoning on a knowledge base written with everyday language rules, producing explanations and detecting logic contradictions between the facts. It was the
first tool showing the AI defined by Edward Feigenbaum in his book about the Japanese Fifth Generation, Artificial Intelligence and Japan's Computer Challenge to the World [26] (1983): "The machines will have reasoning power; they will automatically engineer vast amounts of knowledge to serve whatever purpose humans propose, from medical diagnosis to product design, from management decisions to education", "The reasoning animal has, perhaps inevitably, fashioned the reasoning machine", "the reasoning power of these machines matches or exceeds the reasoning power of the humans who instructed them and, in some cases, the reasoning power of any human performing such tasks". Intelligence Service was in fact "Pandora" (1985), [27] a software developed for their thesis by two academic students of Jean-Louis Laurière, [28] one of the most famous and prolific French AI researcher. [29] Unfortunately, as this software was not developed by his own IT developers, GSI-TECSI was unable to make it evolve. Sales became scarce and marketing stopped after a few years.

Software architecture

The rule base or knowledge base

In expert system technology, the knowledge base is expressed with natural language rules IF ... THEN ... For examples:

- "IF it is living THEN it is mortal"
- "IF his age = known THEN his year of birth = date of today - his age in years"
- "IF the identity of the germ is not known with certainty AND the germ is gram-positive AND the morphology of the organism is "rod" AND the germ is aerobic THEN there is a strong probability (0.8) that the germ is of type enterobacteriaceae" [30]

This formulation has the advantage of speaking in everyday language which is very rare in computer science (a classic program is coded). Rules express the knowledge to be exploited by the expert system. There exist other formulations of rules, which are not in everyday language, understandable only to computer scientists. Each rule style is adapted to an engine style. The whole problem of expert systems is to collect this knowledge, usually unconscious, from the experts. There are methods but almost all are usable only by computer scientists.

The inference engine

The inference engine is a computer program designed to produce a reasoning on rules. In order to produce a reasoning, it should be based on logic. There are several kinds of logic: propositional logic, predicates of order 1 or more, epistemic logic, modal logic, temporal logic, fuzzy logic, etc. Except for propositional logic, all are complex and can only be understood by mathematicians, logicians or computer scientists. Propositional logic is the basic human logic, that is expressed in syllogisms. The expert system that uses that logic is also called a zeroth-order expert system. With logic, the engine is able to generate new information from the knowledge contained in the rule base and data to be processed.

The engine has two ways to run: batch or conversational. In batch, the expert system has all the necessary data to process from the beginning. For the user, the program works as a classical program: he provides data and receives results immediately. Reasoning is invisible. The conversational method becomes necessary when the developer knows he cannot ask the user for all the necessary data at the start, the problem being too complex. The software must "invent" the way to solve the problem, request the missing data from the user, gradually approaching the goal as quickly as possible. The result gives the impression of a dialogue led by an expert. To guide a dialogue, the engine may have several levels of sophistication: "forward chaining", "backward chaining" and "mixed chaining". Forward chaining is the questioning of an expert who has no idea of the solution and investigates progressively (e.g. fault diagnosis). In backward chaining, the engine has an idea of the target (e.g. is it okay or not? Or: there is danger but what is the level?). It starts from the goal in hopes of finding the solution as soon as possible. In mixed chaining the engine has an idea of the goal but it is not enough: it deduces in forward chaining from previous user responses all...
that is possible before asking the next question. So quite often he deduces the answer to the next question before asking it.

A strong interest in using logic is that this kind of software is able to give the user clear explanation of what it is doing (the "Why?") and what it has deduced (the "How?"). Better yet, thanks to logic, the most sophisticated expert systems are able to detect contradictions[31] in user information or in the knowledge and can explain them clearly, revealing at the same time the expert's knowledge and way of thinking.

**Advantages**

**Conversational**

Expert systems offer many advantages for users when compared to traditional programs because they operate like a human brain [32] [33].

**Quick availability and opportunity to program itself**

As the rule base is in everyday language (the engine is untouchable), expert system can be written much faster than a conventional program, by users or experts, bypassing professional developers and avoiding the need to explain the subject.

**Ability to exploit a considerable amount of knowledge**

The expert system uses a rule base, unlike conventional programs, which means that the volume of knowledge to program is not a major concern. Whether the rule base has 10 rules or 10 000, the engine operation is the same.

**Reliability**

The reliability of an expert system is the same as the reliability of a database, i.e. good, higher than that of a classical program. It also depends on the size of knowledge base.

**Scalability**

Evolving an expert system is to add, modify or delete rules. Since the rules are written in plain language, it is easy to identify those to be removed or modified.

**Pedagogy**

The engines that are run by a true logic are able to explain to the user in plain language why they ask a question and how they arrived at each deduction. In doing so, they show knowledge of the expert contained in the expert system. So, user can learn this knowledge in its context. Moreover, they can communicate their deductions step by step. So, the user has information about their problem even before the final answer of the expert system.

**Preservation and improvement of knowledge**

Valuable knowledge can disappear with the death, resignation or retirement of an expert. Recorded in an expert system, it becomes eternal. To develop an expert system is to interview an expert and make the system aware of their knowledge. In doing so, it reflects and enhances it.
Expert system

New areas neglected by conventional computing

Automating a vast knowledge, the developer may meet a classic problem: "combinatorial explosion" commonly known as "information overload" that greatly complicates his work and results in a complex and time consuming program. The reasoning expert system does not encounter that problem since the engine automatically loads combinatorics between rules. This ability can address areas where combinatorics are enormous: highly interactive or conversational applications, fault diagnosis, decision support in complex systems, educational software, logic simulation of machines or systems, constantly changing software.

Disadvantages

The expert system has a major flaw, which explains its low success despite the principle having existed for 70 years: knowledge collection and its interpretation into rules, or knowledge engineering. Most developers have no automated method to perform this task; instead they work manually, increasing the likelihood of errors. Expert knowledge is generally not well understood; for example, rules may not exist, be contradictory, or be poorly written and unusable. Worse still, most expert systems use an engine incapable of reasoning. As a result, an expert system will often work poorly, and the project abandoned. Correct development methodology can mitigate these problems. There exists software capable of interviewing a true expert on a subject and automatically writing the rule base, or knowledge base, from the answers. The expert system can then be simultaneously run before the true expert's eyes, performing a consistency of rules check. Experts and users can check the quality of the software before it is finished.

Many expert systems are also penalized by the logic used. Most formal systems of logic operate on variable facts, i.e. facts the value of which changes several times during one reasoning. This is considered a property belonging to more powerful logic. This is the case of the Mycin and Dendral expert systems, and of, for example, fuzzy logic, predicate logic (Prolog), symbolic logic and mathematical logic. Propositional logic uses only invariant facts in the human mind, the facts used must remain invariable as long as the brain reasons with them. This makes possible two ways of controlling the consistency of the knowledge: detection of contradictions and production of explanations. That is why expert systems using variable facts, which are more understandable to developers creating such systems and hence more common, are less easy to develop, less clear to users, less reliable, and why they don't produce explanations of their reasoning, or contradiction detection.

Application field

Expert systems address areas where combinatorics is enormous:

- highly interactive or conversational applications, IVR, voice server, chatterbot
- fault diagnosis, medical diagnosis
- decision support in complex systems, process control, interactive user guide
- educational and tutorial software
- logic simulation of machines or systems
- knowledge management
- constantly changing software.

They can also be used in software engineering for rapid prototyping applications (RAD). Indeed, the expert system quickly developed in front of the expert shows him if the future application should be programmed. Indeed, any program contains expert knowledge and classic programming always begins with an expert interview. A program written in the form of expert system receives all the specific benefits of expert system, among others things it can be developed by anyone without computer training and without programming languages. But this solution has a defect: expert system runs slower than a traditional program because he consistently "thinks" when in fact a classic software just follows paths traced by the programmer.
Examples of applications

Expert systems are designed to facilitate tasks in the fields of accounting, the law, 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 standardized, 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 are in trading recommendations in various marketplaces. These markets involve numerous variables and human 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 "the game's AI provided the opposing manager's strategy".

A new application for expert systems is automated computer program generation. Funded by a US Air Force grant, an expert system-based application (hprcARCHITECT) that generates computer programs for mixed processor technology (FPGA/GPU/Multicore) systems without a need for technical specialists has recently been commercially introduced.

There is also a large body of contemporary research and development directed toward using expert systems for human behavior modeling and decision support systems. The former is especially important in the area of intercultural relations and the latter in improving management operations in small businesses.

Knowledge engineering

The building, maintaining and development of expert systems is 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 expertise".

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.
References

[2] Nwigbo Stella and Agbo Okechukwu Chucks (http://www.hrmars.admin/pics/261.pdf), School of Science Education, Expert system: a catalyst in educational development in Nigeria: "The ability of this system to explain the reasoning process through back-traces (...) provides an additional feature that conventional programming does not handle"
[12] Nwigbo Stella and Agbo Okechukwu Chucks (http://www.hrmars.admin/pics/261.pdf), School of Science Education, Expert system: a catalyst in educational development in Nigeria: "Knowledge-based systems collect the small fragments of human know-how into a knowledge-base which is used to reason through a problem, using the knowledge that is appropriated"
[20] "Software developed in Prolog has been criticized for having a high performance penalty compared to conventional programming languages"
[21] Dr. Nikolai Bezroukov (http://softpanorama.org/Lang/prolog.shtml), Softpanorama: "I think that most people exposed to Prolog remember strongly the initial disappointment. Language was/is so hyped but all you can see initially are pretty trivial examples that are solved by complex, obscure notation that lacks real expressive power: some of simple examples can be expressed no less concisely is many other languages"
[24] Flamant B. and Girard G (http://cat.inist.fr/?aModele=afficheN&cpsidt=7001328)., GSI-TECSI, Intelligence Service: build your own expert system : "Intelligence Service is a development environment for expert systems that requires no experience of classic programming that offers to everyone the opportunity to develop its own expert system"
[30] Mycin rule
Expert system


[34] Kenneth Laudon, Jane Laudon, Eric Fimbel, "Management Information Systems: Managing the Digital Firm", Business & Economics, 2010 edition, chapter 11-3.5: The implementation of a large number of expert systems requires the deployment of considerable development efforts, lengthy and expensive. Hiring and training a larger number of experts may be less expensive than building an expert system. (...) Some expert systems, particularly the largest, are so complex that over years, the costs of curative and adaptive maintenance become as high as the cost of development.

[35] Systèmes Experts, April 15, 1990, Miao, authentic expert system generator of fault diagnosis: "MIAO can explain, again in [plain] language, all of his logical approach: why he is asking such a question and how it came to such a conclusion. And that because he is constantly reasoning and not because an IT developer programmed in advance all the possible explanations."

[36] Olivier Rafal (http://www.tree-logic.com/Articles/LMI 2001 cf Maieutique.jpg), Le Monde Informatique, Programming for all (T.Rex generator): "This software allows to develop a conversational application (...) leading to a self-learning" (i.e. thanks to the automatic explanations)

[37] French Technology Survey (http://www.tree-logic.com/Articles/MaIAEutica par FTS (91).jpg), MAIEUTICA, An Expert System Generator which writes its own rules, July 1991: "checking the coherence of the knowledge", "it can detect contradictions", "it react appropriately to changes of mind"

[38] RGU: School of Computing (http://www.comp.rgu.ac.uk/docs/info/index.php), More Complex Inference: "propositional logic, where variables are not allowed"

[39] Ong K. and Lee R.M, Texas University-Austin, A logic model for maintaining consistency of bureaucratic policies, 1993: "Inconsistencies can be detected if any of the integrity constraints is proven false, and an explanation can be provided based on the proof tree. A more general inference mechanism is presented based on the theory of abduction for checking potential inconsistency of policies"

[40] Carl G. Hempel and Paul Oppenheim (http://people.cohums.ohio-state.edu/tennant9/hempel_oppenheim_PS1948.pdf), Philosophy of Science, Studies in the Logic of Explanation, 1948: "The sentences constituting the explanans must be true"


Bibliography

Textbooks


History of AI


Knowledge representation and reasoning

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.[1]

Overview

Knowledge Representation (KR) research involves analysis of how to reason accurately and effectively 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. The logic is interpretation theory. 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. Autoepistemic 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.\(^1\)

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 Markup Language (DAML), Ontology Inference Layer (OIL),\(^2\) and Web Ontology 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 behavior, the fundamental goal of knowledge representation is to facilitate reasoning, inferencing, or drawing conclusions. 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:\(^3\)[4]

- 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 or should 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 tomato could 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:

- Coverage, which means the KR covers a breadth and depth of information. Without a wide coverage, the KR cannot determine anything or resolve ambiguities.
- 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). It should also have simple primitives that combine in complex forms.
- 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.
- Efficient
- Easiness for modifying and updating.
- 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 encyclopedia (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

Knowledge representation and reasoning is also referred to as KRR.

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 encyclopedia, encoding not the information itself, but the information a reader would need in order to understand the encyclopedia: 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).

Topics

Language and notation
Some think it is best to represent knowledge in the same way that it is represented in the human mind, or to represent knowledge in the form of human language.

Psycholinguistics investigates how the human mind stores and manipulates language. Other branches of cognitive science examine how human memory stores sounds, sights, smells, emotions, procedures, and abstract ideas. Science has not yet completely described the internal mechanisms of the brain to the point where they can simply be replicated by computer programmers.

Various artificial languages and notations have been proposed for representing knowledge. They are typically based on logic and mathematics, and have easily parsed grammars to ease machine processing. They usually fall into the broad domain of ontologies.

Ontology Engineering
After CycL, a number of ontology languages have been developed. Most are declarative languages, and are either frame languages, or are based on first-order logic. Most of these languages only define an upper ontology with generic concepts, whereas the domain concepts are not part of the language definition. These languages all use special-purpose knowledge engineering because as stated by Tom Gruber, "Every ontology is a treaty- a social agreement among people with common motive in sharing." There are always many competing and differing views that make any general purpose ontology impossible. A general purpose ontology would have to be applicable in any domain and different areas of knowledge need to be unified. Gellish English is an example of an ontological language that includes a full engineering English Dictionary.

There is a long history of work attempting to build good ontologies for a variety of task domains, including early work on an ontology for liquids, the lumped element model widely used in representing electronic circuits (e.g.,), as well as ontologies for time, belief, and even programming itself. Each of these offers a way to see some part of the world. The lumped element model, for instance, suggests that we think of circuits in terms of components with connections between them, with signals flowing instantaneously along the connections. This is a useful view, but not the only possible one. A different ontology arises if we need to attend to the electrodynamics in the device: Here signals propagate at finite speed and an object (like a resistor) that was previously viewed as a single component with an I/O behavior may now have to be thought of as an extended medium through which an electromagnetic wave flows.

Ontologies can of course be written down in a wide variety of languages and notations (e.g., logic, LISP, etc.); the essential information is not the form of that language but the content, i.e., the set of concepts offered as a way of thinking about the world. Simply put, the important part is notions like connections and components, not whether we choose to write them as predicates or LISP constructs.

The commitment we make by selecting one or another ontology can produce a sharply different view of the task at hand. Consider the difference that arises in selecting the lumped element view of a circuit rather than the electrodynamic view of the same device. As a second example, medical diagnosis viewed in terms of rules (e.g., MYCIN) looks substantially different from the same task viewed in terms of frames (e.g., INTERNIST). Where MYCIN sees the medical world as made up of empirical associations connecting symptom to disease, INTERNIST sees a set of prototypes, in particular prototypical diseases, to be matched against the case at hand.
Commitment begins with the earliest choices
The INTERNIST example also demonstrates that there is significant and unavoidable ontological commitment even at the level of the familiar representation technologies. Logic, rules, frames, etc., each embody a viewpoint on the kinds of things that are important in the world. Logic, for instance, involves a (fairly minimal) commitment to viewing the world in terms of individual entities and relations between them. Rule-based systems view the world in terms of attribute-object-value triples and the rules of plausible inference that connect them, while frames have us thinking in terms of prototypical objects. Each of these thus supplies its own view of what is important to attend to, and each suggests, conversely, that anything not easily seen in those terms may be ignored. This is of course not guaranteed to be correct, since anything ignored may later prove to be relevant. But the task is hopeless in principle—every representation ignores something about the world—hence the best we can do is start with a good guess. The existing representation technologies supply one set of guesses about what to attend to and what to ignore. Selecting any of them thus involves a degree of ontological commitment: the selection will have a significant impact on our perception of and approach to the task, and on our perception of the world being modeled.

Commitments accumulate in layers
The ontologic commitment of a representation thus begins at the level of the representation technologies and accumulates from there. Additional layers of commitment are made as we put the technology to work. The use of frame-like structures in INTERNIST offers an illustrative example. At the most fundamental level, the decision to view diagnosis in terms of frames suggests thinking in terms of prototypes, defaults, and a taxonomic hierarchy. But prototypes of what, and how shall the taxonomy be organized?
An early description of the system [9] shows how these questions were answered in the task at hand, supplying the second layer of commitment:

The knowledge base underlying the INTERNIST system is composed of two basic types of elements: disease entities and manifestations.... [It] also contains a...hierarchy of disease categories, organized primarily around the concept of organ systems, having at the top level such categories as "liver disease," "kidney disease," etc.
The prototypes are thus intended to capture prototypical diseases (e.g., a "classic case" of a disease), and they will be organized in a taxonomy indexed around organ systems. This is a sensible and intuitive set of choices but clearly not the only way to apply frames to the task; hence it is another layer of ontological commitment.
At the third (and in this case final) layer, this set of choices is instantiated: which diseases will be included and in which branches of the hierarchy will they appear? Ontologic questions that arise even at this level can be quite fundamental. Consider for example determining which of the following are to be considered diseases (i.e., abnormal states requiring cure): alcoholism, homosexuality, and chronic fatigue syndrome. The ontologic commitment here is sufficiently obvious and sufficiently important that it is often a subject of debate in the field itself, quite independent of building automated reasoners.

Similar sorts of decisions have to be made with all the representation technologies, because each of them supplies only a first order guess about how to see the world: they offer a way of seeing but don't indicate how to instantiate that view. As frames suggest prototypes and taxonomies but do not tell us which things to select as prototypes, rules suggest thinking in terms of plausible inferences, but don't tell us which plausible inferences to attend to. Similarly logic tells us to view the world in terms of individuals and relations, but does not specify which individuals and relations to use.
Commitment to a particular view of the world thus starts with the choice of a representation technology, and accumulates as subsequent choices are made about how to see the world in those terms.
Knowledge representation and reasoning

**KR is not a data structure**
At each layer, even the first (e.g., selecting rules or frames), the choices being made are about representation, not data structures. Part of what makes a language representational is that it carries meaning,[10] i.e., there is a correspondence between its constructs and things in the external world. That correspondence in turn carries with it constraint. A semantic net, for example, is a representation, while a graph is a data structure. They are different kinds of entities, even though one is invariably used to implement the other, precisely because the net has (should have) a semantics. That semantics will be manifest in part because it constrains the network topology: a network purporting to describe family memberships as we know them cannot have a cycle in its parent links, while graphs (i.e., data structures) are of course under no such constraint and may have arbitrary cycles.

While every representation must be implemented in the machine by some data structure, the representational property is in the correspondence to something in the world and in the constraint that

**Links and structures**
While hyperlinks have come into widespread use, the closely related semantic link is not yet widely used. The mathematical table has been used since Babylonian times. More recently, these tables have been used to represent the outcomes of logic operations, such as truth tables, which were used to study and model Boolean logic, for example. Spreadsheets are yet another tabular representation of knowledge. Other knowledge representations are trees, graphs and hypergraphs, by means of which the connections among fundamental concepts and derivative concepts can be shown.

Visual representations are relatively new in the field of knowledge management but give the user a way to visualise how one thought or idea is connected to other ideas enabling the possibility of moving from one thought to another in order to locate required information.

**Notation**
The recent fashion in knowledge representation languages is to use XML as the low-level syntax. This tends to make the output of these KR languages easy for machines to parse, at the expense of human readability and often space-efficiency.

First-order predicate calculus is commonly used as a mathematical basis for these systems, to avoid excessive complexity. However, even simple systems based on this simple logic can be used to represent data that is well beyond the processing capability of current computer systems: see computability for reasons.

Examples of notations:
- DATR is an example for representing lexical knowledge
- RDF is a simple notation for representing relationships between and among objects

**Storage and manipulation**
One problem in knowledge representation is how to store and manipulate knowledge in an information system in a formal way so that it may be used by mechanisms to accomplish a given task. Examples of applications are expert systems, machine translation systems, computer-aided maintenance systems and information retrieval systems (including database front-ends).

Semantic networks may be used to represent knowledge. Each node represents a concept and arcs are used to define relations between the concepts. The Conceptual graph model is probably the oldest model still alive. One of the most expressive and comprehensively described knowledge representation paradigms along the lines of semantic networks is MultiNet (an acronym for Multilayered Extended Semantic Networks).

From the 1960s, the knowledge frame or just frame has been used. Each frame has its own name and a set of attributes, or slots which contain values; for instance, the frame for house might contain a color slot, number of...
Using frames for expert systems is an application of object-oriented programming, with inheritance of features described by the "is-a" link. However, there has been no small amount of inconsistency in the usage of the "is-a" link: Ronald J. Brachman wrote a paper titled "What IS-A is and isn't", wherein 29 different semantics were found in projects whose knowledge representation schemes involved an "is-a" link. Other links include the "part-of" link.

Frame structures are well-suited for the representation of schematic knowledge and stereotypical cognitive patterns. The elements of such schematic patterns are weighted unequally, attributing higher weights to the more typical elements of a schema [11]. A pattern is activated by certain expectations: If a person sees a big bird, he or she will classify it rather as a sea eagle than a golden eagle, assuming that his or her "sea-scheme" is currently activated and his "land-scheme" is not.

Frame representations are object-centered in the same sense as semantic networks are: All the facts and properties connected with a concept are located in one place - there is no need for costly search processes in the database.

A behavioral script is a type of frame that describes what happens temporally; the usual example given is that of describing going to a restaurant. The steps include waiting to be seated, receiving a menu, ordering, etc. The different solutions can be arranged in a so-called semantic spectrum with respect to their semantic expressivity.

References


Further reading

• Ronald J. Brachman; What IS-A is and isn't. An Analysis of Taxonomic Links in Semantic Networks (http://citeseer.nj. unc. com/context/177306/0); IEEE Computer, 16 (10); October 1983


• Jean-Luc Hainaut, Jean-Marc Hick, Vincent Englebert, Jean Henrard, Didier Roland: Understanding
  Implementations of IS-A Relations (http://www.informatik.uni-trier.de/~ley/db/conf/er/HainautHEHR96.html).
  ER 1996: 42-57
• Hermann Helbig: Knowledge Representation and the Semantics of Natural Language, Springer, Berlin,
  Heidelberg, New York 2006
• Arthur B. Markman: Knowledge Representation Lawrence Erlbaum Associates, 1998
• John F. Sowa: Knowledge Representation: Logical, Philosophical, and Computational Foundations. Brooks/Cole:
  New York, 2000
• Adrian Walker, Michael McCord, John F. Sowa, and Walter G. Wilson: Knowledge Systems and Prolog, Second

External links
• What is a Knowledge Representation? (http://medg.lcs.mit.edu/ftp/psz/k-rep.html) by Randall Davis and
  others
• Introduction to Knowledge Modeling (http://www.makhfi.com/KCM_intro.htm) by Pejman Makhfi
• Introduction to Description Logics course (http://www.inf.unibz.it/~franconi/dl/course/) by Enrico Franconi,
  Faculty of Computer Science, Free University of Bolzano, Italy
• DATR Lexical knowledge representation language (http://www.ccl.kuleuven.ac.be/LKR/html/datr.html)
• Loom Project Home Page (http://www.isi.edu/isd/LOOM/LOOM-HOME.html)
• Description Logic in Practice: A CLASSIC Application (http://www.research.att.com/sw/tools/classic/tm/
  ijcai-95-with-scenario.html)
• The Rule Markup Initiative (http://www.dfki.uni-kl.de/ruleml/)
• Schemas (http://moodle.ed.uiuc.edu/wiked/index.php/Schemas)
• Nelements KOS (http://nelements.org) - a non-free 3d knowledge representation system
Reasoning system

In information technology a reasoning system is any software application, hardware device or combination of software and hardware whose computational function is to generate conclusions from available knowledge using logical techniques of deduction, induction or other forms of reasoning. Reasoning systems are a subset of a broader category of intelligent systems. They play an important role in the practical implementation of knowledge engineering and artificial intelligence.

A reasoning system manipulates previously acquired knowledge in order to generate new knowledge. Knowledge is typically represented symbolically as informational facts and propositional statements that capture assertions, assumptions, beliefs and other premises. Sub-symbolic (connectionist) knowledge representations may also be used (e.g., trained neural nets). Reasoning systems automate the process of inferring or otherwise deriving new knowledge via the application of logic. In a concrete implementation, reasoning systems may support procedural attachments and built-in actions to process or apply knowledge within some given domain or situation.

Reasoning systems have a wide field of application that includes scheduling, business rule processing, problem solving, complex event processing, intrusion detection, predictive analytics, robotics, computer vision and natural language processing.

Use of logic

Reasoning systems apply logic in order to generate knowledge. However, they demonstrate significant variation in terms of systems of logic and formality. Most reasoning systems implement variations of propositional and symbolic (predicate) logic. These variations may be mathematically precise representations of formal logic systems (e.g., FOL), or extended and hybrid versions of those systems (e.g., Courteous logic[1]). Reasoning systems may explicitly implement additional logic types (e.g., modal, deontic, temporal logics). However, many reasoning systems implement imprecise and semi-formal approximations to recognised logic systems. These systems typically support a variety of procedural and semi-declarative techniques in order to model different reasoning strategies. They emphasise pragmatism over formality and may depend on custom extensions and attachments in order to solve real-world problems.

Many reasoning systems employ deductive reasoning to draw inferences from available knowledge. These inference engines support forward reasoning or backward reasoning to infer conclusions via modus ponens. The recursive reasoning methods they employ are termed ‘forward chaining’ and ‘backward chaining’, respectively. Although reasoning systems widely support deductive inference, some systems employ inductive, defeasible and other types of reasoning. Heuristics may also be employed to determine acceptable solutions to intractable problems.

Reasoning systems may employ the closed world assumption (CWA) or open world assumption (OWA). The OWA is often associated with ontological knowledge representation and the Semantic Web. Different systems exhibit a variety of approaches to negation. As well as logical or bitwise complement, systems may support existential forms of strong and weak negation including negation-as-failure and ‘inflationary’ negation (negation of non-ground atoms). Different reasoning systems may support monotonic or non-monotonic reasoning, stratification and other logical techniques.
Reasoning under uncertainty

Many reasoning systems provide capabilities for reasoning under uncertainty. This is important when building situated reasoning agents which must deal with uncertain representations of the world. There are several common approaches to handling uncertainty. These include the use of certainty factors, probabilistic methods such as Bayesian inference or Dempster–Shafer theory, multi-valued (‘fuzzy’) logic and various connectionist approaches.

Types of reasoning system

This section provides a non-exhaustive and informal categorisation of common types of reasoning system. These categories are not discreet. They overlap to a significant degree and share a number of techniques, methods and algorithms.

Constraint solvers

Constraint solvers solve constraint satisfaction problems (CSPs). They support constraint programming. A constraint is a condition which must be met by any valid solution to a problem. Constraints are defined declaratively and applied to variables within given domains. Constraint solvers use search, backtracking and constraint propagation techniques to find solutions and determine optimal solutions. They may employ forms of linear and nonlinear programming. They are often used to perform optimization within highly combinatorial problem spaces. For example, they may be used to calculate optimal scheduling, design efficient integrated circuits or maximise productivity in a manufacturing process.

Theorem provers

Theorem provers use automated reasoning techniques to determine proofs of mathematical theorems. They may also be used to verify existing proofs. In addition to academic use, typical applications of theorem provers include verification of the correctness of integrated circuits, software programs, engineering designs, etc.

Logic programs

Logic programs (LPs) are software programs written using programming languages whose primitives and expressions provide direct representations of constructs drawn from mathematical logic. An example of a general-purpose logic programming language is Prolog. LPs represent the direct application of logic programming to solve problems. Logic programming is characterised by highly declarative approaches based on formal logic, and has wide application across many disciplines.

Expert systems

Expert systems apply reasoning based on knowledge drawn from human (‘expert’) experience and insight. They emulate the reasoning processes of human experts. They provide mechanisms in which expert knowledge in a given domain can be captured and exploited in an automated or semi-automated fashion. Expert systems are often used to support decision making by eliciting facts from human operators and applying rule-based reasoning to recommend appropriate actions or decisions. They may support the elaboration and refinement of knowledge bases over time based on on-going human interaction.
**Rule engines**

Rule engines represent conditional logic as discrete rules. Rule sets can be managed and applied separately to other functionality. They have wide applicability across many domains. Many rule engines implement reasoning capabilities. A common approach is to implement production systems to support forward or backward chaining. Each rule (‘production’) binds a conjunction of predicate clauses to a list of executable actions. At run-time, the rule engine matches productions against facts and executes (‘fires’) the associated action list for each match. If those actions remove or modify any facts, or assert new facts, the engine immediately re-computes the set of matches. Rule engines are widely used to model and apply business rules, to control decision-making in automated processes and to enforce business and technical policies.

**Cognitive systems**

Cognitive systems emulate or support human cognitive abilities and processes such as problem solving, language comprehension, learning, perception and memory. They model aspects of human psychology associated with cognition. Cognitive systems may employ a variety of techniques to implement behaviour and decision-making capabilities comparable to human abilities. Cognitive systems are widely researched and have practical applications in defence, medicine, aviation and other areas.

**Machine learning systems**

Machine learning systems evolve their behavior over time based on experience. This may involve reasoning over observed events or example data provided for training purposes. For example, machine learning systems may use inductive reasoning to generate hypotheses for observed facts. Learning systems search for generalised rules or functions that yield results in line with observations and then use these generalisations to control future behavior.

**Case-based reasoning systems**

Case-based reasoning (CBR) systems provide solutions to problems by analysing similarities to other problems for which known solutions already exist. They use analogical reasoning to infer solutions based on case histories. CBR systems are commonly used in customer/technical support and call centre scenarios and have applications in industrial manufacture, agriculture, medicine, law and many other areas.

**Procedural reasoning systems**

A procedural reasoning system (PRS) uses reasoning techniques to select plans from a procedural knowledge base. Each plan represents a course of action for achievement of a given goal. The PRS implements a belief-desire-intention model by reasoning over facts (‘beliefs’) to select appropriate plans (‘intentions’) for given goals (‘desires’). Typical applications of PRS include management, monitoring and fault detection systems.

**References**


Forward chaining

Forward chaining is one of the two main methods of reasoning when using inference rules (in artificial intelligence) and can be described logically as repeated application of modus ponens. Forward chaining is a popular implementation strategy for expert systems, business and production rule systems. The opposite of forward chaining is backward chaining.

Forward chaining starts with the available data and uses inference rules to extract more data (from an end user for example) until a goal is reached. An inference engine using forward chaining searches the inference rules until it finds one where the antecedent (If clause) is known to be true. When found it can conclude, or infer, the consequent (Then clause), resulting in the addition of new information to its data.

Inference engines will iterate through this process until a goal is reached.

For example, suppose that the goal is to conclude the color of a pet named Fritz, given that he croaks and eats flies, and that the rule base contains the following four rules:

1. If X croaks and eats flies - Then X is a frog
2. If X chirps and sings - Then X is a canary
3. If X is a frog - Then X is green
4. If X is a canary - Then X is yellow

This rule base would be searched and the first rule would be selected, because its antecedent (If Fritz croaks and eats flies) matches our data. Now the consequents (Then X is a frog) is added to the data. The rule base is again searched and this time the third rule is selected, because its antecedent (If Fritz is a frog) matches our data that was just confirmed. Now the new consequent (Then Fritz is green) is added to our data. Nothing more can be inferred from this information, but we have now accomplished our goal of determining the color of Fritz.

Because the data determines which rules are selected and used, this method is called data-driven, in contrast to goal-driven backward chaining inference. The forward chaining approach is often employed by expert systems, such as CLIPS.

One of the advantages of forward-chaining over backward-chaining is that the reception of new data can trigger new inferences, which makes the engine better suited to dynamic situations in which conditions are likely to change.
Rete algorithm

The Rete algorithm is an efficient pattern matching algorithm for implementing production rule systems. The Rete algorithm was designed by Dr Charles L. Forgy of Carnegie Mellon University, first published in a working paper in 1974, and later elaborated in his 1979 Ph.D. thesis and a 1982 paper (see References). Rete has become the basis for many popular rule engines and expert system shells, including Tibco Business Events, CLIPS, Jess, Drools, JRules, OPSJ, Blaze Advisor, BizTalk Rules Engine and Soar. The word 'Rete' is Latin for 'net' or 'comb'. The same word is used in modern Italian to mean network. Charles Forgy has reportedly stated that he adopted the term 'Rete' because of its use in anatomy to describe a network of blood vessels and nerve fibers.[1]

A naïve implementation of an expert system might check each rule against the known facts in the knowledge base, firing that rule if necessary, then moving on to the next rule (and looping back to the first rule when finished). For even moderate sized rules and facts knowledge-bases, this naïve approach performs far too slowly. The Rete algorithm provides the basis for a more efficient implementation. A Rete-based expert system builds a network of nodes, where each node (except the root) corresponds to a pattern occurring in the left-hand-side (the condition part) of a rule. The path from the root node to a leaf node defines a complete rule left-hand-side. Each node has a memory of facts which satisfy that pattern. This structure is essentially a generalized trie. As new facts are asserted or modified, they propagate along the network, causing nodes to be annotated when that fact matches that pattern. When a fact or combination of facts causes all of the patterns for a given rule to be satisfied, a leaf node is reached and the corresponding rule is triggered.

The Rete algorithm is designed to sacrifice memory for increased speed. In most cases, the speed increase over naïve implementations is several orders of magnitude (because Rete performance is theoretically independent of the number of rules in the system). In very large expert systems, however, the original Rete algorithm tends to run into memory consumption problems. Other algorithms, both novel and Rete-based, have since been designed which require less memory.

Description

The Rete algorithm provides a generalized logical description of an implementation of functionality responsible for matching data tuples ("facts") against productions ("rules") in a pattern-matching production system (a category of rule engine). A production consists of one or more conditions and a set of actions which may be undertaken for each complete set of facts that match the conditions. Conditions test fact attributes, including fact type specifiers/identifiers. The Rete algorithm exhibits the following major characteristics:

- It reduces or eliminates certain types of redundancy through the use of node sharing.
- It stores partial matches when performing joins between different fact types. This, in turn, allows production systems to avoid complete re-evaluation of all facts each time changes are made to the production system's working memory. Instead, the production system needs only to evaluate the changes (deltas) to working memory.
- It allows for efficient removal of memory elements when facts are retracted from working memory.

The Rete algorithm is widely used to implement matching functionality within pattern-matching engines that exploit a match-resolve-act cycle to support forward chaining and inferencing.

Retes are directed acyclic graphs that represent higher-level rule sets. They are generally represented at run-time using a network of in-memory objects. These networks match rule conditions (patterns) to facts (relational data tuples). Rete networks act as a type of relational query processor, performing projections, selections and joins conditionally on arbitrary numbers of data tuples.

Productions (rules) are typically captured and defined by analysts and developers using some high-level rules language. They are collected into rule sets which are then translated, often at run time, into an executable Rete.
When facts are "asserted" to working memory, the engine creates *working memory elements* (WMEs) for each fact. Facts are n-tuples, and may therefore contain an arbitrary number of data items. Each WME may hold an entire n-tuple, or, alternatively, each fact may be represented by a set of WMEs where each WME contains a fixed-length tuple. In this case, tuples are typically triplets (3-tuples).

Each WME enters the Rete network at a single root node. The root node passes each WME on to its child nodes, and each WME may then be propagated through the network, possibly being stored in intermediate memories, until it arrives at a terminal node.

**Alpha network**

The "left" *(alpha)* side of the node graph forms a discrimination network responsible for selecting individual WMEs based on simple conditional tests which match WME attributes against constant values. Nodes in the discrimination network may also perform tests that compare two or more attributes of the same WME. If a WME is successfully matched against the conditions represented by one node, it is passed to the next node. In most engines, the immediate child nodes of the root node are used to test the entity identifier or fact type of each WME. Hence, all the WMEs which represent the same entity type typically traverse a given branch of nodes in the discrimination network.

Within the discrimination network, each branch of alpha nodes (also called 1-input nodes) terminates at a memory, called an *alpha memory*. These memories store collections of WMEs that match each condition in each node in a given node branch. WMEs that fail to match at least one condition in a branch are not materialised within the corresponding alpha memory. Alpha node branches may fork in order to minimise condition redundancy.

A possible variation is to introduce additional memories for each intermediate node in the discrimination network. This increases the overhead of the Rete, but may have advantages in situations where rules are dynamically added to or removed from the Rete, making it easier to vary the topography of the discrimination network dynamically.

An alternative implementation is described by Doorenbos.[2] In this case, the discrimination network is replaced by a set of memories and an index. The index may be implemented using a hash table. Each memory holds WMEs that match a single conditional pattern, and the index is used to reference memories by their pattern. This approach is only practical when WMEs represent fixed-length tuples, and the length of each tuple is short (e.g., 3-tuples). In addition, the approach only applies to conditional patterns that perform equality tests against constant values. When a WME enters the Rete, the index is used to locate a set of memories whose conditional pattern matches the WME attributes, and the WME is then added directly to each of these memories. In itself, this implementation contains no 1-input nodes. However, in order to implement non-equality tests, the Rete may contain additional 1-input node networks through which WMEs are passed before being placed in a memory. Alternatively, non-equality tests may be performed in the beta network described below.

**Beta network**

The "right" *(beta)* side of the graph chiefly performs joins between different WMEs. It is optional, and is only included if required. It consists of 2-input nodes where each node has a "left" and a "right" input. Each beta node sends its output to a *beta memory*.

Beta nodes process tokens. A token is a unit of storage within a memory and also a unit of exchange between memories and nodes. In many implementations, tokens are introduced within alpha memories where they are used to hold single WMEs. These tokens are then passed to the beta network.

Each beta node performs its work and, as a result, may create new tokens to hold a list of WMEs representing a partial match. These extended tokens are then stored in beta memories, and passed to subsequent beta nodes. In this case, the beta nodes typically pass lists of WMEs through the beta network by copying existing WME lists from each received token into new tokens and then adding a further WMEs to the lists as a result of performing a join or some other action. The new tokens are then stored in the output memory.
A common variation is to build linked lists of tokens where each token holds a single WME. In this case, lists of WMEs for a partial match are represented by the linked list of tokens. This approach may be better because it eliminates the need to copy lists of WMEs from one token to another. Instead, a beta node needs only to create a new token to hold a WME it wishes to join to the partial match list, and then link the new token to a parent token stored in the input beta memory. The new token now forms the head of the token list, and is stored in the output beta memory.

In descriptions of Rete, it is common to refer to token passing within the beta network. In this article, however, we will describe data propagation in terms of WME lists, rather than tokens, in recognition of different implementation options and the underlying purpose and use of tokens. As any one WME list passes through the beta network, new WMEs may be added to it, and the list may be stored in beta memories. A WME list in a beta memory represents a partial match for the conditions in a given production.

WME lists that reach the end of a branch of beta nodes represent a complete match for a single production, and are passed to terminal nodes. These nodes are sometimes called p-nodes, where "p" stands for production. Each terminal node represents a single production, and each WME list that arrives at a terminal node represents a complete set of matching WMEs for the conditions in that production. For each WME list it receives, a production node will "activate" a new production instance on the "agenda". Agendas are typically implemented as prioritised queues.

Beta nodes typically perform joins between WME lists stored in beta memories and individual WMEs stored in alpha memories. Each beta node is associated with two input memories. An alpha memory holds WM and performs "right" activations on the beta node each time it stores a new WME. A beta memory holds WME lists and performs "left" activations on the beta node each time it stores a new WME list. When a join node is right-activated, it compares one or more attributes of the newly stored WME from its input alpha memory against given attributes of specific WMEs in each WME list contained in the input beta memory. When a join node is left-activated it traverses a single newly stored WME list in the beta memory, retrieving specific attribute values of given WMEs. It compares these values with attribute values of each WME in the alpha memory.

Beta nodes take their input from two alpha memories. In order to eliminate node redundancies, any one alpha or beta memory may be used to perform activations on multiple beta nodes. As well as join nodes, the beta network may contain additional node types, some of which are described below. If a Rete contains no beta network, alpha nodes feed tokens, each containing a single WME, directly to p-nodes. In this case, there may be no need to store WMEs in alpha memories.

**Conflict resolution**

During any one match-resolve-act cycle, the engine will find all possible matches for the facts currently asserted to working memory. Once all the current matches have been found, and corresponding production instances have been activated on the agenda, the engine determines an order in which the production instances may be "fired". This is termed conflict resolution, and the list of activated production instances is termed the conflict set. The order may be based on rule priority (salience), rule order, the time at which facts contained in each instance were asserted to the working memory, the complexity of each production, or some other criteria. Many engines allow rule developers to select between different conflict resolution strategies or to chain a selection of multiple strategies.
Conflict resolution is not defined as part of the Rete algorithm, but is used alongside the algorithm. Some specialised production systems do not perform conflict resolution.

**Production execution**

Having performed conflict resolution, the engine now "fires" the first production instance, executing a list of actions associated with the production. The actions act on the data represented by the production instance's WME list.

By default, the engine will continue to fire each production instance in order until all production instances have been fired. Each production instance will fire only once, at most, during any one match-resolve-act cycle. This characteristic is termed refraction. However, the sequence of production instance firings may be interrupted at any stage by performing changes to the working memory. Rule actions can contain instructions to assert or retract WMEs from the working memory of the engine. Each time any single production instance performs one or more such changes, the engine immediately enters a new match-resolve-act cycle. This includes "updates" to WMEs currently in the working memory. Updates are represented by retracting and then re-asserting the WME. The engine undertakes matching of the changed data which, in turn, may result in changes to the list of production instances on the agenda. Hence, after the actions for any one specific production instance have been executed, previously activated instances may have been de-activated and removed from the agenda, and new instances may have been activated.

As part of the new match-resolve-act cycle, the engine performs conflict resolution on the agenda and then executes the current first instance. The engine continues to fire production instances, and to enter new match-resolve-act cycles, until no further production instances exist on the agenda. At this point the rule engine is deemed to have completed its work, and halts.

Some engines support advanced refraction strategies in which certain production instances executed in a previous cycle are not re-executed in the new cycle, even though they may still exist on the agenda.

It is possible for the engine to enter into never-ending loops in which the agenda never reaches the empty state. For this reason, most engines support explicit "halt" verbs that can be invoked from production action lists. They may also provide automatic loop detection in which never-ending loops are automatically halted after a given number of iterations. Some engines support a model in which, instead of halting when the agenda is empty, the engine enters a wait state until new facts are asserted externally.

As for conflict resolution, the firing of activated production instances is not a feature of the Rete algorithm. However, it is a central feature of engines that use Rete networks. Some of the optimisations offered by Rete networks are only useful in scenarios where the engine performs multiple match-resolve-act cycles.

**Existential and universal quantifications**

Conditional tests are most commonly used to perform selections and joins on individual tuples. However, by implementing additional beta node types, it is possible for Rete networks to perform quantifications. Existential quantification involves testing for the existence of at least one set of matching WMEs in working memory. Universal quantification involves testing that an entire set of WMEs in working memory meets a given condition. A variation of universal quantification might test that a given number of WMEs, drawn from a set of WMEs, meets given criteria. This might be in terms of testing for either an exact number or a minimum number of matches.

Quantification is not universally implemented in Rete engines, and, where it is supported, several variations exist. A variant of existential quantification referred to as negation is widely, though not universally, supported, and is described in seminal documents. Existentially negated conditions and conjunctions involve the use of specialised beta nodes that test for non-existence of matching WMEs or sets of WMEs. These nodes propagate WME lists only when no match is found. The exact implementation of negation varies. In one approach, the node maintains a simple count on each WME list it receives from its left input. The count specifies the number of matches found with WMEs received from the right input. The node only propagates WME lists whose count is zero. In another approach, the
node maintains an additional memory on each WME list received from the left input. These memories are a form of beta memory, and store WME lists for each match with WMEs received on the right input. If a WME list does not have any WME lists in its memory, it is propagated down the network. In this approach, negation nodes generally activate further beta nodes directly, rather than storing their output in an additional beta memory. Negation nodes provide a form of 'negation as failure'.

When changes are made to working memory, a WME list that previously matched no WMEs may now match newly asserted WMEs. In this case, the propagated WME list and all its extended copies need to be retracted from beta memories further down the network. The second approach described above is often used to support efficient mechanisms for removal of WME lists. When WME lists are removed, any corresponding production instances are de-activated and removed from the agenda.

Existential quantification can be performed by combining two negation beta nodes. This represents the semantics of double negation (e.g., "If NOT NOT any matching WMEs, then..."). This is a common approach taken by several production systems.

**Memory indexing**

The Rete algorithm does not mandate any specific approach to indexing the working memory. However, most modern production systems provide indexing mechanisms. In some cases, only beta memories are indexed, whilst in others, indexing is used for both alpha and beta memories. A good indexing strategy is a major factor in deciding the overall performance of a production system, especially when executing rule sets that result in highly combinatorial pattern matching (i.e., intensive use of beta join nodes), or, for some engines, when executing rules sets that perform a significant number of WME retraction during multiple match-resolve-act cycles. Memories are often implemented using combinations of hash tables, and hash values are used to perform conditional joins on subsets of WME lists and WMEs, rather than on the entire contents of memories. This, in turn, often significantly reduces the number of evaluations performed by the Rete network.

**Removal of WMEs and WME lists**

When a WME is retracted from working memory, it must be removed from every alpha memory in which it is stored. In addition, WME lists that contain the WME must be removed from beta memories, and activated production instances for these WME lists must be de-activated and removed from the agenda. Several implementation variations exist, including tree-based and rematch-based removal. Memory indexing may be used in some cases to optimise removal.

**Handling ORed conditions**

When defining productions in a rule set, it is common to allow conditions to be grouped using an OR connective. In many production systems, this is handled by interpreting a single production containing multiple ORed patterns as the equivalent of multiple productions. The resulting Rete network contains sets of terminal nodes which, together, represent single productions. This approach disallows any form of short-circuiting of the ORed conditions. It can also, in some cases, lead to duplicate production instances being activated on the agenda where the same set of WMEs match multiple internal productions. Some engines provide agenda de-duplication in order to handle this issue.
Diagram

The following diagram illustrates the basic Rete topography, and shows the associations between different node types and memories.

- Most implementations use type nodes to perform the first level of selection on n-tuple working memory elements. Type nodes can be considered as specialized select nodes. They discriminate between different tuple relation types.
- The diagram does not illustrate the use of specialized nodes types such as negated conjunction nodes. Some engines implement several different node specialisations in order to extend functionality and maximise optimisation.
- The diagram provides a logical view of the Rete. Implementations may differ in physical detail. In particular, the diagram shows dummy inputs providing right activations at the head of beta node branches. Engines may implement other approaches, such as adapters that allow alpha memories to perform right activations directly.
- The diagram does not illustrate all node-sharing possibilities.

For a more detailed and complete description of the Rete algorithm, see chapter 2 of Production Matching for Large Learning Systems by Robert Doorenbos (see link below).

Miscellaneous considerations

Although not defined by the Rete algorithm, some engines provide extended functionality to support greater control of truth maintenance. For example, when a match is found for one production, this may result in the assertion of new WMEs which, in turn, match the conditions for another production. If a subsequent change to working memory causes the first match to become invalid, it may be that this implies that the second match is also invalid. The Rete algorithm does not define any mechanism to define and handle these logical truth dependencies automatically. Some engines, however, support additional functionality in which truth dependencies can be automatically maintained. In this case, the retraction of one WME may lead to the automatic retraction of additional WMEs in order to maintain logical truth assertions.

The Rete algorithm does not define any approach to justification. Justification refers to mechanisms commonly required in expert and decision systems in which, at its simplest, the system reports each of the inner decisions used to reach some final conclusion. For example, an expert system might justify a conclusion that an animal is an elephant by reporting that it is large, grey, has big ears, a trunk and tusks. Some engines provide built-in justification systems in conjunction with their implementation of the Rete algorithm.

This article does not provide an exhaustive description of every possible variation or extension of the Rete algorithm. Other considerations and innovations exist. For example, engines may provide specialised support within the Rete network in order to apply pattern-matching rule processing to specific data types and sources such as programmatic objects, XML data or relational data tables. Another example concerns additional time-stamping facilities provided by many engines for each WME entering a Rete network, and the use of these time-stamps in conjunction with conflict resolution strategies. Engines exhibit significant variation in the way they allow programmatic access to the
engine and its working memory, and may extend the basic Rete model to support forms of parallel and distributed processing.

**Optimisation and performance**

Several optimisations for Rete have been identified and described in academic literature. Several of these, however, apply only in very specific scenarios, and therefore often have little or no application in a general-purpose rules engine. In addition, alternative algorithms such as TREAT and LEAPS have been formulated which may provide additional performance improvements. There are currently very few commercial or open source examples of productions systems that support these alternative algorithms.

The Rete algorithm is orientated to scenarios where forward chaining and "inferencing" is used to calculate new facts from existing facts, or to filter and discard facts in order to arrive at some conclusion. It is also exploited as a reasonably efficient mechanism for performing highly combinatorial evaluations of facts where large numbers of joins must be performed between fact tuples. Other approaches to performing rule evaluation, such as the use of decision trees, or the implementation of sequential engines, may be more appropriate for simple scenarios, and should be considered as possible alternatives.

**Rete II**

In the 1980s, Dr Charles Forgy developed a successor to the Rete algorithm named Rete II.[3] Unlike the original Rete (which is public domain) this algorithm was not disclosed. Rete II claims better performance for more complex problems (even orders of magnitude[4]), and is officially implemented in CLIPS/R2.

Rete II can be characterized by two areas of improvement; specific optimizations relating to the general performance of the Rete network (including the use of hashed memories in order to increase performance with larger sets of data), and the inclusion of a backward chaining algorithm tailored to run on top of the Rete network. Backward chaining alone can account for the most extreme changes in benchmarks relating to Rete vs. Rete II.

Jess (at least versions 5.0 and later) also adds a backward chaining algorithm on top of the Rete network, but it cannot be said to fully implement Rete II, in part due to the fact that no full specification is publicly available.

**Rete-NT**

In 2010, Dr Charles Forgy developed a new generation of the Rete algorithm. In an InfoWorld benchmark, the algorithm was deemed 500 times faster than the original Rete algorithm and 10 times faster than its predecessor, Rete III.[5] This algorithm is now licensed to Sparkling Logic, the company that Charles joined as investor and strategic advisor,[6][7] as the inference engine of the SMARTS product.

**References**


• Charles Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", Artificial Intelligence, 19, pp 17–37, 1982

External links
• Rete Algorithm explained (http://drdobbs.com/184405218) Bruce Schneier, Dr. Dobb's Journal
• Production Matching for Large Learning Systems – R Doorenbos (http://reports-archive.adm.cs.cmu.edu/anon/1995/CMU-CS-95-113.pdf) Detailed and accessible description of Rete, also describes a variant named Rete/UL, optimised for large systems (PDF)
• According to the Rules (http://www.cut-the-knot.org/classes/Last.shtml) (A short introduction from cut-the-knot)

Backward chaining

Backward chaining (or backward reasoning) is an inference method that can be described (in lay terms) as working backward from the goal(s). It is used in automated theorem provers, proof assistants and other artificial intelligence applications, but it has also been observed in primates.

In game theory, its application to (simpler) subgames in order to find a solution to the game is called backward induction. In chess, it is called retrograde analysis, and it is used to generate tablebases for chess endgames for computer chess.

Backward chaining is implemented in logic programming by SLD resolution. Both rules are based on the modus ponens inference rule. It is one of the two most commonly used methods of reasoning with inference rules and logical implications – the other is forward chaining. Backward chaining systems usually employ a depth-first search strategy, e.g. Prolog.[1]

How it works

Backward chaining starts with a list of goals (or a hypothesis) and works backwards from the consequent to the antecedent to see if there is data available that will support any of these consequents.[2] An inference engine using backward chaining would search the inference rules until it finds one which has a consequent (Then clause) that matches a desired goal. If the antecedent (If clause) of that rule is not known to be true, then it is added to the list of goals (in order for one's goal to be confirmed one must also provide data that confirms this new rule).

For example, suppose that the goal is to conclude the color of my pet Fritz, given that he croaks and eats flies, and that the rule base contains the following four rules:

1. **If** X croaks and eats flies **Then** X is a frog
2. **If** X chirps and sings **Then** X is a canary
3. **If** X is a frog **Then** X is green
4. **If** X is a canary **Then** X is yellow

This rule base would be searched and the third and fourth rules would be selected, because their consequents (Then Fritz is green, Then Fritz is yellow) match the goal (to determine Fritz's color). It is not yet known that Fritz is a frog, so both the antecedents (If Fritz is a frog, If Fritz is a canary) are added to the goal list. The rule base is again
searched and this time the first two rules are selected, because their consequents (**Then X is a frog, Then X is a canary**) match the new goals that were just added to the list. The antecedent (**If Fritz croaks and eats flies**) is known to be true and therefore it can be concluded that Fritz is a frog, and not a canary. The goal of determining Fritz's color is now achieved (Fritz is green if he is a frog, and yellow if he is a canary, but he is a frog since he croaks and eats flies; therefore, Fritz is green).

Note that the goals always match the affirmed versions of the consequents of implications (and not the negated versions as in modus tollens) and even then, their antecedents are then considered as the new goals (and not the conclusions as in affirming the consequent) which ultimately must match known facts (usually defined as consequents whose antecedents are always true); thus, the inference rule which is used is modus ponens.

Because the list of goals determines which rules are selected and used, this method is called goal-driven, in contrast to data-driven forward-chaining inference. The backward chaining approach is often employed by expert systems. Programming languages such as Prolog, Knowledge Machine and ECLiPSe support backward chaining within their inference engines.[3]

References


[2] Definition of backward chaining as a depth-first search method:
- Russell & Norvig 2009, p. 337

[3] Languages that support backward chaining:

Other sources

General References

External links
- Backward chaining example (http://www.j-paine.org/students/lectures/lect3/node12.html)
**Backward induction**

**Backward induction** is the process of reasoning backwards in time, from the end of a problem or situation, to determine a sequence of optimal actions. It proceeds by first considering the last time a decision might be made and choosing what to do in any situation at that time. Using this information, one can then determine what to do at the second-to-last time of decision. This process continues backwards until one has determined the best action for every possible situation (i.e. for every possible information set) at every point in time.

In the mathematical optimization method of dynamic programming, backward induction is one of the main methods for solving the Bellman equation.[1][2] In game theory, backward induction is a method used to compute subgame perfect equilibria in sequential games.[3] The only difference is that optimization involves just one decision maker, who chooses what to do at each point of time, whereas game theory analyzes how the decisions of several players interact. That is, by anticipating what the last player will do in each situation, it is possible to determine what the second-to-last player will do, and so on. In the related fields of automated planning and scheduling and automated theorem proving, the method is called **backward search** or **backward chaining**. In chess it is called retrograde analysis.

Backward induction has been used to solve games as long as the field of game theory has existed. John von Neumann and Oskar Morgenstern suggested solving zero-sum, two-person games by backward induction in their *Theory of Games and Economic Behavior* (1944), the book which established game theory as a field of study.[4][5]

**An example of decision-making by backward induction**

Consider an unemployed person who will be able to work for ten more years $t = 1, 2, \ldots, 10$. Suppose that each year in which she remains unemployed, she may be offered a 'good' job that pays $100, or a 'bad' job that pays $44, with equal probability (50/50). Once she accepts a job, she will remain in that job for the rest of the ten years. (Assume for simplicity that she cares only about her monetary earnings, and that she values earnings at different times equally, i.e., the discount rate is zero.)

Should this person accept bad jobs? To answer this question, we can reason backwards from time $t = 10$.

- **At time 10,** the value of accepting a good job is $100; the value of accepting a bad job is $44; the value of rejecting the job that is available is zero. Therefore, if she is still unemployed in the last period, she should accept whatever job she is offered at that time.

- **At time 9,** the value of accepting a good job is $200 (because that job will last for two years); the value of accepting a bad job is $2*44 = $88. The value of rejecting a job offer is $0 now, plus the value of waiting for the next job offer, which will either be $44 with 50% probability or $100 with 50% probability, for an average ('expected') value of 0.5*($100+$44) = $72. Therefore regardless of whether the job available at time 9 is good or bad, it is better to accept that offer than wait for a better one.

- **At time 8,** the value of accepting a good job is $300 (it will last for three years); the value of accepting a bad job is $3*44 = $132. The value of rejecting a job offer is $0 now, plus the value of waiting for a job offer at time 9. Since we have already concluded that offers at time 9 should be accepted, the expected value of waiting for a job offer at time 9 is 0.5*($200+$88) = $144. Therefore at time 8, it is more valuable to wait for the next offer than to accept a bad job.

It can be verified by continuing to work backwards that bad offers should only be accepted if one is still unemployed at times 9 or 10; they should be rejected at all times up to $t = 8$. The intuition is that if one expects to work in a job for a long time, this makes it more valuable to be picky about what job to accept.

A dynamic optimization problem of this kind is called an optimal stopping problem, because the issue at hand is when to stop waiting for a better offer. Search theory is the field of microeconomics that applies problems of this type to contexts like shopping, job search, and marriage.
An example of backward induction in game theory

Consider the ultimatum game, where one player proposes to split a dollar with another. The first player (the proposer) suggests a division of the dollar between the two players. The second player is then given the option to either accept the split or reject it. If the second player accepts, both get the amount suggested by the proposer. If rejected, neither receives anything.

Consider the actions of the second player given any arbitrary proposal by the first player (that gives the second player more than zero). Since the only choice the second player has at each of these points in the game is to choose between something and nothing, one can expect that the second will accept. Given that the second will accept all proposals offered by the first (that give the second anything at all), the first ought to propose giving the second as little as possible. This is the unique subgame perfect equilibrium of the Ultimatum Game. (However, the Ultimatum Game does have several other Nash equilibria which are not subgame perfect.)

See also centipede game.

Backward induction and economic entry

Consider a dynamic game in which the players are an incumbent firm in an industry and a potential entrant to that industry. As it stands, the incumbent has a monopoly over the industry and does not want to lose some of its market share to the entrant. If the entrant chooses not to enter, the payoff to the incumbent is high (it maintains its monopoly) and the entrant neither loses nor gains (its payoff is zero). If the entrant enters, the incumbent can "fight" or "accommodate" the entrant. It will fight by lowering its price, running the entrant out of business (and incurring exit costs — a negative payoff) and damaging its own profits. If it accommodates the entrant it will lose some of its sales, but a high price will be maintained and it will receive greater profits than by lowering its price (but lower than monopoly profits).

Say that, the best response of the incumbent is to accommodate if the entrant enters. If the incumbent accommodates, the best response of the entrant is to enter (and gain profit). Hence the strategy profile in which the entrant enters and the incumbent accommodates if the entrant enters is a Nash equilibrium consistent with backward induction. However, if the incumbent is going to fight, the best response of the entrant is to not enter, and if the entrant does not enter, it does not matter what the incumbent chooses to do in the hypothetical case that the incumbent does enter. Hence the strategy profile in which the incumbent fights if the entrant enters, but the entrant does not enter is also a Nash equilibrium. However, were the entrant to deviate and enter, the incumbent's best response is to accommodate—the threat of fighting is not credible. This second Nash equilibrium can therefore be eliminated by backward induction.

A paradox of backward induction

The unexpected hanging paradox is a paradox related to backward induction. Suppose a prisoner is told that she will be hanged sometime between Monday and Friday of next week. However, the exact day will be a surprise (i.e. she will not know the night before that she will be executed the next day). The prisoner, interested in outsmarting her executioner, attempts to determine which day the execution will occur.

She reasons that it cannot occur on Friday, since if it had not occurred by the end of Thursday, she would know the execution would be on Friday. Therefore she can eliminate Friday as a possibility. With Friday eliminated, she decides that it cannot occur on Thursday, since if it had not occurred on Wednesday, she would know that it had to be on Thursday. Therefore she can eliminate Thursday. This reasoning proceeds until she has eliminated all possibilities. She concludes that she will not be hanged next week.

To her surprise, she is hanged on Wednesday.

Here the prisoner reasons by backward induction, but seems to come to a false conclusion. Note, however, that the description of the problem assumes it is possible to surprise someone who is performing backward induction. The
mathematical theory of backward induction does not make this assumption, so the paradox does not call into question the results of this theory. Nonetheless, this paradox has received some substantial discussion by philosophers. This paradox is similar to a shorter one: "guess heads or tails but not what you think". While it is easy to come with a flip fulfilling the statement, described reasoning leads to conclusion that there is no way to do that.

Notes


Production system

A production system (or production rule system) is a computer program typically used to provide some form of artificial intelligence, which consists primarily of a set of rules about behavior. These rules, termed productions, are a basic representation found useful in automated planning, expert systems and action selection. A production system provides the mechanism necessary to execute productions in order to achieve some goal for the system.

Productions consist of two parts: a sensory precondition (or "IF" statement) and an action (or "THEN"). If a production's precondition matches the current state of the world, then the production is said to be triggered. If a production's action is executed, it is said to have fired. A production system also contains a database, sometimes called working memory, which maintains data about current state or knowledge, and a rule interpreter. The rule interpreter must provide a mechanism for prioritizing productions when more than one is triggered.

Basic operation

Rule interpreters generally execute a forward chaining algorithm for selecting productions to execute to meet current goals, which can include updating the system's data or beliefs. The condition portion of each rule (left-hand side or LHS) is tested against the current state of the working memory.

In idealized or data-oriented production systems, there is an assumption that any triggered conditions should be executed: the consequent actions (right-hand side or RHS) will update the agent's knowledge, removing or adding data to the working memory. The system stops processing either when the user interrupts the forward chaining loop; when a given number of cycles has been performed; when a "halt" RHS is executed, or when no rules have LHSs that are true.

Real-time and expert systems, in contrast, often have to choose between mutually exclusive productions --- since actions take time, only one action can be taken, or (in the case of an expert system) recommended. In such systems, the rule interpreter, or inference engine, cycles through two steps: matching production rules against the database, followed by selecting which of the matched rules to apply and executing the selected actions.
Matching production rules against working memory

Production systems may vary on the expressive power of conditions in production rules. Accordingly, the pattern matching algorithm which collects production rules with matched conditions may range from the naive -- trying all rules in sequence, stopping at the first match -- to the optimized, in which rules are "compiled" into a network of inter-related conditions.

The latter is illustrated by the RETE algorithm, designed by Charles L. Forgy in 1983, which is used in a series of production systems, called OPS and originally developed at Carnegie Mellon University culminating in OPS5 in the early eighties. OPS5 may be viewed as a full-fledged programming language for production system programming.

Choosing which rules to evaluate

Production systems may also differ in the final selection of production rules to execute, or fire. The collection of rules resulting from the previous matching algorithm is called the conflict set, and the selection process is also called a conflict resolution strategy.

Here again, such strategies may vary from the simple -- use the order in which production rules were written; assign weights or priorities to production rules and sort the conflict set accordingly -- to the complex -- sort the conflict set according to the times at which production rules were previously fired; or according to the extent of the modifications induced by their RHSs. Whichever conflict resolution strategy is implemented, the method is indeed crucial to the efficiency and correctness of the production system.

Using production systems

The use of production systems varies from simple string rewriting rules to the modeling of human cognitive processes, from term rewriting and reduction systems to expert systems.

A simple string rewriting production system example

This example shows a set of production rules for reversing a string from an alphabet that does not contain the symbols "$" and "*" (which are used as marker symbols).

P1: $$ -> *
P2: *$ -> *
P3: *x -> x*
P4: * -> null & halt
P5: $xy -> y$x
P6: null -> $

In this example, production rules are chosen for testing according to their order in this production list. For each rule, the input string is examined from left to right with a moving window to find a match with the LHS of the production rule. When a match is found, the matched substring in the input string is replaced with the RHS of the production rule. In this production system, x and y are variables matching any character of the input string alphabet. Matching resumes with P1 once the replacement has been made.

The string "ABC", for instance, undergoes the following sequence of transformations under these production rules:

$ABC (P6)
B$AC (P5)
BC$A (P5)
$BC$A (P6)
C$B$A (P5)
In such a simple system, the ordering of the production rules is crucial. Often, the lack of control structure makes production systems difficult to design. It is, of course, possible to add control structure to the production systems model, namely in the inference engine, or in the working memory.

**An OPS5 production rule example**

In a toy simulation world where a monkey in a room can grab different objects and climb on others, an example production rule to grab an object suspended from the ceiling would look like:

```plaintext
(p Holds::Object-Ceiling
  {(goal ^status active ^type holds ^objid <O1>) <goal>}
  {(physical-object
    ^id <O1>
    ^weight light
    ^at <p>
    ^on ceiling) <object-1>}
  {(physical-object ^id ladder ^at <p> ^on floor) <object-2>}
  {(monkey ^on ladder ^holds NIL) <monkey>}
  -(physical-object ^on <O1>)
  -->
  (write (crlf) Grab <O1> (crlf))
  (modify <object1> ^on NIL)
  (modify <monkey> ^holds <O1>)
  (modify <goal> ^status satisfied)
)
```

In this example, data in working memory is structured and variables appear between angle brackets. The name of the data structure, such as "goal" and "physical-object", is the first literal in conditions; the fields of a structure are prefixed with "^". The "-" indicates a negative condition.

Production rules in OPS5 apply to all instances of data structures that match conditions and conform to variable bindings. In this example, should several objects be suspended from the ceiling, each with a different ladder nearby supporting an empty-handed monkey, the conflict set would contain as many production rule instances derived from the same production "Holds::Object-Ceiling". The conflict resolution step would later select which production instances to fire.

Note that the binding of variables resulting from the pattern matching in the LHS is used in the RHS to refer to the data to be modified. Note also that the working memory contains explicit control structure data in the form of "goal" data structure instances. In the example, once a monkey holds the suspended object, the status of the goal is set to "satisfied" and the same production rule can no longer apply as its first condition fails.
Related systems

- CLIPS: public domain software tool for building expert systems.
- ILOG rules: a business rule management system.
- JBoss Drools: a open-source business rule management system (BRMS).
- JESS: a rule engine for the Java platform - it is a superset of CLIPS programming language.
- Prolog: a general purpose logic programming language.
- Lisa: a rule engine written in Common Lisp.
- DTRules: a Decision Table based, open-sourced rule engine for Java.
- OpenL Tablets: business centric rules and open source BRMS.

References


Production Rule Representation

The Production Rule Representation (PRR) is a proposed standard of the Object Management Group (OMG) to provide a vendor-neutral rule-model representation in UML for production rules as used in forward-chaining rule engines.

History

The OMG set up a Business Rules Working Group in 2002 as the first standards body to recognize the importance of the "Business Rules Approach". It issued 2 main RFPs in 2003 – a standard for modeling production rules (PRR), and a standard for modeling business rules as business documentation (BSBR, now SBVR).

PRR was mostly defined by and for vendors of Business Rule Engines (BREs) (sometimes termed Business Rules Engine(s), like in Wikipedia). Contributors have included all the major BRE vendors, members of RuleML, and leading UML vendors.

Evolution

1. The PRR RFP originally suggested that PRR use a combination of UML OCL and Action Semantics for rule conditions and actions. However, expecting modellers to learn 2 relatively obscure UML languages in order to define a production rule proved unpalatable. Therefore PRR OCL was defined that included OCL extensions for simple rule actions (as well as external functions). PRR OCL is currently considered "non-normative" i.e. is not part of the PRR standard per se. PRR beta applies just to a PRR Core that excludes an explicit expression language.

2. The PRR RFP envisaged covering both forward and backward chaining rule engines. However, the lack of vendor support for / interest in backward chaining caused this to be revise to forward chaining and "sequential" semantics. The latter is simply the scripting mode provided by many BPM tools, where rules are listed and executed sequentially as if programmed. This provides PRR with better compatibility with typical BPM scripting engines (and acknowledges the fact that most BREs today support a "sequential" mode of operation, improving
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 matchings 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 is called an instantiation of the rule.

In many applications, where large volume of data are concerned and/or when performance time considerations are critical, the computation of the conflict set is a non-trivial problem. Earlier research work on inference engines focused on better algorithms for matching rules to data. The Rete algorithm, developed by Charles Forgy, is an
example of such a matching algorithm; it was used in the OPS series of production system languages. Daniel P. Miranker later improved on Rete with another algorithm, TREAT, which combined it with optimization techniques derived from relational database systems.

The inference engine then passes along the conflict set to the second state, select rules. In this state, the inference engine applies some selection strategy to determine which rules will actually be executed. The selection strategy can be hard-coded into the engine or may be specified as part of the model. In the larger context of AI, these selection strategies are often referred to as heuristics following Allen Newell's Unified theory of cognition.

In OPS5, for instance, a choice of two conflict resolution strategies is presented to the programmer. The LEX strategy orders instantiations on the basis of recency of the time tags attached to their data items. Instantiations with data items having recently matched rules in previous cycles are considered with higher priority. Within this ordering, instantiations are further sorted on the complexity of the conditions in the rule. The other strategy, MEA, puts special emphasis on the recency of working memory elements that match the first condition of the rule. (The latter heuristic is heavily used in means-ends analysis.)

Finally the selected instantiations are passed over to the third state, execute rules. The inference engine executes or fires the selected rules, with the instantiation's data items as parameters. Usually the actions in the right-hand side of a rule change the data store, but they may also trigger further processing outside of the inference engine (interacting with users through a graphical user interface or calling local or remote programs, for instance). Since the data store is usually updated by firing rules, a different set of rules will match during the next cycle after these actions are performed.

The inference engine then cycles back to the first state and is ready to start over again. This control mechanism is referred to as the recognize-act cycle. The inference engine stops either on a given number of cycles, controlled by the operator, or on a quiescent state of the data store when no rules match the data.

**Data-driven computation versus procedural control**

The inference engine control is based on the frequent reevaluation 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).
Fuzzy logic

Fuzzy logic is a form of many-valued logic or probabilistic logic; it deals with reasoning that is approximate rather than fixed and exact. In contrast with traditional logic they can have varying values, where binary sets have two-valued logic, true or false, fuzzy logic variables may have a truth value that ranges in degree between 0 and 1. Fuzzy logic has been extended to handle the concept of partial truth, where the truth value may range between completely true and completely false. Furthermore, when linguistic variables are used, these degrees may be managed by specific functions.

Fuzzy logic began with the 1965 proposal of fuzzy set theory by Lotfi Zadeh. Fuzzy logic has been applied to many fields, from control theory to artificial intelligence.

Overview

Fuzzy logic allows for approximate values and inferences as well as incomplete or ambiguous data (fuzzy data) as opposed to only relying on crisp data (binary yes/no choices).

Degrees of truth

Fuzzy logic and probabilistic logic are mathematically similar — both have truth values ranging between 0 and 1 — but conceptually distinct, due to different interpretations—see interpretations of probability theory. Fuzzy logic corresponds to "degrees of truth", while probabilistic logic corresponds to "probability, likelihood"; as these differ, fuzzy logic and probabilistic logic yield different models of the same real-world situations.

Both degrees of truth and probabilities range between 0 and 1 and hence may seem similar at first. For example, let a 100 ml glass contain 30 ml of water. Then we may consider two concepts: Empty and Full. The meaning of each of them can be represented by a certain fuzzy set. Then one might define the glass as being 0.7 empty and 0.3 full. Note that the concept of emptiness would be subjective and thus would depend on the observer or designer. Another designer might equally well design a set membership function where the glass would be considered full for all values down to 50 ml. It is essential to realize that fuzzy logic uses truth degrees as a mathematical model of the vagueness phenomenon while probability is a mathematical model of ignorance.

Applying truth values

A basic application might characterize subranges of a continuous variable. For instance, a temperature measurement for anti-lock brakes might have several separate membership functions defining particular temperature ranges needed to control the brakes properly. Each function maps the same temperature value to a truth value in the 0 to 1 range. These truth values can then be used to determine how the brakes should be controlled.

In this image, the meanings of the expressions cold, warm, and hot are represented by functions mapping a temperature scale. A point on that scale has three "truth values"—one for each of the three functions. The vertical
line in the image represents a particular temperature that the three arrows (truth values) gauge. Since the red arrow points to zero, this temperature may be interpreted as "not hot". The orange arrow (pointing at 0.2) may describe it as "slightly warm" and the blue arrow (pointing at 0.8) "fairly cold".

**Linguistic variables**

While variables in mathematics usually take numerical values, in fuzzy logic applications, the non-numeric linguistic variables are often used to facilitate the expression of rules and facts.\[^4\]

A linguistic variable such as age may have a value such as young or its antonym old. However, the great utility of linguistic variables is that they can be modified via linguistic hedges applied to primary terms. The linguistic hedges can be associated with certain functions.

**Example**

Fuzzy set theory defines fuzzy operators on fuzzy sets. The problem in applying this is that the appropriate fuzzy operator may not be known. For this reason, fuzzy logic usually uses IF-THEN rules, or constructs that are equivalent, such as fuzzy associative matrices.

Rules are usually expressed in the form:

**IF** variable **IS** property **THEN** action

For example, a simple temperature regulator that uses a fan might look like this:

- IF temperature IS very cold THEN stop fan
- IF temperature IS cold THEN turn down fan
- IF temperature IS normal THEN maintain level
- IF temperature IS hot THEN speed up fan

There is no "ELSE" – all of the rules are evaluated, because the temperature might be "cold" and "normal" at the same time to different degrees.

The AND, OR, and NOT operators of boolean logic exist in fuzzy logic, usually defined as the minimum, maximum, and complement; when they are defined this way, they are called the Zadeh operators. So for the fuzzy variables x and y:

- \( \text{NOT } x = (1 - \text{truth}(x)) \)
- \( x \text{ AND } y = \text{minimum} (\text{truth}(x), \text{truth}(y)) \)
- \( x \text{ OR } y = \text{maximum} (\text{truth}(x), \text{truth}(y)) \)

There are also other operators, more linguistic in nature, called hedges that can be applied. These are generally adverbs such as "very", or "somewhat", which modify the meaning of a set using a mathematical formula.
Logical analysis

In mathematical logic, there are several formal systems of "fuzzy logic"; most of them belong among so-called t-norm fuzzy logics.

Propositional fuzzy logics

The most important propositional fuzzy logics are:

- Monoidal t-norm-based propositional fuzzy logic MTL is an axiomatization of logic where conjunction is defined by a left continuous t-norm, and implication is defined as the residuum of the t-norm. Its models correspond to MTL-algebras that are prelinear commutative bounded integral residuated lattices.

- Basic propositional fuzzy logic BL is an extension of MTL logic where conjunction is defined by a continuous t-norm, and implication is also defined as the residuum of the t-norm. Its models correspond to BL-algebras.

- Łukasiewicz fuzzy logic is the extension of basic fuzzy logic BL where standard conjunction is the Łukasiewicz t-norm. It has the axioms of basic fuzzy logic plus an axiom of double negation, and its models correspond to MV-algebras.

- Gödel fuzzy logic is the extension of basic fuzzy logic BL where conjunction is Gödel t-norm. It has the axioms of BL plus an axiom of idempotence of conjunction, and its models are called G-algebras.

- Product fuzzy logic is the extension of basic fuzzy logic BL where conjunction is product t-norm. It has the axioms of BL plus another axiom for cancellativity of conjunction, and its models are called product algebras.

- Fuzzy logic with evaluated syntax (sometimes also called Pavelka's logic), denoted by EVŁ, is a further generalization of mathematical fuzzy logic. While the above kinds of fuzzy logic have traditional syntax and many-valued semantics, in EVŁ is evaluated also syntax. This means that each formula has an evaluation. Axiomatization of EVŁ stems from Łukasiewicz fuzzy logic. A generalization of classical Gödel completeness theorem is provable in EVŁ.

Predicate fuzzy logics

These extend the above-mentioned fuzzy logics by adding universal and existential quantifiers in a manner similar to the way that predicate logic is created from propositional logic. The semantics of the universal (resp. existential) quantifier in t-norm fuzzy logics is the infimum (resp. supremum) of the truth degrees of the instances of the quantified subformula.

Decidability issues for fuzzy logic

The notions of a "decidable subset" and "recursively enumerable subset" are basic ones for classical mathematics and classical logic. Then, the question of a suitable extension of these concepts to fuzzy set theory arises. A first proposal in such a direction was made by E.S. Santos by the notions of fuzzy Turing machine, Markov normal fuzzy algorithm and fuzzy program (see Santos 1970). Successively, L. Biacino and G. Gerla argued that the proposed definitions are rather questionable and therefore they proposed the following ones. Denote by $\hat{U}$ the set of rational numbers in $[0,1]$. Then a fuzzy subset $s : S \rightarrow [0,1]$ of a set $S$ is recursively enumerable if a recursive map $h : S \times N \rightarrow \hat{U}$ exists such that, for every $x$ in $S$, the function $ht(x,n)$ is increasing with respect to $n$ and $s(x) = \lim n h(x,n)$. We say that $s$ is decidable if both $s$ and its complement $\neg s$ are recursively enumerable. An extension of such a theory to the general case of the L-subsets is possible (see Gerla 2006). The proposed definitions are well related with fuzzy logic. Indeed, the following theorem holds true (provided that the deduction apparatus of the considered fuzzy logic satisfies some obvious effectiveness property).

**Theorem.** Any axiomatizable fuzzy theory is recursively enumerable. In particular, the fuzzy set of logically true formulas is recursively enumerable in spite of the fact that the crisp set of valid formulas is not recursively enumerable, in general. Moreover, any axiomatizable and complete theory is decidable.
It is an open question to give supports for a *Church thesis* for fuzzy mathematics the proposed notion of recursive enumerability for fuzzy subsets is the adequate one. To this aim, an extension of the notions of fuzzy grammar and fuzzy Turing machine should be necessary (see for example Wiedermann's paper). Another open question is to start from this notion to find an extension of Gödel's theorems to fuzzy logic.

### Fuzzy databases

Once fuzzy relations are defined, it is possible to develop fuzzy relational databases. The first fuzzy relational database, FRDB, appeared in Maria Zemankova's dissertation. Later, some other models arose like the Buckles-Petry model, the Prade-Testemale Model, the Umano-Fukami model or the GEFRED model by J.M. Medina, M.A. Vila et al. In the context of fuzzy databases, some fuzzy querying languages have been defined, highlighting the SQLf by P. Bosc et al. and the FSQL by J. Galindo et al. These languages define some structures in order to include fuzzy aspects in the SQL statements, like fuzzy conditions, fuzzy comparators, fuzzy constants, fuzzy constraints, fuzzy thresholds, linguistic labels and so on.

### Comparison to probability

Fuzzy logic and probability are different ways of expressing uncertainty. While both fuzzy logic and probability theory can be used to represent subjective belief, fuzzy set theory uses the concept of fuzzy set membership (i.e., *how much* a variable is in a set), and probability theory uses the concept of subjective probability (i.e., *how probable* do I think that a variable is in a set). While this distinction is mostly philosophical, the fuzzy-logic-derived possibility measure is inherently different from the probability measure, hence they are not *directly* equivalent. However, many statisticians are persuaded by the work of Bruno de Finetti that only one kind of mathematical uncertainty is needed and thus fuzzy logic is unnecessary. On the other hand, Bart Kosko argues that probability is a subtheory of fuzzy logic, as probability only handles one kind of uncertainty. He also claims to have proven a derivation of Bayes’ theorem from the concept of fuzzy subsethood. Lotfi Zadeh argues that fuzzy logic is different in character from probability, and is not a replacement for it. He fuzzified probability to fuzzy probability and also generalized it to what is called possibility theory. (cf. More generally, fuzzy logic is one of many different proposed extensions to classical logic, known as probabilistic logics, intended to deal with issues of uncertainty in classical logic, the inapplicability of probability theory in many domains, and the paradoxes of Dempster-Shafer theory.

### Notes


### Bibliography


**External links**

**Additional articles**

- Formal fuzzy logic (http://en.citizendium.org/wiki/Formal_fuzzy_logic) - article at Citizendium
- Fuzzy Logic (http://www.scholarpedia.org/article/Fuzzy_Logic) - article at Scholarpedia
- Modeling With Words (http://www.scholarpedia.org/article/Modeling_with_words) - article at Scholarpedia
- Fuzzy logic (http://plato.stanford.edu/entries/logic-fuzzy/) - article at Stanford Encyclopedia of Philosophy
- Fuzzy Logic and the Internet of Things: I-o-T (http://www.i-o-t.org/post/WEB_3)

**Applications**

- Research conference paper providing a way to establish a fuzzy workload plan while dealing with project planning problem under uncertainty (http://www.enim.fr/mosim2010/articles/279.pdf)
- Research article that describes how industrial foresight could be integrated into capital budgeting with intelligent agents and Fuzzy Logic (http://econpapers.repec.org/paper/amrwpaper/398.htm)
- A doctoral dissertation describing how Fuzzy Logic can be applied in profitability analysis of very large industrial investments (http://econpapers.repec.org/paper/pramprapa/4328.htm)
- A method for asset valuation that uses fuzzy logic and fuzzy numbers for real option valuation (http://payoffmethod.com)
Fuzzy control system

A fuzzy control system is a control system based on fuzzy logic—a mathematical system that analyzes analog input values in terms of logical variables that take on continuous values between 0 and 1, in contrast to classical or digital logic, which operates on discrete values of either 1 or 0 (true or false, respectively).

Overview

Fuzzy logic is widely used in machine control. The term itself inspires a certain skepticism, sounding equivalent to "half-baked logic" or "bogus logic", but the "fuzzy" part does not refer to a lack of rigour in the method, rather to the fact that the logic involved can deal with concepts that cannot be expressed as "true" or "false" but rather as "partially true". Although genetic algorithms and neural networks can perform just as well as fuzzy logic in many cases, fuzzy logic has the advantage that the solution to the problem can be cast in terms that human operators can understand, so that their experience can be used in the design of the controller. This makes it easier to mechanize tasks that are already successfully performed by humans.

History and applications

Fuzzy logic was first proposed by Lotfi A. Zadeh of the University of California at Berkeley in a 1965 paper. He elaborated on his ideas in a 1973 paper that introduced the concept of "linguistic variables", which in this article equates to a variable defined as a fuzzy set. Other research followed, with the first industrial application, a cement kiln built in Denmark, coming on line in 1975.

Fuzzy systems were largely ignored in the U.S. because they were associated with artificial intelligence, a field that periodically oversells itself, especially in the mid-1980s, resulting in a lack of credibility within the commercial domain.

The Japanese did not have this prejudice. Interest in fuzzy systems was sparked by Seiji Yasunobu and Soji Miyamoto of Hitachi, who in 1985 provided simulations that demonstrated the superiority of fuzzy control systems for the Sendai railway. Their ideas were adopted, and fuzzy systems were used to control accelerating, braking, and stopping when the line opened in 1987.

Another event in 1987 helped promote interest in fuzzy systems. During an international meeting of fuzzy researchers in Tokyo that year, Takeshi Yamakawa demonstrated the use of fuzzy control, through a set of simple dedicated fuzzy logic chips, in an "inverted pendulum" experiment. This is a classic control problem, in which a vehicle tries to keep a pole mounted on its top by a hinge upright by moving back and forth.

Observers were impressed with this demonstration, as well as later experiments by Yamakawa in which he mounted a wine glass containing water or even a live mouse to the top of the pendulum. The system maintained stability in both cases. Yamakawa eventually went on to organize his own fuzzy-systems research lab to help exploit his patents in the field.

Following such demonstrations, Japanese engineers developed a wide range of fuzzy systems for both industrial and consumer applications. In 1988 Japan established the Laboratory for International Fuzzy Engineering (LIFE), a cooperative arrangement between 48 companies to pursue fuzzy research. The automotive company Volkswagen was the only foreign corporate member of LIFE, dispatching a researcher for a duration of three years.

Japanese consumer goods often incorporate fuzzy systems. Matsushita vacuum cleaners use microcontrollers running fuzzy algorithms to interrogate dust sensors and adjust suction power accordingly. Hitachi washing machines use fuzzy controllers to load-weight, fabric-mix, and dirt sensors and automatically set the wash cycle for the best use of power, water, and detergent.
As a more specific example, Canon developed an autofocusing camera that uses a charge-coupled device (CCD) to measure the clarity of the image in six regions of its field of view and use the information provided to determine if the image is in focus. It also tracks the rate of change of lens movement during focusing, and controls its speed to prevent overshoot.

The camera’s fuzzy control system uses 12 inputs: 6 to obtain the current clarity data provided by the CCD and 6 to measure the rate of change of lens movement. The output is the position of the lens. The fuzzy control system uses 13 rules and requires 1.1 kilobytes of memory.

As another example of a practical system, an industrial air conditioner designed by Mitsubishi uses 25 heating rules and 25 cooling rules. A temperature sensor provides input, with control outputs fed to an inverter, a compressor valve, and a fan motor. Compared to the previous design, the fuzzy controller heats and cools five times faster, reduces power consumption by 24%, increases temperature stability by a factor of two, and uses fewer sensors.

The enthusiasm of the Japanese for fuzzy logic is reflected in the wide range of other applications they have investigated or implemented: character and handwriting recognition; optical fuzzy systems; robots, including one for making Japanese flower arrangements; voice-controlled robot helicopters, this being no mean feat, as hovering is a "balancing act" rather similar to the inverted pendulum problem; control of flow of powders in film manufacture; elevator systems; and so on.

Work on fuzzy systems is also proceeding in the US and Europe, though not with the same enthusiasm shown in Japan. The US Environmental Protection Agency has investigated fuzzy control for energy-efficient motors, and NASA has studied fuzzy control for automated space docking: simulations show that a fuzzy control system can greatly reduce fuel consumption. Firms such as Boeing, General Motors, Allen-Bradley, Chrysler, Eaton, and Whirlpool have worked on fuzzy logic for use in low-power refrigerators, improved automotive transmissions, and energy-efficient electric motors.

In 1995 Maytag introduced an "intelligent" dishwasher based on a fuzzy controller and a "one-stop sensing module" that combines a thermistor, for temperature measurement; a conductivity sensor, to measure detergent level from the ions present in the wash; a turbidity sensor that measures scattered and transmitted light to measure the soiling of the wash; and a magnetostrictive sensor to read spin rate. The system determines the optimum wash cycle for any load to obtain the best results with the least amount of energy, detergent, and water. It even adjusts for dried-on foods by tracking the last time the door was opened, and estimates the number of dishes by the number of times the door was opened.

Research and development is also continuing on fuzzy applications in software, as opposed to firmware, design, including fuzzy expert systems and integration of fuzzy logic with neural-network and so-called adaptive "genetic" software systems, with the ultimate goal of building "self-learning" fuzzy-control systems.

**Fuzzy sets**

The input variables in a fuzzy control system are in general mapped by sets of membership functions similar to this, known as "fuzzy sets". The process of converting a crisp input value to a fuzzy value is called "fuzzification".

A control system may also have various types of switch, or "ON-OFF", inputs along with its analog inputs, and such switch inputs of course will always have a truth value equal to either 1 or 0, but the scheme can deal with them as simplified fuzzy functions that happen to be either one value or another.

Given "mappings" of input variables into membership functions and truth values, the microcontroller then makes decisions for what action to take, based on a set of "rules", each of the form:

```
IF brake temperature IS warm AND speed IS not very fast
THEN brake pressure IS slightly decreased.
```
In this example, the two input variables are "brake temperature" and "speed" that have values defined as fuzzy sets. The output variable, "brake pressure" is also defined by a fuzzy set that can have values like "static" or "slightly increased" or "slightly decreased" etc.

This rule by itself is very puzzling since it looks like it could be used without bothering with fuzzy logic, but remember that the decision is based on a set of rules:

- All the rules that apply are invoked, using the membership functions and truth values obtained from the inputs, to determine the result of the rule.
- This result in turn will be mapped into a membership function and truth value controlling the output variable.
- These results are combined to give a specific ("crisp") answer, the actual brake pressure, a procedure known as "defuzzification".

This combination of fuzzy operations and rule-based "inference" describes a "fuzzy expert system".

Traditional control systems are based on mathematical models in which the control system is described using one or more differential equations that define the system response to its inputs. Such systems are often implemented as "PID controllers" (proportional-integral-derivative controllers). They are the products of decades of development and theoretical analysis, and are highly effective.

If PID and other traditional control systems are so well-developed, why bother with fuzzy control? It has some advantages. In many cases, the mathematical model of the control process may not exist, or may be too "expensive" in terms of computer processing power and memory, and a system based on empirical rules may be more effective.

Furthermore, fuzzy logic is well suited to low-cost implementations based on cheap sensors, low-resolution analog-to-digital converters, and 4-bit or 8-bit one-chip microcontroller chips. Such systems can be easily upgraded by adding new rules to improve performance or add new features. In many cases, fuzzy control can be used to improve existing traditional controller systems by adding an extra layer of intelligence to the current control method.

**Fuzzy control in detail**

Fuzzy controllers are very simple conceptually. They consist of an input stage, a processing stage, and an output stage. The input stage maps sensor or other inputs, such as switches, thumbwheels, and so on, to the appropriate membership functions and truth values. The processing stage invokes each appropriate rule and generates a result for each, then combines the results of the rules. Finally, the output stage converts the combined result back into a specific control output value.

The most common shape of membership functions is triangular, although trapezoidal and bell curves are also used, but the shape is generally less important than the number of curves and their placement. From three to seven curves are generally appropriate to cover the required range of an input value, or the "universe of discourse" in fuzzy jargon.

As discussed earlier, the processing stage is based on a collection of logic rules in the form of IF-THEN statements, where the IF part is called the "antecedent" and the THEN part is called the "consequent". Typical fuzzy control systems have dozens of rules.

Consider a rule for a thermostat:

| IF (temperature is "cold") THEN (heater is "high") |

This rule uses the truth value of the "temperature" input, which is some truth value of "cold", to generate a result in the fuzzy set for the "heater" output, which is some value of "high". This result is used with the results of other rules to finally generate the crisp composite output. Obviously, the greater the truth value of "cold", the higher the truth value of "high", though this does not necessarily mean that the output itself will be set to "high" since this is only one rule among many. In some cases, the membership functions can be modified by "hedges" that are equivalent to adjectives. Common hedges include "about", "near", "close to", "approximately", "very", "slightly", "too", "extremely", and "somewhat". These operations may have precise definitions, though the definitions can vary
considerably between different implementations. "Very", for one example, squares membership functions; since the membership values are always less than 1, this narrows the membership function. "Extremely" cubes the values to give greater narrowing, while "somewhat" broadens the function by taking the square root.

In practice, the fuzzy rule sets usually have several antecedents that are combined using fuzzy operators, such as AND, OR, and NOT, though again the definitions tend to vary: AND, in one popular definition, simply uses the minimum weight of all the antecedents, while OR uses the maximum value. There is also a NOT operator that subtracts a membership function from 1 to give the "complementary" function.

There are several ways to define the result of a rule, but one of the most common and simplest is the "max-min" inference method, in which the output membership function is given the truth value generated by the premise.

Rules can be solved in parallel in hardware, or sequentially in software. The results of all the rules that have fired are "defuzzified" to a crisp value by one of several methods. There are dozens, in theory, each with various advantages or drawbacks.

The "centroid" method is very popular, in which the "center of mass" of the result provides the crisp value. Another approach is the "height" method, which takes the value of the biggest contributor. The centroid method favors the rule with the output of greatest area, while the height method obviously favors the rule with the greatest output value.

The diagram below demonstrates max-min inferencing and centroid defuzzification for a system with input variables "x", "y", and "z" and an output variable "n". Note that "mu" is standard fuzzy-logic nomenclature for "truth value":

Notice how each rule provides a result as a truth value of a particular membership function for the output variable. In centroid defuzzification the values are OR'd, that is, the maximum value is used and values are not added, and the results are then combined using a centroid calculation.

Fuzzy control system design is based on empirical methods, basically a methodical approach to trial-and-error. The general process is as follows:

- Document the system's operational specifications and inputs and outputs.
- Document the fuzzy sets for the inputs.
• Document the rule set.
• Determine the defuzzification method.
• Run through test suite to validate system, adjust details as required.
• Complete document and release to production.

As a general example, consider the design of a fuzzy controller for a steam turbine. The block diagram of this control system appears as follows:

The input and output variables map into the following fuzzy set:

- N3: Large negative.
- N2: Medium negative.
- N1: Small negative.
- Z: Zero.
- P1: Small positive.
- P2: Medium positive.
- P3: Large positive.

The rule set includes such rules as:

rule 1: IF temperature IS cool AND pressure IS weak, THEN throttle is P3.
rule 2: IF temperature IS cool AND pressure IS low, THEN throttle is P2.
rule 3: IF temperature IS cool AND pressure IS ok, THEN throttle is Z.
rule 4: IF temperature IS cool AND pressure IS strong, THEN throttle is N2.

In practice, the controller accepts the inputs and maps them into their membership functions and truth values. These mappings are then fed into the rules. If the rule specifies an AND relationship between the mappings of the two input variables, as the examples above do, the minimum of the two is used as the combined truth value; if an OR is
specified, the maximum is used. The appropriate output state is selected and assigned a membership value at the truth level of the premise. The truth values are then defuzzified. For an example, assume the temperature is in the "cool" state, and the pressure is in the "low" and "ok" states. The pressure values ensure that only rules 2 and 3 fire:

The two outputs are then defuzzified through centroid defuzzification:
The output value will adjust the throttle and then the control cycle will begin again to generate the next value.

**Building a fuzzy controller**

Consider implementing with a microcontroller chip a simple feedback controller:

A fuzzy set is defined for the input error variable "e", and the derived change in error, "delta", as well as the "output", as follows:

- **LP**: large positive
- **SP**: small positive
- **ZE**: zero
- **SN**: small negative
- **LN**: large negative

If the error ranges from -1 to +1, with the analog-to-digital converter used having a resolution of 0.25, then the input variable's fuzzy set (which, in this case, also applies to the output variable) can be described very simply as a table, with the error / delta / output values in the top row and the truth values for each membership function arranged in rows beneath:

<table>
<thead>
<tr>
<th>-1</th>
<th>-0.75</th>
<th>-0.5</th>
<th>-0.25</th>
<th>0</th>
<th>0.25</th>
<th>0.5</th>
<th>0.75</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>mu(LP)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.3</td>
<td>0.7</td>
<td>1</td>
</tr>
<tr>
<td>mu(SP)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.3</td>
<td>0.7</td>
<td>1</td>
<td>0.7</td>
<td>0.3</td>
</tr>
<tr>
<td>mu(ZE)</td>
<td>0</td>
<td>0</td>
<td>0.3</td>
<td>0.7</td>
<td>1</td>
<td>0.7</td>
<td>0.3</td>
<td>0</td>
</tr>
<tr>
<td>mu(SN)</td>
<td>0.3</td>
<td>0.7</td>
<td>1</td>
<td>0.7</td>
<td>0.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mu(LN)</td>
<td>1</td>
<td>0.7</td>
<td>0.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

—or, in graphical form (where each "X" has a value of 0.1):

```
LN: XXXXXXXXXXX XXX XXX XXX XXX XXX XXX |
SN: XXXXXXXXXXX XXX XXX XXX XXX XXX |
ZE: XXXXXXXXXXX XXX XXX XXX XXX |
SP: XXXXXXXXXXX XXX XXX XXX |
LP: XXXXXXXXXXX XXX XXX |
```
Suppose this fuzzy system has the following rule base:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IF ( e = ZE ) AND ( \delta = ZE )</td>
<td>( ZE )</td>
</tr>
<tr>
<td>2</td>
<td>IF ( e = ZE ) AND ( \delta = SP )</td>
<td>( SN )</td>
</tr>
<tr>
<td>3</td>
<td>IF ( e = SN ) AND ( \delta = SN )</td>
<td>( LP )</td>
</tr>
<tr>
<td>4</td>
<td>IF ( e = LP ) OR ( \delta = LP )</td>
<td>( LN )</td>
</tr>
</tbody>
</table>

These rules are typical for control applications in that the antecedents consist of the logical combination of the error and error-delta signals, while the consequent is a control command output. The rule outputs can be defuzzified using a discrete centroid computation:

\[
\text{SUM} \left( \sum_{I = 1}^{4} (\mu(I) \times \text{output}(I)) \right) / \sum_{I = 1}^{4} \mu(I)
\]

Now, suppose that at a given time we have:

- \( e = 0.25 \)
- \( \delta = 0.5 \)

Then this gives:

<table>
<thead>
<tr>
<th>( e )</th>
<th>( \delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>0.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Membership Function</th>
<th>( e )</th>
<th>( \delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mu(LP) )</td>
<td>0</td>
<td>0.3</td>
</tr>
<tr>
<td>( \mu(SP) )</td>
<td>0.7</td>
<td>1</td>
</tr>
<tr>
<td>( \mu(ZE) )</td>
<td>0.7</td>
<td>0.3</td>
</tr>
<tr>
<td>( \mu(SN) )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( \mu(LN) )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Plugging this into rule 1 gives:

\[
\mu(1) = \text{MIN}(0.7, 0.3) = 0.3 \\
\text{output}(1) = 0
\]

-- where:

- \( \mu(1) \): Truth value of the result membership function for rule 1. In terms of a centroid calculation, this is the "mass" of this result for this discrete case.
- \( \text{output}(1) \): Value (for rule 1) where the result membership function (ZE) is maximum over the output variable fuzzy set range. That is, in terms of a centroid calculation, the location of the "center of mass" for this individual result. This value is independent of the value of \( \mu \). It simply identifies the location of ZE along the output range.

The other rules give:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>IF ( e = ZE ) AND ( \delta = SP )</td>
<td>( SN )</td>
</tr>
</tbody>
</table>

\[
\mu(2) = \text{MIN}(0.7, 1) = 0.7
\]
Fuzzy control system

output(2) = -0.5

rule 3: IF e = SN AND delta = SN THEN output = LP

\[ \mu(3) = \text{MIN}(0.0, 0.0) = 0 \]
output(3) = 1

rule 4: IF e = LP OR delta = LP THEN output = LN

\[ \mu(4) = \text{MAX}(0.0, 0.3) = 0.3 \]
output(4) = -1

The centroid computation yields:

\[
\frac{\mu(1) \cdot \text{output}(1) + \mu(2) \cdot \text{output}(2) + \mu(3) \cdot \text{output}(3) + \mu(4) \cdot \text{output}(4)}{\mu(1) + \mu(2) + \mu(3) + \mu(4)}
= \frac{(0.3 \cdot 0) + (0.7 \cdot -0.5) + (0 \cdot 1) + (0.3 \cdot -1)}{0.3 + 0.7 + 0 + 0.3}
= -0.5
\]

-- for the final control output. Simple. Of course the hard part is figuring out what rules actually work correctly in practice.

If you have problems figuring out the centroid equation, remember that a centroid is defined by summing all the moments (location times mass) around the center of gravity and equating the sum to zero. So if \( X_0 \) is the center of gravity, \( X_i \) is the location of each mass, and \( M_i \) is each mass, this gives:

\[
\begin{align*}
0 &= (X_1 - X_0) \cdot M_1 + (X_2 - X_0) \cdot M_2 + \ldots + (X_n - X_0) \cdot M_n \\
0 &= (X_1 \cdot M_1 + X_2 \cdot M_2 + \ldots + X_n \cdot M_n) - X_0 \cdot (M_1 + M_2 + \ldots + M_n) \\
X_0 &= X_1 \cdot M_1 + X_2 \cdot M_2 + \ldots + X_n \cdot M_n \\
M_1 + M_2 + \ldots + M_n
\end{align*}
\]

In our example, the values of \( \mu \) correspond to the masses, and the values of \( X \) to location of the masses (\( \mu \), however, only 'corresponds to the masses' if the initial 'mass' of the output functions are all the same/equivalent. If they are not the same, i.e. some are narrow triangles, while others maybe wide trapezoids or shouldered triangles, then the mass or area of the output function must be known or calculated. It is this mass that is then scaled by \( \mu \) and multiplied by its location \( X_i \).

This system can be implemented on a standard microprocessor, but dedicated fuzzy chips are now available. For example, Adaptive Logic INC of San Jose, California, sells a "fuzzy chip", the AL220, that can accept four analog inputs and generate four analog outputs. A block diagram of the chip is shown below:
Antilock brakes

As a first example, consider an anti-lock braking system, directed by a microcontroller chip. The microcontroller has to make decisions based on brake temperature, speed, and other variables in the system.

The variable "temperature" in this system can be subdivided into a range of "states": "cold", "cool", "moderate", "warm", "hot", "very hot". The transition from one state to the next is hard to define.

An arbitrary static threshold might be set to divide "warm" from "hot". For example, at exactly 90 degrees, warm ends and hot begins. But this would result in a discontinuous change when the input value passed over that threshold. The transition wouldn't be smooth, as would be required in braking situations.

The way around this is to make the states fuzzy. That is, allow them to change gradually from one state to the next. In order to do this there must be a dynamic relationship established between different factors.

We start by defining the input temperature states using "membership functions":

ADC: analog-to-digital converter

DAC: digital-to-analog converter

SH: sample/hold
With this scheme, the input variable's state no longer jumps abruptly from one state to the next. Instead, as the temperature changes, it loses value in one membership function while gaining value in the next. In other words, its ranking in the category of cold decreases as it becomes more highly ranked in the warmer category.

At any sampled timeframe, the "truth value" of the brake temperature will almost always be in some degree part of two membership functions: i.e.: '0.6 nominal and 0.4 warm', or '0.7 nominal and 0.3 cool', and so on.

The above example demonstrates a simple application, using the abstraction of values from multiple values. This only represents one kind of data, however, in this case, temperature.

Adding additional sophistication to this braking system, could be done by additional factors such as traction, speed, inertia, set up in dynamic functions, according to the designed fuzzy system.

**Logical interpretation of fuzzy control**

In spite of the appearance there are several difficulties to give a rigorous logical interpretation of the *IF-THEN* rules. As an example, interpret a rule as *IF (temperature is "cold") THEN (heater is "high")* by the first order formula \( \text{Cold}(x) \rightarrow \text{High}(y) \) and assume that \( r \) is an input such that \( \text{Cold}(r) \) is false. Then the formula \( \text{Cold}(r) \rightarrow \text{High}(t) \) is true for any \( t \) and therefore any \( t \) gives a correct control given \( r \). A rigorous logical justification of fuzzy control is given in Hájek's book (see Chapter 7) where fuzzy control is represented as a theory of Hájek's basic logic. Also in Gerla 2005 a logical approach to fuzzy control is proposed based on fuzzy logic programming. Indeed, denote by \( f \) the fuzzy function arising of a IF-THEN systems of rules. Then we can translate this system into fuzzy program in such a way that \( f \) is the interpretation of a vague predicate \( \text{Good}(x,y) \) in the least fuzzy Herbrand model of this program. This gives further useful tools to fuzzy control.

**References**

Further reading


External links

- Introduction to Fuzzy Control (http://inside.mines.edu/~msimoes/documents/Intro_Fuzzy_Logic.pdf)
- Designing Fuzzy Systems with Evolutionary Optimisation (http://www.razorrobotics.com/fuzzy_control_system.html)
- fuzzyTECH (http://www.fuzzytech.com), a commercial fuzzy logic development system containing the specification document for IEC1131-7 (select Fuzzy Application Library)
- IEC 1131-7 CD1 (http://www.fuzzytech.com/binaries/iec1131-7.pdf) IEC 1131-7 CD1 PDF
- fuzzylite (http://code.google.com/p/fuzzylite), A fuzzy logic controller library written in C++.
- JFuzzyLogic (http://jfuzzylogic.sourceforge.net), open source FCL + Fuzzy Logic Package (sourceforge, java)
- AwiFuzz (http://sourceforge.net/projects/awifuzz), open source implementation written in C++ covering all three levels of IEC 61131-7
Artificial neural network

An Artificial Neural Network (ANN), usually called neural network (NN), is a mathematical model or computational model that is inspired by the structure and/or functional aspects of biological neural networks. A neural network consists of an interconnected group of artificial neurons, and it processes information using a connectionist approach to computation. In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase. Modern neural networks are non-linear statistical data modeling tools. They are usually used to model complex relationships between inputs and outputs or to find patterns in data.

Background

The original inspiration for the term Artificial Neural Network came from examination of central nervous systems and their neurons, axons, dendrites, and synapses, which constitute the processing elements of biological neural networks investigated by neuroscience. In an artificial neural network, simple artificial nodes, variously called "neurons", "neurodes", "processing elements" (PEs) or "units", are connected together to form a network of nodes mimicking the biological neural networks — hence the term "artificial neural network".

Because neuroscience is still full of unanswered questions, and since there are many levels of abstraction and therefore many ways to take inspiration from the brain, there is no single formal definition of what an artificial neural network is. Generally, it involves a network of simple processing elements that exhibit complex global behavior determined by connections between processing elements and element parameters. While an artificial neural network does not have to be adaptive per se, its practical use comes with algorithms designed to alter the strength (weights) of the connections in the network to produce a desired signal flow.

These networks are also similar to the biological neural networks in the sense that functions are performed collectively and in parallel by the units, rather than there being a clear delineation of subtasks to which various units are assigned (see also connectionism). Currently, the term Artificial Neural Network (ANN) tends to refer mostly to neural network models employed in statistics, cognitive psychology and artificial intelligence. Neural network models designed with emulation of the central nervous system (CNS) in mind are a subject of theoretical neuroscience and computational neuroscience.

In modern software implementations of artificial neural networks, the approach inspired by biology has been largely abandoned for a more practical approach based on statistics and signal processing. In some of these systems, neural networks or parts of neural networks (such as artificial neurons) are used as components in larger systems that combine both adaptive and non-adaptive elements. While the more general approach of such adaptive systems is
more suitable for real-world problem solving, it has far less to do with the traditional artificial intelligence connectionist models. What they do have in common, however, is the principle of non-linear, distributed, parallel and local processing and adaptation. Historically, the use of neural networks models marked a paradigm shift in the late eighties from high-level (symbolic) artificial intelligence, characterized by expert systems with knowledge embodied in *if-then* rules, to low-level (sub-symbolic) machine learning, characterized by knowledge embodied in the parameters of a dynamical system.

### Models

Neural network models in artificial intelligence are usually referred to as artificial neural networks (ANNs); these are essentially simple mathematical models defining a function $f : \mathbf{X} \rightarrow \mathbf{Y}$ or a distribution over $\mathbf{X}$ or both $\mathbf{X}$ and $\mathbf{Y}$, but sometimes models are also intimately associated with a particular learning algorithm or learning rule. A common use of the phrase ANN model really means the definition of a class of such functions (where members of the class are obtained by varying parameters, connection weights, or specifics of the architecture such as the number of neurons or their connectivity).

### Network function

The word *network* in the term 'artificial neural network' refers to the inter–connections between the neurons in the different layers of each system. An example system has three layers. The first layer has input neurons, which send data via synapses to the second layer of neurons, and then via more synapses to the third layer of output neurons. More complex systems will have more layers of neurons with some having increased layers of input neurons and output neurons. The synapses store parameters called "weights" that manipulate the data in the calculations.

An ANN is typically defined by three types of parameters:

1. The interconnection pattern between different layers of neurons
2. The learning process for updating the weights of the interconnections
3. The activation function that converts a neuron’s weighted input to its output activation.

Mathematically, a neuron’s network function $f(x)$ is defined as a composition of other functions $g_k(x)$, which can further be defined as a composition of other functions. This can be conveniently represented as a network structure, with arrows depicting the dependencies between variables. A widely used type of composition is the nonlinear weighted sum, where $f(x) = K(\sum_i w_i g_i(x))$, where $K$ (commonly referred to as the activation function) is some predefined function, such as the hyperbolic tangent. It will be convenient for the following to refer to a collection of functions $g_k$ as simply a vector $g = (g_1, g_2, ..., g_n)$.

This figure depicts such a decomposition of $f$, with dependencies between variables indicated by arrows. These can be interpreted in two ways.

The first view is the functional view: the input $x$ is transformed into a 3-dimensional vector $h$, which is then transformed into a 2-dimensional vector $g$, which is finally transformed into $f$. This view is most commonly encountered in the context of optimization.

The second view is the probabilistic view: the random variable $F = f(G)$ depends upon the random variable $G = g(H)$, which depends upon $H = k(X)$, which depends upon the random variable $X$. This view is most commonly encountered in the context of graphical models.

The two views are largely equivalent. In either case, for this particular network architecture, the components of individual layers are independent of each other (e.g., the components of $g$ are independent of each other given their input $h$). This naturally enables a degree of parallelism in the implementation.
Networks such as the previous one are commonly called feedforward, because their graph is a directed acyclic graph. Networks with cycles are commonly called recurrent. Such networks are commonly depicted in the manner shown at the top of the figure, where \( f \) is shown as being dependent upon itself. However, an implied temporal dependence is not shown.

**Learning**

What has attracted the most interest in neural networks is the possibility of learning. Given a specific task to solve, and a class of functions \( F \), learning means using a set of observations to find \( f^* \in F \) which solves the task in some optimal sense.

This entails defining a cost function \( C: F \rightarrow \mathbb{R} \) such that, for the optimal solution \( f^* \), \( C(f^*) \leq C(f) \ \forall f \in F \) i.e., no solution has a cost less than the cost of the optimal solution (see Mathematical optimization).

The cost function \( C \) is an important concept in learning, as it is a measure of how far away a particular solution is from an optimal solution to the problem to be solved. Learning algorithms search through the solution space to find a function that has the smallest possible cost.

For applications where the solution is dependent on some data, the cost must necessarily be a function of the observations, otherwise we would not be modelling anything related to the data. It is frequently defined as a statistic to which only approximations can be made. As a simple example, consider the problem of finding the model \( f \), which minimizes \( C = \mathbb{E}[ (f(x) - y)^2 ] \) for data pairs \((x,y)\) drawn from some distribution \( \mathcal{D} \). In practical situations we would only have \( N \) samples from \( \mathcal{D} \) and thus, for the above example, we would only minimize \( \hat{C} = \frac{1}{N} \sum_{i=1}^{N} (f(x_i) - y_i)^2 \). Thus, the cost is minimized over a sample of the data rather than the entire data set. When \( N \rightarrow \infty \) some form of online machine learning must be used, where the cost is partially minimized as each new example is seen. While online machine learning is often used when \( \mathcal{D} \) is fixed, it is most useful in the case where the distribution changes slowly over time. In neural network methods, some form of online machine learning is frequently used for finite datasets.

**Choosing a cost function**

While it is possible to define some arbitrary, ad hoc cost function, frequently a particular cost will be used, either because it has desirable properties (such as convexity) or because it arises naturally from a particular formulation of the problem (e.g., in a probabilistic formulation the posterior probability of the model can be used as an inverse cost). Ultimately, the cost function will depend on the desired task. An overview of the three main categories of learning tasks is provided below.

**Learning paradigms**

There are three major learning paradigms, each corresponding to a particular abstract learning task. These are supervised learning, unsupervised learning and reinforcement learning.

**Supervised learning**

In supervised learning, we are given a set of example pairs \((x,y)\), \(x \in X, y \in Y\) and the aim is to find a function \( f: X \rightarrow Y \) in the allowed class of functions that matches the examples. In other words, we wish to infer the mapping implied by the data; the cost function is related to the mismatch between our mapping and the data and it implicitly contains prior knowledge about the problem domain.
A commonly used cost is the mean-squared error, which tries to minimize the average squared error between the network's output, \( f(x) \), and the target value \( y \) over all the example pairs. When one tries to minimize this cost using gradient descent for the class of neural networks called multilayer perceptrons, one obtains the common and well-known backpropagation algorithm for training neural networks.

Tasks that fall within the paradigm of supervised learning are pattern recognition (also known as classification) and regression (also known as function approximation). The supervised learning paradigm is also applicable to sequential data (e.g., for speech and gesture recognition). This can be thought of as learning with a "teacher," in the form of a function that provides continuous feedback on the quality of solutions obtained thus far.

Unsupervised learning

In unsupervised learning, some data \( x \) is given and the cost function to be minimized, that can be any function of the data \( x \) and the network's output, \( f \).

The cost function is dependent on the task (what we are trying to model) and our \textit{a priori} assumptions (the implicit properties of our model, its parameters and the observed variables).

As a trivial example, consider the model \( f(x) = a \), where \( a \) is a constant and the cost \( C = E[(x - f(x))^2] \). Minimizing this cost will give us a value of \( a \) that is equal to the mean of the data. The cost function can be much more complicated. Its form depends on the application: for example, in compression it could be related to the mutual information between \( x \) and \( f(x) \), whereas in statistical modeling, it could be related to the posterior probability of the model given the data. (Note that in both of those examples those quantities would be maximized rather than minimized).

Tasks that fall within the paradigm of unsupervised learning are in general estimation problems; the applications include clustering, the estimation of statistical distributions, compression and filtering.

Reinforcement learning

In reinforcement learning, data \( x \) are usually not given, but generated by an agent's interactions with the environment. At each point in time \( t \), the agent performs an action \( y_t \) and the environment generates an observation \( x_t \) and an instantaneous cost \( c_t \), according to some (usually unknown) dynamics. The aim is to discover a \textit{policy} for selecting actions that minimizes some measure of a long-term cost; i.e., the expected cumulative cost. The environment's dynamics and the long-term cost for each policy are usually unknown, but can be estimated.

More formally, the environment is modeled as a Markov decision process (MDP) with states \( s_1, \ldots, s_m \in S \) and actions \( a_1, \ldots, a_n \in A \) with the following probability distributions: the instantaneous cost distribution \( P(c_t|s_t) \), the observation distribution \( P(x_t|s_t) \) and the transition \( P(s_{t+1}|s_t, a_t) \), while a policy is defined as conditional distribution over actions given the observations. Taken together, the two define a Markov chain (MC). The aim is to discover the policy that minimizes the cost; i.e., the MC for which the cost is minimal.

ANNs are frequently used in reinforcement learning as part of the overall algorithm. Dynamic programming has been coupled with ANNs (Neuro dynamic programming) by Bertsekas and Tsitsiklis \cite{1} and applied to multi-dimensional nonlinear problems such as those involved in vehicle routing or natural resources management because of the ability of ANNs to mitigate losses of accuracy even when reducing the discretization grid density for numerically approximating the solution of the original control problems.

Tasks that fall within the paradigm of reinforcement learning are control problems, games and other sequential decision making tasks.
Learning algorithms

Training a neural network model essentially means selecting one model from the set of allowed models (or, in a Bayesian framework, determining a distribution over the set of allowed models) that minimizes the cost criterion. There are numerous algorithms available for training neural network models; most of them can be viewed as a straightforward application of optimization theory and statistical estimation.

Most of the algorithms used in training artificial neural networks employ some form of gradient descent. This is done by simply taking the derivative of the cost function with respect to the network parameters and then changing those parameters in a gradient-related direction.


Employing artificial neural networks

Perhaps the greatest advantage of ANNs is their ability to be used as an arbitrary function approximation mechanism that 'learns' from observed data. However, using them is not so straightforward and a relatively good understanding of the underlying theory is essential.

- Choice of model: This will depend on the data representation and the application. Overly complex models tend to lead to problems with learning.
- Learning algorithm: There are numerous trade-offs between learning algorithms. Almost any algorithm will work well with the correct hyperparameters for training on a particular fixed data set. However selecting and tuning an algorithm for training on unseen data requires a significant amount of experimentation.
- Robustness: If the model, cost function and learning algorithm are selected appropriately the resulting ANN can be extremely robust.

With the correct implementation, ANNs can be used naturally in online learning and large data set applications. Their simple implementation and the existence of mostly local dependencies exhibited in the structure allows for fast, parallel implementations in hardware.

Applications

The utility of artificial neural network models lies in the fact that they can be used to infer a function from observations. This is particularly useful in applications where the complexity of the data or task makes the design of such a function by hand impractical.

Real-life applications

The tasks artificial neural networks are applied to tend to fall within the following broad categories:

- Function approximation, or regression analysis, including time series prediction, fitness approximation and modeling.
- Classification, including pattern and sequence recognition, novelty detection and sequential decision making.
- Data processing, including filtering, clustering, blind source separation and compression.
- Robotics, including directing manipulators, Computer numerical control.

Application areas include system identification and control (vehicle control, process control, natural resources management), quantum chemistry,[6] game-playing and decision making (backgammon, chess, poker), pattern recognition (radar systems, face identification, object recognition and more), sequence recognition (gesture, speech, handwritten text recognition), medical diagnosis, financial applications (automated trading systems), data mining (or knowledge discovery in databases, "KDD"), visualization and e-mail spam filtering.
Artificial neural networks have also been used to diagnose several cancers. An ANN based hybrid lung cancer detection system named HLND improves the accuracy of diagnosis and the speed of lung cancer radiology. These networks have also been used to diagnose prostate cancer. The diagnoses can be used to make specific models taken from a large group of patients compared to information of one given patient. The models do not depend on assumptions about correlations of different variables. Colorectal cancer has also been predicted using the neural networks. Neural networks could predict the outcome for a patient with colorectal cancer with a lot more accuracy than the current clinical methods. After training, the networks could predict multiple patient outcomes from unrelated institutions.

**Neural networks and neuroscience**

Theoretical and computational neuroscience is the field concerned with the theoretical analysis and computational modeling of biological neural systems. Since neural systems are intimately related to cognitive processes and behavior, the field is closely related to cognitive and behavioral modeling.

The aim of the field is to create models of biological neural systems in order to understand how biological systems work. To gain this understanding, neuroscientists strive to make a link between observed biological processes (data), biologically plausible mechanisms for neural processing and learning (biological neural network models) and theory (statistical learning theory and information theory).

**Types of models**

Many models are used in the field defined at different levels of abstraction and modeling different aspects of neural systems. They range from models of the short-term behavior of individual neurons, models of how the dynamics of neural circuitry arise from interactions between individual neurons and finally to models of how behavior can arise from abstract neural modules that represent complete subsystems. These include models of the long-term, and short-term plasticity, of neural systems and their relations to learning and memory from the individual neuron to the system level.

**Current research**

While initial research had been concerned mostly with the electrical characteristics of neurons, a particularly important part of the investigation in recent years has been the exploration of the role of neuromodulators such as dopamine, acetylcholine, and serotonin on behavior and learning.

Biophysical models, such as BCM theory, have been important in understanding mechanisms for synaptic plasticity, and have had applications in both computer science and neuroscience. Research is ongoing in understanding the computational algorithms used in the brain, with some recent biological evidence for radial basis networks and neural backpropagation as mechanisms for processing data.

Computational devices have been created in CMOS for both biophysical simulation and neuromorphic computing. More recent efforts show promise for creating nanodevices for very large scale principal components analyses and convolution. If successful, these effort could usher in a new era of neural computing that is a step beyond digital computing, because it depends on learning rather than programming and because it is fundamentally analog rather than digital even though the first instantiations may in fact be with CMOS digital devices.
Neural network software

Neural network software is used to simulate, research, develop and apply artificial neural networks, biological neural networks and in some cases a wider array of adaptive systems.

Types of artificial neural networks

Artificial neural network types vary from those with only one or two layers of single direction logic, to complicated multi-input many directional feedback loop and layers. On the whole, these systems use algorithms in their programming to determine control and organization of their functions. Some may be as simple as a one neuron layer with an input and an output, and others can mimic complex systems such as dANN, which can mimic chromosomal DNA through sizes at cellular level, into artificial organisms and simulate reproduction, mutation and population sizes.[9]

Most systems use "weights" to change the parameters of the throughput and the varying connections to the neurons. Artificial neural networks can be autonomous and learn by input from outside "teachers" or even self-teaching from written in rules.

Theoretical properties

Computational power

The multi-layer perceptron (MLP) is a universal function approximator, as proven by the Cybenko theorem. However, the proof is not constructive regarding the number of neurons required or the settings of the weights.

Work by Hava Siegelmann and Eduardo D. Sontag has provided a proof that a specific recurrent architecture with rational valued weights (as opposed to full precision real number-valued weights) has the full power of a Universal Turing Machine[10] using a finite number of neurons and standard linear connections. They have further shown that the use of irrational values for weights results in a machine with super-Turing power.

Capacity

Artificial neural network models have a property called 'capacity', which roughly corresponds to their ability to model any given function. It is related to the amount of information that can be stored in the network and to the notion of complexity.

Convergence

Nothing can be said in general about convergence since it depends on a number of factors. Firstly, there may exist many local minima. This depends on the cost function and the model. Secondly, the optimization method used might not be guaranteed to converge when far away from a local minimum. Thirdly, for a very large amount of data or parameters, some methods become impractical. In general, it has been found that theoretical guarantees regarding convergence are an unreliable guide to practical application.

Generalization and statistics

In applications where the goal is to create a system that generalizes well in unseen examples, the problem of over-training has emerged. This arises in convoluted or over-specified systems when the capacity of the network significantly exceeds the needed free parameters. There are two schools of thought for avoiding this problem: The first is to use cross-validation and similar techniques to check for the presence of overtraining and optimally select hyperparameters such as to minimize the generalization error. The second is to use some form of regularization. This is a concept that emerges naturally in a probabilistic (Bayesian) framework, where the regularization can be performed by selecting a larger prior probability over simpler models; but also in statistical learning theory, where
the goal is to minimize over two quantities: the 'empirical risk' and the 'structural risk', which roughly corresponds to the error over the training set and the predicted error in unseen data due to overfitting.

Supervised neural networks that use an MSE cost function can use formal statistical methods to determine the confidence of the trained model. The MSE on a validation set can be used as an estimate for variance. This value can then be used to calculate the confidence interval of the output of the network, assuming a normal distribution. A confidence analysis made this way is statistically valid as long as the output probability distribution stays the same and the network is not modified.

By assigning a softmax activation function on the output layer of the neural network (or a softmax component in a component-based neural network) for categorical target variables, the outputs can be interpreted as posterior probabilities. This is very useful in classification as it gives a certainty measure on classifications.

The softmax activation function is:

\[ y_i = \frac{e^{x_i}}{\sum_{j=1}^{c} e^{x_j}} \]

**Dynamic properties**

Various techniques originally developed for studying disordered magnetic systems (i.e., the spin glass) have been successfully applied to simple neural network architectures, such as the Hopfield network. Influential work by E. Gardner and B. Derrida has revealed many interesting properties about perceptrons with real-valued synaptic weights, while later work by W. Krauth and M. Mezard has extended these principles to binary-valued synapses.

**Disadvantages**

One drawback to using artificial neural networks, particularly in robotics, is that they require a large diversity of training for real-world operation. Dean Pomerleau, in his research presented in the paper "Knowledge-based Training of Artificial Neural Networks for Autonomous Robot Driving," uses a neural network to train a robotic vehicle to drive on multiple types of roads (single lane, multi-lane, dirt, etc.). A large amount of his research is devoted to (1) extrapolating multiple training scenarios from a single training experience, and (2) preserving past training diversity so that the system does not become overtrained (if, for example, it is presented with a series of right turns – it should not learn to always turn right). These issues are common in neural networks that must decide from amongst a wide variety of responses.

A. K. Dewdney, a former *Scientific American* columnist, wrote in 1997, "Although neural nets do solve a few toy problems, their powers of computation are so limited that I am surprised anyone takes them seriously as a general problem-solving tool." (Dewdney, p. 82)

Arguments for Dewdney’s position are that to implement large and effective software neural networks, much processing and storage resources need to be committed. While the brain has hardware tailored to the task of processing signals through a graph of neurons, simulating even a most simplified form on Von Neumann technology may compel a NN designer to fill many millions of database rows for its connections - which can lead to excessive RAM and HD necessities. Furthermore, the designer of NN systems will often need to simulate the transmission of signals through many of these connections and their associated neurons - which must often be matched with incredible amounts of CPU processing power and time. While neural networks often yield effective programs, they too often do so at the cost of time and monetary efficiency.
Arguments against Dewdney’s position are that neural nets have been successfully used to solve many complex and diverse tasks, ranging from autonomously flying aircraft\[11\] to detecting credit card fraud.\[12\] Technology writer Roger Bridgman commented on Dewdney’s statements about neural nets:

Neural networks, for instance, are in the dock not only because they have been hyped to high heaven, (what hasn’t?) but also because you could create a successful net without understanding how it worked: the bunch of numbers that captures its behaviour would in all probability be “an opaque, unreadable table...valueless as a scientific resource”. In spite of his emphatic declaration that science is not technology, Dewdney seems here to pillory neural nets as bad science when most of those devising them are just trying to be good engineers. An unreadable table that a useful machine could read would still be well worth having.\[13\]

Some other criticisms came from believers of hybrid models (combining neural networks and symbolic approaches). They advocate the intermix of these two approaches and believe that hybrid models can better capture the mechanisms of the human mind (Sun and Bookman 1994).

**Gallery**

A single-layer feedforward artificial neural network. Arrows originating from \(x\) are omitted for clarity. There are \(p\) inputs to this network and \(q\) outputs. There is no activation function (or equivalently, the activation function is \(g(x) = x\)). In this system, the value of the \(q\)th output, \(y_q\), would be calculated as \(y_q = \sum (x_i \cdot w_{iq})\).

A two-layer feedforward artificial neural network.

**References**


Artificial neural network

References

Artificial neural network


**Further reading**

• Dedicated issue of *Philosophical Transactions B* on Neural Networks and Perception. Some articles are freely available. (http://publishing.royalsociety.org/neural-networks)

**External links**

• Neural Networks (http://www.dmoz.org/Computers/Artificial_Intelligence/Neural_Networks/) at the Open Directory Project
• A close view to Artificial Neural Networks Algorithms (http://www.learnartificialneuralnetworks.com)
• A Brief Introduction to Neural Networks (D. Kriesel) (http://www.dkriesel.com/en/science/neural_networks)
  - Illustrated, bilingual manuscript about artificial neural networks; Topics so far: Perceptrons, Backpropagation, Radial Basis Functions, Recurrent Neural Networks, Self Organizing Maps, Hopfield Networks.
• Neural Networks in Materials Science (http://www.msm.cam.ac.uk/phase-trans/abstracts/neural.review.html)
• A practical tutorial on Neural Networks (http://www.ai-junkie.com/ann/evolved/nt1.html)
Types of artificial neural networks

There are many types of artificial neural networks (ANN). An artificial neural network is a computational simulation of a biological neural network. These models mimic the real life behaviour of neurons and the electrical messages they produce between input (such as from the eyes or nerve endings in the hand), processing by the brain and the final output from the brain (such as reacting to light or from sensing touch or heat). There are other ANNs which are adaptive systems used to model things such as environments and population.

The systems can be hardware and software based specifically built systems or purely software based and run in computer models.

Feedforward neural network

The feedforward neural network was the first and arguably most simple type of artificial neural network devised. In this network the information moves in only one direction — forwards: From the input nodes data goes through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network. Feedforward networks can be constructed from different types of units, e.g. binary McCulloch-Pitts neurons, the simplest example being the perceptron. Continuous neurons, frequently with sigmoidal activation, are used in the context of backpropagation of error.

Radial basis function (RBF) network

Radial basis functions are powerful techniques for interpolation in multidimensional space. A RBF is a function which has built into a distance criterion with respect to a center. Radial basis functions have been applied in the area of neural networks where they may be used as a replacement for the sigmoidal hidden layer transfer characteristic in multi-layer perceptrons. RBF networks have two layers of processing: In the first, input is mapped onto each RBF in the 'hidden' layer. The RBF chosen is usually a Gaussian. In regression problems the output layer is then a linear combination of hidden layer values representing mean predicted output. The interpretation of this output layer value is the same as a regression model in statistics. In classification problems the output layer is typically a sigmoid function of a linear combination of hidden layer values, representing a posterior probability. Performance in both cases is often improved by shrinkage techniques, known as ridge regression in classical statistics and known to correspond to a prior belief in small parameter values (and therefore smooth output functions) in a Bayesian framework.

RBF networks have the advantage of not suffering from local minima in the same way as Multi-Layer Perceptrons. This is because the only parameters that are adjusted in the learning process are the linear mapping from hidden layer to output layer. Linearity ensures that the error surface is quadratic and therefore has a single easily found minimum. In regression problems this can be found in one matrix operation. In classification problems the fixed non-linearity introduced by the sigmoid output function is most efficiently dealt with using iteratively re-weighted least squares.

RBF networks have the disadvantage of requiring good coverage of the input space by radial basis functions. RBF centres are determined with reference to the distribution of the input data, but without reference to the prediction task. As a result, representational resources may be wasted on areas of the input space that are irrelevant to the learning task. A common solution is to associate each data point with its own centre, although this can make the linear system to be solved in the final layer rather large, and requires shrinkage techniques to avoid overfitting.

Associating each input datum with an RBF leads naturally to kernel methods such as support vector machines and Gaussian processes (the RBF is the kernel function). All three approaches use a non-linear kernel function to project the input data into a space where the learning problem can be solved using a linear model. Like Gaussian Processes, and unlike SVMs, RBF networks are typically trained in a Maximum Likelihood framework by maximizing the probability (minimizing the error) of the data under the model. SVMs take a different approach to avoiding
overfitting by maximizing instead a margin. RBF networks are outperformed in most classification applications by SVMs. In regression applications they can be competitive when the dimensionality of the input space is relatively small.

**Kohonen self-organizing network**

The self-organizing map (SOM) invented by Teuvo Kohonen performs a form of unsupervised learning. A set of artificial neurons learn to map points in an input space to coordinates in an output space. The input space can have different dimensions and topology from the output space, and the SOM will attempt to preserve these.

**Learning Vector Quantization**

Learning Vector Quantization (LVQ) can also be interpreted as a neural network architecture. It was suggested by Teuvo Kohonen, originally. In LVQ, prototypical representatives of the classes parameterize, together with an appropriate distance measure, a distance-based classification scheme.

**Recurrent neural network**

Contrary to feedforward networks recurrent neural networks (RNNs) are models with bi-directional data flow. While a feedforward network propagates data linearly from input to output, RNNs also propagate data from later processing stages to earlier stages. RNNs can be used as general sequence processors.

**Fully recurrent network**

This is the basic architecture developed in the 1980s: a network of neuron-like units, each with a directed connection to every other unit. Each unit has a time-varying real-valued activation. Each connection has a modifiable real-valued weight. Some of the nodes are called input nodes, some output nodes, the rest hidden nodes. Most architectures below are special cases.

For supervised learning in discrete time settings, training sequences of real-valued input vectors become sequences of activations of the input nodes, one input vector at a time. At any given time step, each non-input unit computes its current activation as a nonlinear function of the weighted sum of the activations of all units from which it receives connections. There may be teacher-given target activations for some of the output units at certain time steps. For example, if the input sequence is a speech signal corresponding to a spoken digit, the final target output at the end of the sequence may be a label classifying the digit. For each sequence, its error is the sum of the deviations of all target signals from the corresponding activations computed by the network. For a training set of numerous sequences, the total error is the sum of the errors of all individual sequences.

To minimize total error, gradient descent can be used to change each weight in proportion to its derivative with respect to the error, provided the non-linear activation functions are differentiable. Various methods for doing so were developed in the 1980s and early 1990s by Paul Werbos, Ronald J. Williams, Tony Robinson, Jürgen Schmidhuber, Barak Pearlmutter, and others. The standard method is called "backpropagation through time" or BPTT, a generalization of back-propagation for feedforward networks. A more computationally expensive online variant is called "Real-Time Recurrent Learning" or RTRL. Unlike BPTT this algorithm is local in time but not local in space. There also is an online hybrid between BPTT and RTRL with intermediate complexity, and there are variants for continuous time. A major problem with gradient descent for standard RNN architectures is that error gradients vanish exponentially quickly with the size of the time lag between important events, as first realized by Sepp Hochreiter in 1991. The Long short term memory architecture overcomes these problems.

In reinforcement learning settings, there is no teacher providing target signals for the RNN, instead a fitness function or reward function or utility function is occasionally used to evaluate the performance of the RNN, which is
influencing its input stream through output units connected to actuators affecting the environment. Variants of evolutionary computation are often used to optimize the weight matrix.

**Hopfield network**

The Hopfield network (like similar attractor-based networks) is of historic interest although it is not a general RNN, as it is not designed to process sequences of patterns. Instead it requires stationary inputs. It is an RNN in which all connections are symmetric. Invented by John Hopfield in 1982 it guarantees that its dynamics will converge. If the connections are trained using Hebbian learning then the Hopfield network can perform as robust content-addressable memory, resistant to connection alteration.

**Boltzmann machine**

The Boltzmann machine can be thought of as a noisy Hopfield network. Invented by Geoff Hinton and Terry Sejnowski in 1985, the Boltzmann machine is important because it is one of the first neural networks to demonstrate learning of latent variables (hidden units). Boltzmann machine learning was at first slow to simulate, but the contrastive divergence algorithm of Geoff Hinton (circa 2000) allows models such as Boltzmann machines and Products of Experts to be trained much faster.

**Simple recurrent networks**

This special case of the basic architecture above was employed by Jeff Elman and Michael I. Jordan. A three-layer network is used, with the addition of a set of "context units" in the input layer. There are connections from the hidden layer (Elman) or from the output layer (Jordan) to these context units fixed with a weight of one. At each time step, the input is propagated in a standard feedforward fashion, and then a simple backprop-like learning rule is applied (this rule is not performing proper gradient descent, however). The fixed back connections result in the context units always maintaining a copy of the previous values of the hidden units (since they propagate over the connections before the learning rule is applied).

**Echo state network**

The echo state network (ESN) is a recurrent neural network with a sparsely connected random hidden layer. The weights of output neurons are the only part of the network that can change and be trained. ESN are good at reproducing certain time series. A variant for spiking neurons is known as Liquid state machines.

**Long short term memory network**

The Long short term memory (LSTM), developed by Hochreiter and Schmidhuber in 1997, is an artificial neural net structure that unlike traditional RNNs doesn't have the problem of vanishing gradients. It works even when there are long delays, and it can handle signals that have a mix of low and high frequency components. LSTM RNN outperformed other RNN and other sequence learning methods methods such as HMM in numerous applications such as language learning and connected handwriting recognition.

**Bi-directional RNN**

Invented by Schuster & Paliwal in 1997 bi-directional RNNs, or BRNNs, use a finite sequence to predict or label each element of the sequence based on both the past and the future context of the element. This is done by adding the outputs of two RNNs: one processing the sequence from left to right, the other one from right to left. The combined outputs are the predictions of the teacher-given target signals. This technique proved to be especially useful when combined with LSTM RNNs.
Hierarchical RNN

There are many instances of hierarchical RNN whose elements are connected in various ways to decompose hierarchical behavior into useful subprograms.\textsuperscript{[20][21]}

Stochastic neural networks

A stochastic neural network differs from a typical neural network because it introduces random variations into the network. In a probabilistic view of neural networks, such random variations can be viewed as a form of statistical sampling, such as Monte Carlo sampling.

Modular neural networks

Biological studies have shown that the human brain functions not as a single massive network, but as a collection of small networks. This realization gave birth to the concept of modular neural networks, in which several small networks cooperate or compete to solve problems.

Committee of machines

A committee of machines (CoM) is a collection of different neural networks that together “vote” on a given example. This generally gives a much better result compared to other neural network models. Because neural networks suffer from local minima, starting with the same architecture and training but using different initial random weights often gives vastly different networks. A CoM tends to stabilize the result.

The CoM is similar to the general machine learning bagging method, except that the necessary variety of machines in the committee is obtained by training from different random starting weights rather than training on different randomly selected subsets of the training data.

Associative neural network (ASNN)

The ASNN is an extension of the committee of machines that goes beyond a simple/weighted average of different models. ASNN\textsuperscript{[22]} represents a combination of an ensemble of feedforward neural networks and the k-nearest neighbor technique (kNN). It uses the correlation between ensemble responses as a measure of distance amid the analyzed cases for the kNN. This corrects the bias of the neural network ensemble. An associative neural network has a memory that can coincide with the training set. If new data become available, the network instantly improves its predictive ability and provides data approximation (self-learn the data) without a need to retrain the ensemble. Another important feature of ASNN is the possibility to interpret neural network results by analysis of correlations between data cases in the space of models. The method is demonstrated at www.vcclab.org\textsuperscript{[23]}, where it can be used online or downloaded.


**Physical neural network**

A physical neural network includes electrically adjustable resistance material to simulate artificial synapses. Examples include the ADALINE neural network developed by Bernard Widrow in the 1960s and the memristor based neural network developed by Greg Snider of HP Labs in 2008.

**Other types of networks**

These special networks do not fit in any of the previous categories.

**Holographic associative memory**

*Holographic associative memory* represents a family of analog, correlation-based, associative, stimulus-response memories, where information is mapped onto the phase orientation of complex numbers operating.

**Instantaneously trained networks**

*Instantaneously trained neural networks* (ITNNs) were inspired by the phenomenon of short-term learning that seems to occur instantaneously. In these networks the weights of the hidden and the output layers are mapped directly from the training vector data. Ordinarily, they work on binary data, but versions for continuous data that require small additional processing are also available.

**Spiking neural networks**

Spiking neural networks (SNNs) are models which explicitly take into account the timing of inputs. The network input and output are usually represented as series of spikes (delta function or more complex shapes). SNNs have an advantage of being able to process information in the time domain (signals that vary over time). They are often implemented as recurrent networks. SNNs are also a form of pulse computer.

Spiking neural networks with axonal conduction delays exhibit polychronization, and hence could have a very large memory capacity.[24]

Networks of spiking neurons — and the temporal correlations of neural assemblies in such networks — have been used to model figure/ground separation and region linking in the visual system (see, for example, Reitboeck et al.in Haken and Stadler: Synergetics of the Brain. Berlin, 1989).

In June 2005 IBM announced construction of a Blue Gene supercomputer dedicated to the simulation of a large recurrent spiking neural network.[25]

Gerstner and Kistler have a freely available online textbook on Spiking Neuron Models.[26]

**Dynamic neural networks**

Dynamic neural networks not only deal with nonlinear multivariate behaviour, but also include (learning of) time-dependent behaviour such as various transient phenomena and delay effects. Techniques to estimate a system process from observed data fall under the general category of system identification.

**Cascading neural networks**

Cascade Correlation is an architecture and supervised learning algorithm developed by Scott Fahlman and Christian Lebiere. Instead of just adjusting the weights in a network of fixed topology, Cascade-Correlation begins with a minimal network, then automatically trains and adds new hidden units one by one, creating a multi-layer structure. Once a new hidden unit has been added to the network, its input-side weights are frozen. This unit then becomes a permanent feature-detector in the network, available for producing outputs or for creating other, more complex feature detectors. The Cascade-Correlation architecture has several advantages over existing algorithms: it learns
very quickly, the network determines its own size and topology, it retains the structures it has built even if the training set changes, and it requires no back-propagation of error signals through the connections of the network.

**Neuro-fuzzy networks**

A neuro-fuzzy network is a fuzzy inference system in the body of an artificial neural network. Depending on the FIS type, there are several layers that simulate the processes involved in a fuzzy inference like fuzzification, inference, aggregation and defuzzification. Embedding an FIS in a general structure of an ANN has the benefit of using available ANN training methods to find the parameters of a fuzzy system.

**Compositional pattern-producing networks**

Compositional pattern-producing networks (CPPNs) are a variation of ANNs which differ in their set of activation functions and how they are applied. While typical ANNs often contain only sigmoid functions (and sometimes Gaussian functions), CPPNs can include both types of functions and many others. Furthermore, unlike typical ANNs, CPPNs are applied across the entire space of possible inputs so that they can represent a complete image. Since they are compositions of functions, CPPNs in effect encode images at infinite resolution and can be sampled for a particular display at whatever resolution is optimal.

**One-shot associative memory**

This type of network can add new patterns without the need for re-training. It is done by creating a specific memory structure, which assigns each new pattern to an orthogonal plane using adjacently connected hierarchical arrays [27]. The network offers real-time pattern recognition and high scalability, it however requires parallel processing and is thus best suited for platforms such as Wireless sensor networks (WSN), Grid computing, and GPGPUs.

**References**

Feedforward neural network

A **feedforward neural network** is an artificial neural network where connections between the units do *not* form a directed cycle. This is different from recurrent neural networks.

The feedforward neural network was the first and arguably simplest type of artificial neural network devised. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.

**Single-layer perceptron**

The simplest kind of neural network is a *single-layer perceptron* network, which consists of a single layer of output nodes; the inputs are fed directly to the outputs via a series of weights. In this way it can be considered the simplest kind of feed-forward network. The sum of the products of the weights and the inputs is calculated in each node, and if the value is above some threshold (typically 0) the neuron fires and takes the activated value (typically 1); otherwise it takes the deactivated value (typically -1). Neurons with this kind of activation function are also called *Artificial neurons* or *linear threshold units*. In the literature the term *perceptron* often refers to networks consisting of just one of these units. A similar neuron was described by Warren McCulloch and Walter Pitts in the 1940s.

A perceptron can be created using any values for the activated and deactivated states as long as the threshold value lies between the two. Most perceptrons have outputs of 1 or -1 with a threshold of 0 and there is some evidence that
such networks can be trained more quickly than networks created from nodes with different activation and deactivation values.

Perceptrons can be trained by a simple learning algorithm that is usually called the delta rule. It calculates the errors between calculated output and sample output data, and uses this to create an adjustment to the weights, thus implementing a form of gradient descent.

Single-unit perceptrons are only capable of learning linearly separable patterns; in 1969 in a famous monograph entitled Perceptrons Marvin Minsky and Seymour Papert showed that it was impossible for a single-layer perceptron network to learn an XOR function. It is often believed that they also conjectured (incorrectly) that a similar result would hold for a multi-layer perceptron network. However, this is not true, as both Minsky and Papert already knew that multi-layer perceptrons were capable of producing an XOR Function. (See the page on Perceptrons for more information.)

Although a single threshold unit is quite limited in its computational power, it has been shown that networks of parallel threshold units can approximate any continuous function from a compact interval of the real numbers into the interval [-1,1]. This very recent result can be found in Peter Auer, Harald Burgsteiner and Wolfgang Maass "A learning rule for very simple universal approximators consisting of a single layer of perceptrons". [1]

A multi-layer neural network can compute a continuous output instead of a step function. A common choice is the so-called logistic function:

\[ y = \frac{1}{1 + e^{-z}} \]

(In general form, f(X) is in place of x, where f(X) is an analytic function in set of x's.) With this choice, the single-layer network is identical to the logistic regression model, widely used in statistical modeling. The logistic function is also known as the sigmoid function. It has a continuous derivative, which allows it to be used in backpropagation. This function is also preferred because its derivative is easily calculated:

\[ y' = y(1 - y)(\text{times } \frac{df}{dX}) \]

, in general form, according to the Chain Rule)
Multi-layer perceptron

This class of networks consists of multiple layers of computational units, usually interconnected in a feed-forward way. Each neuron in one layer has directed connections to the neurons of the subsequent layer. In many applications the units of these networks apply a sigmoid function as an activation function.

The universal approximation theorem for neural networks states that every continuous function that maps intervals of real numbers to some output interval of real numbers can be approximated arbitrarily closely by a multi-layer perceptron with just one hidden layer. This result holds only for restricted classes of activation functions, e.g. for the sigmoidal functions.

Multi-layer networks use a variety of learning techniques, the most popular being back-propagation. Here, the output values are compared with the correct answer to compute the value of some predefined error-function. By various techniques, the error is then fed back through the network. Using this information, the algorithm adjusts the weights of each connection in order to reduce the value of the error function by some small amount. After repeating this process for a sufficiently large number of training cycles, the network will usually converge to some state where the error of the calculations is small. In this case, one would say that the network has learned a certain target function. To adjust weights properly, one applies a general method for non-linear optimization that is called gradient descent. For this, the derivative of the error function with respect to the network weights is calculated, and the weights are then changed such that the error decreases (thus going downhill on the surface of the error function). For this reason, back-propagation can only be applied on networks with differentiable activation functions.

In general, the problem of teaching a network to perform well, even on samples that were not used as training samples, is a quite subtle issue that requires additional techniques. This is especially important for cases where only very limited numbers of training samples are available. The danger is that the network overfits the training data and fails to capture the true statistical process generating the data. Computational learning theory is concerned with training classifiers on a limited amount of data. In the context of neural networks a simple heuristic, called early stopping, often ensures that the network will generalize well to examples not in the training set.

Other typical problems of the back-propagation algorithm are the speed of convergence and the possibility of ending up in a local minimum of the error function. Today there are practical solutions that make back-propagation in multi-layer perceptrons the solution of choice for many machine learning tasks.
ADALINE

ADALINE stands for Adaptive Linear Element. It was developed by Professor Bernard Widrow and his graduate student Ted Hoff at Stanford University in 1960. It is based on the McCulloch-Pitts model and consists of a weight, a bias and a summation function.

Operation: \( y_i = w x_i + b \)

Its adaptation is defined through a cost function (error metric) of the residual \( e = d_i - (b + w x_i) \) where \( d_i \) is the desired output. With the MSE error metric \( E = \frac{1}{2N} \sum_{i=1}^{N} e_i^2 \) the adapted weight and bias become:

\[
\begin{align*}
b &= \frac{\sum_i x_i^2 \sum_i d_i - \sum_i x_i \sum_i x_i d_i}{N(\sum_i (x_i - \bar{x})^2)} \\
w &= \frac{\sum_i (x_i - \bar{x})(d_i - \bar{d})}{\sum_i (x_i - \bar{x})^2}
\end{align*}
\]

The Adaline has practical applications in the controls area. A single neuron with tap delayed inputs (the number of inputs is bounded by the lowest frequency present and the Nyquist rate) can be used to determine the higher order transfer function of a physical system via the bi-linear z-transform. This is done as the Adaline is, functionally, an adaptive FIR filter. Like the single-layer perceptron, ADALINE has a counterpart in statistical modelling, in this case least squares regression.

There is an extension of the Adaline, called the Multiple Adaline (MADALINE) that consists of two or more adalines serially connected.

References


External links

• Feedforward neural networks tutorial (http://www.emilstefanov.net/Projects/NeuralNetworks.aspx)
• Feedforward Neural Network: Example (http://wiki.synleus.com/index.php/DANN:Backprop_Feedforward_Neural_Network)
• Feedforward Neural Networks: An Introduction (http://media.wiley.com/product_data/excerpt/19/04713491/0471349119.pdf)
A self-organizing map (SOM) or self-organizing feature map (SOFM) is a type of artificial neural network that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional), discretized representation of the input space of the training samples, called a map. Self-organizing maps are different from other artificial neural networks in the sense that they use a neighborhood function to preserve the topological properties of the input space.

This makes SOMs useful for visualizing low-dimensional views of high-dimensional data, akin to multidimensional scaling. The model was first described as an artificial neural network by the Finnish professor Teuvo Kohonen, and is sometimes called a Kohonen map.[1]

Like most artificial neural networks, SOMs operate in two modes: training and mapping. Training builds the map using input examples. It is a competitive process, also called vector quantization. Mapping automatically classifies a new input vector.

A self-organizing map consists of components called nodes or neurons. Associated with each node is a weight vector of the same dimension as the input data vectors and a position in the map space. The usual arrangement of nodes is a regular spacing in a hexagonal or rectangular grid. The self-organizing map describes a mapping from a higher dimensional input space to a lower dimensional map space. The procedure for placing a vector from data space onto the map is to first find the node with the closest weight vector to the vector taken from data space. Once the closest node is located it is assigned the values from the vector taken from the data space.

While it is typical to consider this type of network structure as related to feedforward networks where the nodes are visualized as being attached, this type of architecture is fundamentally different in arrangement and motivation.

Useful extensions include using toroidal grids where opposite edges are connected and using large numbers of nodes. It has been shown that while self-organizing maps with a small number of nodes behave in a way that is similar to K-means, larger self-organizing maps rearrange data in a way that is fundamentally topological in character.

It is also common to use the U-Matrix. The U-Matrix value of a particular node is the average distance between the node and its closest neighbors (ref. 9). In a square grid for instance, we might consider the closest 4 or 8 nodes (the Von Neumann neighborhood and Moore neighborhood respectively), or six nodes in a hexagonal grid.

Large SOMs display properties which are emergent. In maps consisting of thousands of nodes, it is possible to perform cluster operations on the map itself.[2]
Self-organizing map

Learning algorithm

The goal of learning in the self-organizing map is to cause different parts of the network to respond similarly to certain input patterns. This is partly motivated by how visual, auditory or other sensory information is handled in separate parts of the cerebral cortex in the human brain.[3]

The weights of the neurons are initialized either to small random values or sampled evenly from the subspace spanned by the two largest principal component eigenvectors. With the latter alternative, learning is much faster because the initial weights already give good approximation of SOM weights.[4]

The network must be fed a large number of example vectors that represent, as close as possible, the kinds of vectors expected during mapping. The examples are usually administered several times as iterations.

The training utilizes competitive learning. When a training example is fed to the network, its Euclidean distance to all weight vectors is computed. The neuron with weight vector most similar to the input is called the best matching unit (BMU). The weights of the BMU and neurons close to it in the SOM lattice are adjusted towards the input vector. The magnitude of the change decreases with time and with distance from the BMU. The update formula for a neuron with weight vector $W_v(t)$ is

$$W_v(t + 1) = W_v(t) + \Theta(v, t) \alpha(t)(D(t) - W_v(t)),$$

where $\alpha(t)$ is a monotonically decreasing learning coefficient and $D(t)$ is the input vector. The neighborhood function $\Theta(v, t)$ depends on the lattice distance between the BMU and neuron $v$. In the simplest form it is one for all neurons close enough to BMU and zero for others, but a Gaussian function is a common choice, too. Regardless of the functional form, the neighborhood function shrinks with time.[3] At the beginning when the neighborhood is broad, the self-organizing takes place on the global scale. When the neighborhood has shrunk to just a couple of neurons the weights are converging to local estimates.

This process is repeated for each input vector for a (usually large) number of cycles $\lambda$. The network winds up associating output nodes with groups or patterns in the input data set. If these patterns can be named, the names can be attached to the associated nodes in the trained net.

During mapping, there will be one single winning neuron: the neuron whose weight vector lies closest to the input vector. This can be simply determined by calculating the Euclidean distance between input vector and weight vector.

While representing input data as vectors has been emphasized in this article, it should be noted that any kind of object which can be represented digitally and which has an appropriate distance measure associated with it and in which the necessary operations for training are possible can be used to construct a self-organizing map. This includes matrices, continuous functions or even other self-organizing maps.
Preliminary definitions

Consider a nxm array of nodes each of which contains a weight vector and is aware of its location in the array. Each weight vector is of the same dimension as the node's input vector. The weights may initially be set to random values.

Now we need input to feed the map. (The generated map and the given input exist in separate subspaces.) We will create three vectors to represent colors. Colors can be represented by their red, green, and blue components. Consequently our input vectors will have three components, each corresponding to a color space. The input vectors will be:

\[
\begin{align*}
R &= <255, 0, 0> \\
G &= <0, 255, 0> \\
B &= <0, 0, 255>
\end{align*}
\]

The color training vector data sets used in SOM:

- threeColors = [255, 0, 0], [0, 255, 0], [0, 0, 255]
- eightColors = [0, 0, 0], [255, 0, 0], [0, 255, 0], [0, 0, 255], [255, 255, 0], [0, 255, 255], [255, 0, 255], [255, 255, 255]

The data vectors should preferably be normalized (vector length is equal to one) before training the SOM.

Neurons (40 x 40 square grid) are trained for 250 iterations with a learning rate of 0.1 using the normalized Iris flower data set which has four dimensional data vectors. A color image formed by first three dimensions of the four dimensional SOM weight vectors (top left), pseudo-color image of the magnitude of the SOM weight vectors (top right), U-Matrix (Euclidean distance between weight vectors of neighboring cells) of the SOM (bottom left) and overlay of data points (red: I. setosa, green: I. versicolor and blue: I. virginica) on the U-Matrix based on the minimum Euclidean distance between data vectors and SOM weight vectors (bottom right).

Variables

These are the variables needed, with vectors in bold,

- \( t \) denotes current iteration
- \( \tau \) is the limit on time iteration
- \( W \) is the current weight vector
- \( D \) is the target input
- \( \Theta(t) \) is restraint due to distance from BMU, usually called the neighborhood function, and
- \( \alpha(t) \) is learning restraint due to time.
Algorithm

1. Randomize the map's nodes' weight vectors
2. Grab an input vector
3. Traverse each node in the map
   1. Use Euclidean distance formula to find similarity between the input vector and the map's node's weight vector
   2. Track the node that produces the smallest distance (this node is the best matching unit, BMU)
4. Update the nodes in the neighborhood of BMU by pulling them closer to the input vector
   1. \( \mathbf{W}_v(t + 1) = \mathbf{W}_v(t) + \Theta(t) \alpha(t)(\mathbf{D}(t) - \mathbf{W}_v(t)) \)
5. Increase \( t \) and repeat from 2 while \( t < \lambda \)

Interpretation

There are two ways to interpret a SOM. Because in the training phase weights of the whole neighborhood are moved in the same direction, similar items tend to excite adjacent neurons. Therefore, SOM forms a semantic map where similar samples are mapped close together and dissimilar apart. This may be visualized by a U-Matrix (Euclidean distance between weight vectors of neighboring cells) of the SOM.\[6\]

The other way is to think of neuronal weights as pointers to the input space. They form a discrete approximation of the distribution of training samples. More neurons point to regions with high training sample concentration and fewer where the samples are scarce.

SOM may be considered a nonlinear generalization of Principal components analysis (PCA).\[7\] It has been shown, using both artificial and real geophysical data, that SOM has many advantages\[8][9\] over the conventional feature extraction methods such as Empirical Orthogonal Functions (EOF) or PCA.

Originally, SOM was not formulated as a solution to an optimisation problem. Nevertheless, there have been several attempts to modify the definition of SOM and to formulate an optimisation problem which gives similar results.\[10\]

For example, Elastic maps use for approximation of principal manifolds\[11\] the mechanical metaphor of elasticity and analogy of the map with elastic membrane and plate.

Alternatives

- The generative topographic map (GTM) is a potential alternative to SOMs. In the sense that a GTM explicitly requires a smooth and continuous mapping from the input space to the map space, it is topology preserving. However, in a practical sense, this measure of topological preservation is lacking.\[12\]
- The time adaptive self-organizing map (TASOM) network is an extension of the basic SOM. The TASOM employs adaptive learning rates and neighborhood functions. It also includes a scaling parameter to make the network invariant to scaling, translation and rotation of the input space. The TASOM and its variants have been used in several applications including adaptive clustering, multilevel thresholding, input space approximation, and active contour modeling.\[13\] Moreover, a Binary Tree TASOM or BTASOM, resembling a binary natural tree having nodes composed of TASOM networks has been proposed where the number of its levels and the number of its nodes are adaptive with its environment.\[14\]
- The growing self-organizing map (GSOM) is a growing variant of the self-organizing map. The GSOM was developed to address the issue of identifying a suitable map size in the SOM. It starts with a minimal number of
Self-organizing map

- nodes (usually four) and grows new nodes on the boundary based on a heuristic. By using a value called the spread factor, the data analyst has the ability to control the growth of the GSOM.

- The elastic maps approach borrows from the spline interpolation the idea of minimization of the elastic energy. In learning, it minimizes the sum of quadratic bending and stretching energy with the least squares approximation error.

**References**


**External links**


- Self-organizing maps for Ruby (http://ai4r.org/): Implementation of self-organizing maps in Ruby, for the AI4R project.

- Self-organizing map for JavaScript (http://github.com/LucidTechnics/som): An open-source implementation of a self-organizing map in JavaScript for node.js from Lucid Technics, LLC.

- Spice-SOM (http://www.spice.ci.ritsumei.ac.jp/~thangc/programs/): A free GUI application of self-organizing map

- IFCSoft (http://mathcs.emory.edu/~kthayer/ifcsoft/): An open-source Java platform for generating self-organizing maps
Hybrid intelligent system

**Hybrid intelligent system** denotes a software system which employs, in parallel, a combination of methods and techniques from artificial intelligence subfields as:

- Neuro-fuzzy systems
- Hybrid connectionist-symbolic models
- Fuzzy expert systems
- Connectionist expert systems
- Evolutionary neural networks
- Genetic fuzzy systems
- Rough fuzzy hybridization
- Reinforcement learning with fuzzy, neural, or evolutionary methods as well as symbolic reasoning methods.

From the cognitive science perspective, every natural intelligent system is hybrid because it performs mental operations on both the symbolic and subsymbolic levels. For the past few years there has been an increasing discussion of the importance of A.I. Systems Integration. Based on notions that there have already been created simple and specific AI systems (such as systems for computer vision, speech synthesis, etc., or software that employs some of the models mentioned above) and now is the time for integration to create broad AI systems. Proponents of this approach are researchers such as Marvin Minsky, Ron Sun, Aaron Sloman, and Michael A. Arbib.

An example hybrid is a hierarchical control system in which the lowest, reactive layers are sub-symbolic. The higher layers, having relaxed time constraints, are capable of reasoning from an abstract world model and performing planning.

**References**

- International Journal of Hybrid Intelligent Systems http://www.softcomputing.net/ijhis/
- International Conference on Hybrid Intelligent Systems http://his.hybridsystem.com/
- HIS'01: http://www.softcomputing.net/his01/
- HIS'02: http://tamarugo.cec.uchile.cl/~his02/
- HIS'03: http://www.softcomputing.net/his03/
- HIS'04: http://www.cs.nmt.edu/~his04
- HIS'05: http://www.ica.ele.puc-rio.br/his05
- HIS'06 http://his-nceri06.kedri.info/
- HIS'07 September 17-19, 2007, Kaiserslautern, Germany, http://www.eit.uni-kl.de/koenig/HIS07_Web/ his07main.html
- hybrid systems resources: http://www.cogsci.rpi.edu/~rsun/hybrid-resource.html
Neuro-fuzzy

In the field of artificial intelligence, neuro-fuzzy refers to combinations of artificial neural networks and fuzzy logic. Neuro-fuzzy was proposed by J. S. R. Jang. Neuro-fuzzy hybridization results in a hybrid intelligent system that synergizes these two techniques by combining the human-like reasoning style of fuzzy systems with the learning and connectionist structure of neural networks. Neuro-fuzzy hybridization is widely termed as Fuzzy Neural Network (FNN) or Neuro-Fuzzy System (NFS) in the literature. Neuro-fuzzy system (the more popular term is used henceforth) incorporates the human-like reasoning style of fuzzy systems through the use of fuzzy sets and a linguistic model consisting of a set of IF-THEN fuzzy rules. The main strength of neuro-fuzzy systems is that they are universal approximators with the ability to solicit interpretable IF-THEN rules.

The strength of neuro-fuzzy systems involves two contradictory requirements in fuzzy modeling: interpretability versus accuracy. In practice, one of the two properties prevails. The neuro-fuzzy in fuzzy modeling research field is divided into two areas: linguistic fuzzy modeling that is focused on interpretability, mainly the Mamdani model; and precise fuzzy modeling that is focused on accuracy, mainly the Takagi-Sugeno-Kang (TSK) model.

Although generally assumed to be the realization of a fuzzy system through connectionist networks, this term is also used to describe some other configurations including:

- Deriving fuzzy rules from trained RBF networks.
- Fuzzy logic based tuning of neural network training parameters.
- Fuzzy logic criteria for increasing a network size.
- Realising fuzzy membership function through clustering algorithms in unsupervised learning in SOMs and neural networks.
- Representing fuzzification, fuzzy inference and defuzzification through multi-layers feed-forward connectionist networks.

It must be pointed out that interpretability of the Mamdani-type neuro-fuzzy systems can be lost. To improve the interpretability of neuro-fuzzy systems, certain measures must be taken, wherein important aspects of interpretability of neuro-fuzzy systems are also discussed.[1]

Pseudo outer-product-based fuzzy neural networks

Pseudo outer-product-based fuzzy neural networks ("POPFNN") are a family of neuro-fuzzy systems that are based on the linguistic fuzzy model.[2]

Three members of POPFNN exist in the literature:

- POPFNN-AARS(S), which is based on the Approximate Analogical Reasoning Scheme[3]
- POPFNN-CRI(S), which is based on commonly accepted fuzzy Compositional Rule of Inference[4]
- POPFNN-TV, which is based on Truth Value Restriction

The "POPFNN" architecture is a five-layer neural network where the layers from 1 to 5 are called: input linguistic layer, condition layer, rule layer, consequent layer, output linguistic layer. The fuzzification of the inputs and the defuzzification of the outputs are respectively performed by the input linguistic and output linguistic layers while the fuzzy inference is collectively performed by the rule, condition and consequence layers.

The learning process of POPFNN consists of three phases:

1. Fuzzy membership generation
2. Fuzzy rule identification
3. Supervised fine-tuning

Various fuzzy membership generation algorithms can be used: Learning Vector Quantization (LVQ), Fuzzy Kohonen Partitioning (FKP) or Discrete Incremental Clustering (DIC). Generally, the POP algorithm and its variant LazyPOP are used to identify the fuzzy rules.

References


External links

- A Definition of Interpretability of Fuzzy Systems (http://www.soft-computing.de/detail1.html)
Genetic fuzzy systems

The term **Genetic fuzzy systems** is sometimes used to describe fuzzy systems constructed by using genetic algorithms or genetic programming to identify its structure and parameter. In a strict sense there is nothing genetic about the resulting fuzzy system, as the term genetic refers only to the identification method.

When it comes to automatically identify and build a fuzzy system, given the high degree of nonlinearity of the output, traditional linear optimization tools have several limitations. Therefore in the framework of soft computing, genetic algorithms (GAs) and genetic programming (GP) methods have been used successfully to identify structure and parameters of fuzzy systems.

Fuzzy systems

Fuzzy systems are fundamental methodologies to represent and process linguistic information, with mechanisms to deal with uncertainty and imprecision. Take for instance the task of modeling a driver parking a car. The closer we look at this problem, the more we realize the difficulty of writing down a rather concise mathematical model to describe this action. Yet, we actually can describe the action of this driver in terms of simple linguistic rules. With such remarkable attributes, fuzzy systems have been widely and successfully applied to control, classification and modeling problems (Mamdani, 1974) (Klir and Yuan, 1995) (Pedrycz and Gomide, 1998).

Although simplistic in its design, the identification of a fuzzy system is a rather complex task that comprises the identification of (a) the input and output variables, (b) the rule base (knowledge base), (c) the membership functions and (d) the mapping parameters.

Usually the rule base rule base consists of several IF-THEN rules, linking input(s) and output(s). A simple rule of a fuzzy controller could be:

**IF** (TEMPERATURE = HOT) **THEN** (COOLING = HIGH)

The numerical impact/meaning of this rule depends on how the membership functions of HOT and HIGH are shaped and defined.

The construction and identification of a fuzzy system can be divided into (a) the structure and (b) the parameter identification of a fuzzy system.

The structure of a fuzzy system is expressed by the input and output variables and the rule base, while the parameters of a fuzzy system are the rule parameters (defining the membership functions, the aggregation operator and the implication function) and the mapping parameters related to the mapping of a crisp set to a fuzzy set, and vice versa. (Bastian, 2000).

Much work has been done to develop or adapt methodologies that are capable of automatically identifying a fuzzy system from numerical data. Particularly in the framework of soft computing, significant methodologies have been proposed with the objective of building fuzzy systems by means of genetic algorithms (GAs) or genetic programming (GP).

Genetic algorithms for fuzzy system identification

Given the high degree of nonlinearity of the output of a fuzzy system, traditional linear optimization tools do have their limitations. Genetic algorithms have demonstrated to be a robust and very powerful tool to perform tasks such as the generation of fuzzy rule base, optimization of fuzzy rule bases, generation of membership functions, and tuning of membership functions (Cordón et al., 2001a). All these tasks can be considered as optimization or search processes within large solution spaces (Bastian and Hayashi, 1995) (Yuan and Zhuang, 1996) (Cordón et al., 2001b).
Genetic fuzzy systems

Genetic programming for fuzzy system identification

While genetic algorithms are very powerful tools to identify the fuzzy membership functions of a pre-defined rule base, they have their limitation especially when it also comes to identify the input and output variables of a fuzzy system from a given set of data. Genetic programming has been used to identify the input variables, the rule base as well as the involved membership functions of a fuzzy model (Bastian, 2000).

References


Knowledge engineering

Knowledge engineering (KE) was defined in 1983 by Edward Feigenbaum, and Pamela McCorduck as follows:

KE is an engineering discipline that involves integrating knowledge into computer systems in order to solve complex problems normally requiring a high level of human expertise.\(^1\)

At present, it refers to the building, maintaining and development of knowledge-based systems.\(^2\) It has a great deal in common with software engineering, and is used in many computer science domains such as artificial intelligence,\(^3\)[4] including databases, data mining, expert systems, decision support systems and geographic information systems. Knowledge engineering is also related to mathematical logic, as well as strongly involved in cognitive science and socio-cognitive engineering where the knowledge is produced by socio-cognitive aggregates (mainly humans) and is structured according to our understanding of how human reasoning and logic works.

Various activities of KE specific for the development of a knowledge-based system:

- Assessment of the problem
- Development of a knowledge-based system shell/structure
- Acquisition and structuring of the related information, knowledge and specific preferences (IPK model)
- Implementation of the structured knowledge into knowledge bases
- Testing and validation of the inserted knowledge
- Integration and maintenance of the system
- Revision and evaluation of the system.

Being still more art than engineering, KE is not as neat as the above list in practice. The phases overlap, the process might be iterative, and many challenges could appear.
Knowledge engineering principles

Since the mid-1980s, knowledge engineers have developed a number of principles, methods and tools to improve the knowledge acquisition and ordering. Some of the key principles are:

- There are different:
  - types of knowledge each requiring its own approach and technique.
  - types of experts and expertise, such that methods should be chosen appropriately.
  - ways of representing knowledge, which can aid the acquisition, validation and re-use of knowledge.
  - ways of using knowledge, so that the acquisition process can be guided by the project aims (goal-oriented).
- Structured methods increase the efficiency of the acquisition process.
- Knowledge Engineering is the process of eliciting Knowledge for any purpose be it Expert system or AI development

Views of knowledge engineering

There are two main views to knowledge engineering:[5]

- Transfer View – This is the traditional view. In this view, the assumption is to apply conventional knowledge engineering techniques to transfer human knowledge into artificial intelligence systems.
- Modeling View – This is the alternative view. In this view, the knowledge engineer attempts to model the knowledge and problem solving techniques of the domain expert into the artificial intelligence system.

A major concern in knowledge engineering is the construction of ontologies. One philosophical question in this area is the debate between foundationalism and coherentism - are fundamental axioms of belief required, or merely consistency of beliefs which may have no lower-level beliefs to justify them?

Overview of Trends in Knowledge Engineering

Some of the trends in Knowledge Engineering in the last few years are discussed in this section. The text below is a brief overview of paper "Knowledge Engineering: Principles and methods" authored by Rudi Studer, V. Richard Benjamins and Dieter Fensel.

The paradigm Shift from a transfer view to a modeling view

According to the transfer view the human knowledge required to solve a problem is transferred and implemented into the knowledge base. However this assumes that concrete knowledge is already present in humans to solve a problem. The transfer view disregards the tacit knowledge an individual acquires in order to solve a problem. This is one of the reasons for a paradigm shift towards modeling view. This shift is compared to a shift from first generation expert systems to second generation expert systems.

The modeling view is a closer approximate of reality and perceives solving problems as a dynamic, cyclic, incessant process dependent on the knowledge acquired and the interpretations made by the system. This is similar to how an expert solves problems in real life.

The evolving of Role Limiting methods and Generic Tasks

Role limiting methods are based on reusable problem solving methods. Different knowledge roles are decided and the knowledge expected from each of these roles is clarified. However the disadvantage of role limiting methods is that there is no logical means of deciding whether a specific problem can be solved by a specific role-limiting method.

This disadvantage gave rise to Configurable role limiting methods. Configurable role limiting methods are based on the idea that a problem solving method can further be broken up into several smaller sub tasks each task solved by its own problem solving method.
Generic Tasks include a rigid knowledge structure, a standard strategy to solve problems, a specific input and a specific output.

The GT approach is based on the strong interaction problem hypothesis which states that the structure and representation of domain knowledge is completely determined by its use.

The usage of Modeling Frameworks

The development of Specification languages and problem solving methods of knowledge based systems. Over the past few years the modeling frameworks that became prominent within Knowledge engineering are Common KADS, MIKE (Model-based and Incremental knowledge engineering) and PROTÉGÉ-II. PROTÉGÉ-II is a modeling framework influenced by the concept of ‘Ontology’.

The influence of Ontology

Ontologies help building model of a domain and define the terms inside the domain and the relationships between them. There are different types of Ontologies including Domain ontologies, Generic ontologies, application ontologies and representational ontologies.

While categorizing knowledge, storing, retrieving and managing information is not only useful for solving problems without direct need of human expertise but also leads to ‘Knowledge Management’ efforts that enable an organization to function efficiently in the long run.

Bibliography


External links

- Data & Knowledge Engineering (http://www.elsevier.com/wps/find/journaldescription.cws_home/505608/description#description) – Elsevier Journal
- IEEE Transactions on Knowledge and Data Engineering (http://www.informatik.uni-trier.de/~ley/db/journals/tkde/index.html)
Knowledge retrieval

**Knowledge Retrieval** seeks to return information in a structured form, consistent with human cognitive processes as opposed to simple lists of data items. It draws on a range of fields including epistemology (theory of knowledge), cognitive psychology, cognitive neuroscience, logic and inference, machine learning and knowledge discovery, linguistics, and information technology.

**Overview**

In the field of retrieval systems, established approaches include:

- Data Retrieval Systems (DRS), such as database management systems, are well suited for the storage and retrieval of structured data.
- Information Retrieval Systems (IRS), such as web search engines, are very effective in finding the relevant documents or web pages.

Both approaches require a user to read and analyze often long lists of data sets or documents in order to extract meaning.

The goal of knowledge retrieval systems is to reduce the burden of those processes by improved search and representation. This improvement is needed to leverage the increasing data volumes available on the Internet. [1][2][3][4][5][6][7][8][9][10][11]

**Comparison of Data, Information and Knowledge Retrieval**

Data Retrieval and Information Retrieval are earlier and more basic forms of information access. [12]

<table>
<thead>
<tr>
<th></th>
<th>Data Retrieval</th>
<th>Information Retrieval</th>
<th>Knowledge Retrieval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Match</td>
<td>Boolean match</td>
<td>partial match, best match</td>
<td>partial match, best match</td>
</tr>
<tr>
<td>Inference</td>
<td>deductive inference</td>
<td>inductive inference</td>
<td>deductive inference, inductive inference, associative reasoning, analogical reasoning</td>
</tr>
<tr>
<td>Model</td>
<td>deterministic model</td>
<td>statistical and probabilistic model</td>
<td>semantic model, inference model</td>
</tr>
<tr>
<td>Query</td>
<td>artificial language</td>
<td>natural language</td>
<td>knowledge structure, natural language</td>
</tr>
<tr>
<td>Organization</td>
<td>table, index</td>
<td>table, index</td>
<td>knowledge unit, knowledge structure</td>
</tr>
<tr>
<td>Representation</td>
<td>number, rule</td>
<td>natural language, markup language</td>
<td>concept graph, predicate logic, production rule, frame, semantic network, ontology</td>
</tr>
<tr>
<td>Storage</td>
<td>database</td>
<td>document collections</td>
<td>knowledge base</td>
</tr>
<tr>
<td>Retrieved Results</td>
<td>data set</td>
<td>sections or documents</td>
<td>a set of knowledge unit</td>
</tr>
</tbody>
</table>

Knowledge retrieval (KR) focuses on the knowledge level. We need to examine how to extract, represent, and use the knowledge in data and information. [13] Knowledge retrieval systems provide knowledge to users in a structured way. Compared to data retrieval and information retrieval, they use different inference models, retrieval methods, result organization, etc. Table 1, extending van Rijsbergen’s comparison of the difference between data retrieval and information retrieval [14], summarizes the main characteristics of data retrieval, information retrieval, and knowledge retrieval. [15] The core of data retrieval and information retrieval is retrieval subsystems. Data retrieval gets results through Boolean match. [16] Information retrieval uses partial match and best match. Knowledge retrieval is also based on partial match and best match.
From an inference perspective, data retrieval uses deductive inference, and information retrieval uses inductive inference. Considering the limitations from the assumptions of different logics, traditional logic systems (e.g., Horn subset of first order logic) cannot reasoning efficiently. Associative reasoning, analogical reasoning and the idea of unifying reasoning and search may be effective methods of reasoning at the web scale.

From the retrieval perspective, knowledge retrieval systems focus on semantics and better organization of information. Data retrieval and information retrieval organize the data and documents by indexing, while knowledge retrieval organize information by indicating connections between elements in those documents.

**Frameworks for Knowledge Retrieval systems**

From computer science perspective, a logic framework concentrating on fuzziness of knowledge queries has been proposed and investigated in detail. Markup languages for knowledge reasoning and relevant strategies have been investigated, which may serve as possible logic reasoning foundations for text based knowledge retrieval.

From cognitive science perspective, especially from cognitive psychology and cognitive neuroscience perspective, the neurobiological basis for knowledge retrieval in the human brain has been investigated, and may serve as a cognitive model for knowledge retrieval.

**Knowledge Retrieval related Disciplines**

Knowledge retrieval can draw results from the following related theories and technologies:

- **Theory of Knowledge**: knowledge acquisition, knowledge organization, knowledge representation, knowledge validation, knowledge management.
- **Cognitive Science**: cognitive psychology, cognitive neuroscience, cognitive informatics, concept formation and learning, decision making, human–computer interaction.
- **Machine Learning and Knowledge Discovery**: preprocessing, classification, clustering, prediction, postprocessing, statistical learning theory.
- **Logic and Inference**: propositional logic, predicate logic, attribute logic, universal logic, inductive inference, deductive inference, associative reasoning, analogical reasoning, approximate reasoning.
- **Information Technology**: information theory, information science, information retrieval, database systems, knowledge-based systems, rule-based systems, expert systems, decision support systems, intelligent agent technology.
- **Linguistics**: computational linguistics, natural language understanding, natural language processing.

Topics listed under each entry serve as examples and do not form a complete list. And many related disciplines should be added as the field grows mature.

**References**

Knowledge retrieval


Knowledge acquisition

Knowledge acquisition is a method of learning, first proposed by Aristotle in his seminal work "Organon". Aristotle proposed that the mind at birth is a blank slate, or tabula rasa. As a blank slate it contains no knowledge of the objective, empirical universe, nor of itself.

("Knowing subject" is often the description of a mind with acquired knowledge; It is found in the writings of David Hume, Karl Popper, and many others. [See also subject-object problem]. Therefore, a human mind cannot be a "knowing subject" until it has "acquired knowledge". "Acquired" in this sense can be either an adjective, as in "that which has been acquired"; or a verb, as in the act of acquisition.)

As a method, it is opposed to the concept of "a priori" knowledge, and to "intuition" when conceived as religious revelation.

It has been suggested [1][2] that the mind is "hard wired" to begin operating at birth, beginning a lifetime process of acquisition through abstraction, induction, and conception.

The acquisition of empirical knowledge, which begins the process of filling the tabula rasa, is thus by means of the experience of sensation and perception. Though sensation and perception are described elsewhere in Wikipedia as parts of "psychology, and not [of] anatomy or physiology," they belong to cognitive science. [See also cognitive revolution and philosophy of perception.]

The "five senses" referred to by the word sensation [see sense] are metaphorically the interface between empirical (sensate) reality and the consciousness of the knowing subject. A knowing subject for the purpose of this discussion of knowledge acquisition may be defined as any conscious creature capable of deriving direct and immediate sensate data from its environment.

Sensate data, or sensation, are distinct from perception. Perception is the recognition within the knowing subject of the event of having had a sensation. The tabula rasa and must learn the nature of sensation as the awareness of something which is outside itself. Commonly recognized sensory systems are those for vision, hearing, somatic sensation (touch), taste and olfaction (smell). [See sensory system]
Perception is the *retention of a group of sensations* transmitted through the sensory system(s), which gives the knowing subject the ability to be aware, not only of the singularity of stimuli presented by sensation itself, but of an entity, a thing, an existent.[3]

Retention of percepts allows the human mind to *abstract* information from the percepts. The abstraction is considered the extensional definition of the percept. An extension is "every object that falls under the definition of the concept or term in question." [4] This is the same as a universal (metaphysics) or genus or denotation, or class (philosophy).

Once a universal (class) has been identified, then the next step in the acquisition of knowledge is the abstraction of the intension, which is the particular, the species, or the connotation. Connotation as its meaning as particular is "the assertion that at least one member of one class of things is either included or excluded as a member of some other class." [5] This means, for example, that a poodle is the particular in a class or universal concept called "dog" or "canine".

Knowledge management

**Knowledge management (KM)** comprises a range of strategies and practices used in an organization to identify, create, represent, distribute, and enable adoption of insights and experiences. Such insights and experiences comprise knowledge, either embodied in individuals or embedded in organizations as processes or practices.

An established discipline since 1991 (see Nonaka 1991), KM includes courses taught in the fields of business administration, information systems, management, and library and information sciences (Alavi & Leidner 1999). More recently, other fields have started contributing to KM research; these include information and media, computer science, public health, and public policy.

Many large companies and non-profit organizations have resources dedicated to internal KM efforts, often as a part of their business strategy, information technology, or human resource management departments (Addicott, McGivern & Ferlie 2006). Several consulting companies also exist that provide strategy and advice regarding KM to these organizations.

Knowledge management efforts typically focus on organizational objectives such as improved performance, competitive advantage, innovation, the sharing of lessons learned, integration and continuous improvement of the organization. KM efforts overlap with organizational learning, and may be distinguished from that by a greater focus on the management of knowledge as a strategic asset and a focus on encouraging the sharing of knowledge. It is seen as an enabler of organisational learning[1] and a more concrete mechanism than the previous abstract research.

**History**

KM efforts have a long history, to include on-the-job discussions, formal apprenticeship, discussion forums, corporate libraries, professional training and mentoring programs. More recently, with increased use of computers in the second half of the 20th century, specific adaptations of technologies such as knowledge bases, expert systems, knowledge repositories, group decision support systems, intranets, and computer-supported cooperative work have been introduced to further enhance such efforts.[2]

In 1999, the term personal knowledge management was introduced which refers to the management of knowledge at the individual level (Wright 2005).
In terms of the enterprise, early collections of case studies recognized the importance of knowledge management dimensions of strategy, process, and measurement (Morey, Maybury & Thuraisingham 2002). Key lessons learned included: people and the cultural norms which influence their behaviors are the most critical resources for successful knowledge creation, dissemination, and application; cognitive, social, and organizational learning processes are essential to the success of a knowledge management strategy; and measurement, benchmarking, and incentives are essential to accelerate the learning process and to drive cultural change. In short, knowledge management programs can yield impressive benefits to individuals and organizations if they are purposeful, concrete, and action-oriented.

More recently with the advent of the Web 2.0, the concept of Knowledge Management has evolved towards a vision more based on people participation and emergence. This line of evolution is termed Enterprise 2.0 (McAfee 2006). However, there is an ongoing debate and discussions (Lakhani & McAfee 2007) as to whether Enterprise 2.0 is just a fad that does not bring anything new or useful or whether it is, indeed, the future of knowledge management (Davenport 2008).

**Research**

KM emerged as a scientific discipline in the earlier 1990s. It was initially supported solely by practitioners, when Skandia hired Leif Edvinsson of Sweden as the world’s first Chief Knowledge Officer (CKO). Hubert Saint-Onge (formerly of CIBC, Canada), started investigating various sides of KM long before that. The objective of CKOs is to manage and maximize the intangible assets of their organizations. Gradually, CKOs became interested in not only practical but also theoretical aspects of KM, and the new research field was formed. The KM ideas taken up by academics, such as Ikujiro Nonaka (Hitotsubashi University), Hirotaka Takeuchi (Hitotsubashi University), Thomas H. Davenport (Babson College) and Baruch Lev (New York University). In 2001, Thomas A. Stewart, former editor at FORTUNE Magazine and subsequently the editor of *Harvard Business Review*, published a cover story highlighting the importance of intellectual capital of organizations. Since its establishment, the KM discipline has been gradually moving towards academic maturity. First, there is a trend towards higher cooperation among academics; particularly, there has been a drop in single-authored publications. Second, the role of practitioners has changed. Their contribution to academic research has been dramatically declining from 30% of overall contributions up to 2002, to only 10% by 2009 (Serenko et al. 2010).

A broad range of thoughts on the KM discipline exist; approaches vary by author and school. As the discipline matures, academic debates have increased regarding both the theory and practice of KM, to include the following perspectives:

- **Techno-centric** with a focus on technology, ideally those that enhance knowledge sharing and creation.
- **Organizational** with a focus on how an organization can be designed to facilitate knowledge processes best.
- **Ecological** with a focus on the interaction of people, identity, knowledge, and environmental factors as a complex adaptive system akin to a natural ecosystem.

Regardless of the school of thought, core components of KM include people, processes, technology (or) culture, structure, technology, depending on the specific perspective (Spender & Scherer 2007). Different KM schools of thought include various lenses through which KM can be viewed and explained, to include:

- community of practice (Wenger, McDermott & Synder 2001)
- social network analysis
- intellectual capital (Bontis & Choo 2002)
- information theory (McInerney 2002)
- complexity science
- constructivism (Nanjappa & Grant 2003)

The practical relevance of academic research in KM has been questioned (Ferguson 2005) with action research suggested as having more relevance (Andriessen 2004) and the need to translate the findings presented in academic journals to a practice (Booker, Bontis & Serenko 2008).


Dimensions

Different frameworks for distinguishing between different 'types of' knowledge exist. One proposed framework for categorizing the dimensions of knowledge distinguishes between tacit knowledge and explicit knowledge. Tacit knowledge represents internalized knowledge that an individual may not be consciously aware of, such as how he or she accomplishes particular tasks. At the opposite end of the spectrum, explicit knowledge represents knowledge that the individual holds consciously in mental focus, in a form that can easily be communicated to others.\[^{10}\] (Alavi & Leidner 2001). Similarly, Hayes and Walsham (2003) describe content and relational perspectives of knowledge and knowledge management as two fundamentally different epistemological perspectives. The content perspective suggest that knowledge is easily stored because it may be codified, while the relational perspective recognizes the contextual and relational aspects of knowledge which can make knowledge difficult to share outside of the specific location where the knowledge is developed.\[^{11}\]

Early research suggested that a successful KM effort needs to convert internalized tacit knowledge into explicit knowledge in order to share it, but the same effort must also permit individuals to internalize and make personally meaningful any codified knowledge retrieved from the KM effort. Subsequent research into KM suggested that a distinction between tacit knowledge and explicit knowledge represented an oversimplification and that the notion of explicit knowledge is self-contradictory. Specifically, for knowledge to be made explicit, it must be translated into information (i.e., symbols outside of our heads) (Serenko & Bontis 2004). Later on, Ikujiro Nonaka proposed a model (SECI for Socialization, Externalization, Combination, Internalization) which considers a spiraling knowledge process interaction between explicit knowledge and tacit knowledge (Nonaka & Takeuchi 1995). In this model, knowledge follows a cycle in which implicit knowledge is 'extracted' to become explicit knowledge, and explicit knowledge is 're-internalized' into implicit knowledge. More recently, together with Georg von Krogh, Nonaka returned to his earlier work in an attempt to move the debate about knowledge conversion forwards (Nonaka & von Krogh 2009).

A second proposed framework for categorizing the dimensions of knowledge distinguishes between embedded knowledge of a system outside of a human individual (e.g., an information system may have knowledge embedded into its design) and embodied knowledge representing a learned capability of a human body's nervous and endocrine systems (Sensky 2002).

A third proposed framework for categorizing the dimensions of knowledge distinguishes between the exploratory creation of "new knowledge" (i.e., innovation) vs. the transfer or exploitation of "established knowledge" within a group, organization, or community. Collaborative environments such as communities of practice or the use of social computing tools can be used for both knowledge creation and transfer.\[^{12}\]
Strategies

Knowledge may be accessed at three stages: before, during, or after KM-related activities. Different organizations have tried various knowledge capture incentives, including making content submission mandatory and incorporating rewards into performance measurement plans. Considerable controversy exists over whether incentives work or not in this field and no consensus has emerged.

One strategy to KM involves actively managing knowledge (push strategy). In such an instance, individuals strive to explicitly encode their knowledge into a shared knowledge repository, such as a database, as well as retrieving knowledge they need that other individuals have provided to the repository. This is also commonly known as the Codification approach to KM.

Another strategy to KM involves individuals making knowledge requests of experts associated with a particular subject on an ad hoc basis (pull strategy). In such an instance, expert individual(s) can provide their insights to the particular person or people needing this (Snowden 2002). This is also commonly known as the Personalization approach to KM.

Other knowledge management strategies and instruments for companies include:

- rewards (as a means of motivating for knowledge sharing)
- storytelling (as a means of transferring tacit knowledge)
- cross-project learning
- after action reviews
- knowledge mapping (a map of knowledge repositories within a company accessible by all)
- communities of practice
- expert directories (to enable knowledge seeker to reach to the experts)
- best practice transfer
- knowledge fairs
- competence management (systematic evaluation and planning of competences of individual organization members)
- proximity & architecture (the physical situation of employees can be either conducive or obstructive to knowledge sharing)
- master-apprentice relationship
- collaborative technologies (groupware, etc.)
- knowledge repositories (databases, bookmarking engines, etc.)
- measuring and reporting intellectual capital (a way of making explicit knowledge for companies)
- knowledge brokers (some organizational members take on responsibility for a specific "field" and act as first reference on whom to talk about a specific subject)
- social software (wikis, social bookmarking, blogs, etc.)
- Inter-project knowledge transfer

Motivations

A number of claims exist as to the motivations leading organizations to undertake a KM effort. Typical considerations driving a KM effort include:

- Making available increased knowledge content in the development and provision of products and services
- Achieving shorter new product development cycles
- Facilitating and managing innovation and organizational learning
- Leveraging the expertise of people across the organization
- Increasing network connectivity between internal and external individuals
- Managing business environments and allowing employees to obtain relevant insights and ideas appropriate to their work
• Solving intractable or wicked problems
• Managing intellectual capital and intellectual assets in the workforce (such as the expertise and know-how possessed by key individuals)

Debate exists whether KM is more than a passing fad, though increasing amount of research in this field may hopefully help to answer this question, as well as create consensus on what elements of KM help determine the success or failure of such efforts (Wilson 2002). Knowledge Sharing remains a challenging issue for knowledge management, and while there is no clear agreement barriers may include time issues for knowledge works, the level of trust, lack of effective support technologies and culture (Jennex 2008).

Technologies
Early KM technologies included online corporate yellow pages as expertise locators and document management systems. Combined with the early development of collaborative technologies (in particular Lotus Notes), KM technologies expanded in the mid-1990s. Subsequent KM efforts leveraged semantic technologies for search and retrieval and the development of e-learning tools for communities of practice (Capozzi 2007). Knowledge management systems can thus be categorized as falling into one or more of the following groups: Groupware, document management systems, expert systems, semantic networks, relational and object oriented databases, simulation tools, and artificial intelligence (Gupta & Sharma 2004)

More recently, development of social computing tools (such as bookmarks, blogs, and wikis) have allowed more unstructured, self-governing or ecosystem approaches to the transfer, capture and creation of knowledge, including the development of new forms of communities, networks, or matrixed organizations. However such tools for the most part are still based on text and code, and thus represent explicit knowledge transfer. These tools face challenges in distilling meaningful re-usable knowledge and ensuring that their content is transmissible through diverse channels (Andrus 2005).

Software tools in knowledge management are a collection of technologies and are not necessarily acquired as a single software solution. Furthermore, these knowledge management software tools have the advantage of using the organization existing information technology infrastructure. Organizations and business decision makers spend a great deal of resources and make significant investments in the latest technology, systems and infrastructure to support knowledge management. It is imperative that these investments are validated properly, made wisely and that the most appropriate technologies and software tools are selected or combined to facilitate knowledge management. Knowledge management has also become a cornerstone in emerging business strategies such as Service Lifecycle Management (SLM) with companies increasingly turning to software vendors to enhance their efficiency in industries including, but not limited to, the aviation industry.

Notes
References

This article is based on material taken from the Free On-line Dictionary of Computing prior to 1 November 2008 and incorporated under the "relicensing" terms of the GFDL, version 1.3 or later.


• Rhetorical Structure Theory (assumed from the reference of RST Theory above) http://acl.ldc.upenn.edu/W/W01/W01-1605.pdf


• The RST site at http://www.sfu.ca/rst/run by Bill Mann


**External links**

• Knowledge management (http://www.dmoz.org/Reference/Knowledge_Management/) at the Open Directory Project

• Knowledge@work community (http://www.ami-communities.eu/wiki/Knowledge@Work)
Data warehouse

In computing, a data warehouse (DW or DWH) is a database used for reporting and data analysis. The data stored in the warehouse are uploaded from the operational systems (such as marketing, sales etc., shown in the figure to the right). The data may pass through an operational data store for additional operations before they are used in the DW for reporting.

The typical ETL-based data warehouse uses staging, integration, and access layers to house its key functions. The staging layer or staging database stores raw data extracted from each of the disparate source data systems. The integration layer integrates the disparate data sets by transforming the data from the staging layer often storing transformed data in an ODS database. The integrated data are then moved to yet another database, often called the data warehouse database, where the data is arranged into hierarchal groups often called dimensions and into facts and aggregate facts. The combination of facts and dimensions is sometimes called a star schema. The access layer helps users retrieve data.

A data warehouse constructed from integrated data source systems does not require ETL, staging databases, or operational data store databases. The integrated data source systems may be considered to be a part of a distributed operational data store layer. Data federation methods or data virtualization methods may be used to access the distributed integrated source data systems to consolidate and aggregate data directly into the data warehouse database tables. Unlike the ETL-based data warehouse, the integrated source data systems and the data warehouse are all integrated since there is no transformation of dimensional or reference data. This integrated data warehouse architecture supports the drill down from the aggregate data of the data warehouse to the transactional data of the integrated source data systems.

Data warehouses can be subdivided into data marts. Data marts store subsets of data from a warehouse. This definition of the data warehouse focuses on data storage. The main source of the data is cleaned, transformed, cataloged and made available for use by managers and other business professionals for data mining, online analytical processing, market research and decision support (Marakas & O’Brien 2009). However, the means to retrieve and analyze data, to extract, transform and load data, and to manage the data dictionary are also considered essential components of a data warehousing system. Many references to data warehousing use this broader context. Thus, an expanded definition for data warehousing includes business intelligence tools, tools to extract, transform and load data into the repository, and tools to manage and retrieve metadata.

Benefits of a data warehouse

A data warehouse maintains a copy of information from the source transaction systems. This architectural complexity provides the opportunity to:

• Maintain data history, even if the source transaction systems do not.
• Integrate data from multiple source systems, enabling a central view across the enterprise. This benefit is always valuable, but particularly so when the organization has grown by merger.
• Improve data quality, by providing consistent codes and descriptions, flagging or even fixing bad data.
• Present the organization’s information consistently.
• Provide a single common data model for all data of interest regardless of the data’s source.
• Restructure the data so that it makes sense to the business users.
• Restructure the data so that it delivers excellent query performance, even for complex analytic queries, without impacting the operational systems.
• Add value to operational business applications, notably customer relationship management (CRM) systems.

**History**

The concept of data warehousing dates back to the late 1980s[2] when IBM researchers Barry Devlin and Paul Murphy developed the "business data warehouse". In essence, the data warehousing concept was intended to provide an architectural model for the flow of data from operational systems to decision support environments. The concept attempted to address the various problems associated with this flow, mainly the high costs associated with it. In the absence of a data warehousing architecture, an enormous amount of redundancy was required to support multiple decision support environments. In larger corporations it was typical for multiple decision support environments to operate independently. Though each environment served different users, they often required much of the same stored data. The process of gathering, cleaning and integrating data from various sources, usually from long-term existing operational systems (usually referred to as legacy systems), was typically in part replicated for each environment. Moreover, the operational systems were frequently reexamined as new decision support requirements emerged. Often new requirements necessitated gathering, cleaning and integrating new data from "data marts" that were tailored for ready access by users.

Key developments in early years of data warehousing were:

• 1960s — General Mills and Dartmouth College, in a joint research project, develop the terms *dimensions* and *facts*.\[^{3}\]
• 1970s — ACNielsen and IRI provide dimensional data marts for retail sales.\[^{3}\]
• 1970s — Bill Inmon begins to define and discuss the term: Data Warehouse
• 1975 — Sperry Univac Introduce MAPPER (MAintain, Prepare, and Produce Executive Reports) is a database management and reporting system that includes the world's first 4GL. It was the first platform specifically designed for building Information Centers (a forerunner of contemporary Enterprise Data Warehousing platforms)
• 1983 — Teradata introduces a database management system specifically designed for decision support.
• 1983 — Sperry Corporation Martyn Richard Jones defines the Sperry Information Center approach, which while not being a true DW in the Inmon sense, did contain many of the characteristics of DW structures and process as defined previously by Inmon, and later by Devlin. First used at the TSB England & Wales
• 1984 — Metaphor Computer Systems, founded by David Liddle and Don Massaro, releases Data Interpretation System (DIS). DIS was a hardware/software package and GUI for business users to create a database management and analytic system.
• 1988 — Barry Devlin and Paul Murphy publish the article *An architecture for a business and information system* \[^{4}\] in *IBM Systems Journal* where they introduce the term "business data warehouse".
• 1991 — Prism Solutions, founded by Bill Inmon, introduces Prism Warehouse Manager, software for developing a data warehouse.
• 1992 — Bill Inmon publishes the book *Building the Data Warehouse*.\[^{5}\]
• 1995 — The Data Warehousing Institute, a for-profit organization that promotes data warehousing, is founded.
• 1996 — Ralph Kimball publishes the book *The Data Warehouse Toolkit*.\[^{6}\]
• 2000 — Daniel Linstedt releases the *Data Vault*, enabling real time auditable Data Warehouses warehouse.
Data warehouse

**Dimensional vs. Normalized approach for storage of data**

There are two leading approaches to storing data in a data warehouse — the dimensional approach and the normalized approach.

The dimensional approach, whose supporters are referred to as “Kimballites”, believe in Ralph Kimball’s approach in which it is stated that the data warehouse should be modeled using a Dimensional Model/star schema. The normalized approach, also called the 3NF model, whose supporters are referred to as “Inmonites”, believe in Bill Inmon’s approach in which it is stated that the data warehouse should be modeled using an E-R model/normalized model.

In a dimensional approach, transaction data are partitioned into "facts", which are generally numeric transaction data, and "dimensions", which are the reference information that gives context to the facts. For example, a sales transaction can be broken up into facts such as the number of products ordered and the price paid for the products, and into dimensions such as order date, customer name, product number, order ship-to and bill-to locations, and salesperson responsible for receiving the order.

A key advantage of a dimensional approach is that the data warehouse is easier for the user to understand and to use. Also, the retrieval of data from the data warehouse tends to operate very quickly. Dimensional structures are easy to understand for business users, because the structure is divided into measurements/facts and context/dimensions. Facts are related to the organization's business processes and operational system whereas the dimensions surrounding them contain context about the measurement (Kimball, Ralph 2008).

The main disadvantages of the dimensional approach are:

1. In order to maintain the integrity of facts and dimensions, loading the data warehouse with data from different operational systems is complicated, and
2. It is difficult to modify the data warehouse structure if the organization adopting the dimensional approach changes the way in which it does business.

In the normalized approach, the data in the data warehouse are stored following, to a degree, database normalization rules. Tables are grouped together by *subject areas* that reflect general data categories (e.g., data on customers, products, finance, etc.). The normalized structure divides data into entities, which creates several tables in a relational database. When applied in large enterprises the result is dozens of tables that are linked together by a web of joins. Furthermore, each of the created entities is converted into separate physical tables when the database is implemented (Kimball, Ralph 2008). The main advantage of this approach is that it is straightforward to add information into the database. A disadvantage of this approach is that, because of the number of tables involved, it can be difficult for users both to:

1. join data from different sources into meaningful information and then
2. access the information without a precise understanding of the sources of data and of the data structure of the data warehouse.

It should be noted that both normalized – and dimensional models can be represented in entity-relationship diagrams as both contain joined relational tables. The difference between the two models is the degree of normalization.

These approaches are not mutually exclusive, and there are other approaches. Dimensional approaches can involve normalizing data to a degree (Kimball, Ralph 2008).

In Information-Driven Business (Wiley 2010), Robert Hillard proposes an approach to comparing the two approaches based on the information needs of the business problem. The technique shows that normalized models hold far more information than their dimensional equivalents (even when the same fields are used in both models) but this extra information comes at the cost of usability. The technique measures information quantity in terms of Information Entropy and usability in terms of the Small Worlds data transformation measure.
**Top-down versus bottom-up design methodologies**

**Bottom-up design**

Ralph Kimball, a well-known author on data warehousing, is a proponent of an approach to data warehouse design which he describes as *bottom-up.*

In the *bottom-up* approach, data marts are first created to provide reporting and analytical capabilities for specific business processes. Though it is important to note that in Kimball methodology, the bottom-up process is the result of an initial business-oriented Top-down analysis of the relevant business processes to be modelled.

Data marts contain, primarily, dimensions and facts. Facts can contain either atomic data and, if necessary, summarized data. The single data mart often models a specific business area such as "Sales" or "Production." These data marts can eventually be integrated to create a comprehensive data warehouse. The integration of data marts is managed through the implementation of what Kimball calls "a data warehouse bus architecture." The data warehouse bus architecture is primarily an implementation of "the bus", a collection of conformed dimensions and conformed facts, which are dimensions that are shared (in a specific way) between facts in two or more data marts.

The integration of the data marts in the data warehouse is centered on the conformed dimensions (residing in "the bus") that define the possible integration "points" between data marts. The actual integration of two or more data marts is then done by a process known as "Drill across". A drill-across works by grouping (summarizing) the data along the keys of the (shared) conformed dimensions of each fact participating in the "drill across" followed by a join on the keys of these grouped (summarized) facts.

Maintaining tight management over the data warehouse bus architecture is fundamental to maintaining the integrity of the data warehouse. The most important management task is making sure dimensions among data marts are consistent. In Kimball's words, this means that the dimensions "conform".

Some consider it an advantage of the Kimball method, that the data warehouse ends up being "segmented" into a number of logically self-contained (up to and including The Bus) and consistent data marts, rather than a big and often complex centralized model. Business value can be returned as quickly as the first data marts can be created, and the method gives itself well to an exploratory and iterative approach to building data warehouses. For example, the data warehousing effort might start in the "Sales" department, by building a Sales-data mart. Upon completion of the Sales-data mart, the business might then decide to expand the warehousing activities into the, say, "Production department" resulting in a Production data mart. The requirement for the Sales data mart and the Production data mart to be integrable, is that they share the same "Bus", that will be, that the data warehousing team has made the effort to identify and implement the conformed dimensions in the bus, and that the individual data marts links that information from the bus. Note that this does not require 100% awareness from the onset of the data warehousing effort, no master plan is required upfront. The Sales-data mart is good as it is (assuming that the bus is complete) and the Production-data mart can be constructed virtually independent of the Sales-data mart (but not independent of the Bus).

If integration via the bus is achieved, the data warehouse, through its two data marts, will not only be able to deliver the specific information that the individual data marts are designed to do, in this example either "Sales" or "Production" information, but can deliver integrated Sales-Production information, which, often, is of critical business value. An integration (possibly) achieved in a flexible and iterative fashion.
Top-down design

Bill Inmon, one of the first authors on the subject of data warehousing, has defined a data warehouse as a centralized repository for the entire enterprise. Inmon is one of the leading proponents of the top-down approach to data warehouse design, in which the data warehouse is designed using a normalized enterprise data model. "Atomic" data, that is, data at the lowest level of detail, are stored in the data warehouse. Dimensional data marts containing data needed for specific business processes or specific departments are created from the data warehouse. In the Inmon vision, the data warehouse is at the center of the "Corporate Information Factory" (CIF), which provides a logical framework for delivering business intelligence (BI) and business management capabilities.

Inmon states that the data warehouse is:

Subject-oriented

The data in the data warehouse is organized so that all the data elements relating to the same real-world event or object are linked together.

Non-volatile

Data in the data warehouse are never over-written or deleted — once committed, the data are static, read-only, and retained for future reporting.

Integrated

The data warehouse contains data from most or all of an organization's operational systems and these data are made consistent.

Time-variant

For An operational system, the stored data contains the current value.

The top-down design methodology generates highly consistent dimensional views of data across data marts since all data marts are loaded from the centralized repository. Top-down design has also proven to be robust against business changes. Generating new dimensional data marts against the data stored in the data warehouse is a relatively simple task. The main disadvantage to the top-down methodology is that it represents a very large project with a very broad scope. The up-front cost for implementing a data warehouse using the top-down methodology is significant, and the duration of time from the start of project to the point that end users experience initial benefits can be substantial. In addition, the top-down methodology can be inflexible and unresponsive to changing departmental needs during the implementation phases.

Hybrid design

Data warehouse (DW) solutions often resemble the hub and spokes architecture. Legacy systems feeding the DW/BI solution often include customer relationship management (CRM) and enterprise resource planning solutions (ERP), generating large amounts of data. To consolidate these various data models, and facilitate the extract transform load (ETL) process, DW solutions often make use of an operational data store (ODS). The information from the ODS is then parsed into the actual DW. To reduce data redundancy, larger systems will often store the data in a normalized way. Data marts for specific reports can then be built on top of the DW solution.

It is important to note that the DW database in a hybrid solution is kept on third normal form to eliminate data redundancy. A normal relational database however, is not efficient for business intelligence reports where dimensional modelling is prevalent. Small data marts can shop for data from the consolidated warehouse and use the filtered, specific data for the fact tables and dimensions required. The DW effectively provides a single source of information from which the data marts can read, creating a highly flexible solution from a BI point of view. The hybrid architecture allows a DW to be replaced with a master data management solution where operational, not static information could reside.
The Data Vault Modeling components follow hub and spokes architecture. This modeling style is a hybrid design, consisting of the best of breed practices from both 3rd normal form and star schema. The Data Vault model is not a true 3rd normal form, and breaks some of the rules that 3NF dictates be followed. It is however, a top-down architecture with a bottom up design. The Data Vault model is geared to be strictly a data warehouse. It is not geared to be end-user accessible, which when built, still requires the use of a data mart or star schema based release area for business purposes.

**Data warehouses versus operational systems**

Operational systems are optimized for preservation of data integrity and speed of recording of business transactions through use of database normalization and an entity-relationship model. Operational system designers generally follow the Codd rules of database normalization in order to ensure data integrity. Codd defined five increasingly stringent rules of normalization. Fully normalized database designs (that is, those satisfying all five Codd rules) often result in information from a business transaction being stored in dozens to hundreds of tables. Relational databases are efficient at managing the relationships between these tables. The databases have very fast insert/update performance because only a small amount of data in those tables is affected each time a transaction is processed. Finally, in order to improve performance, older data are usually periodically purged from operational systems.

**Evolution in organization use**

These terms refer to the level of sophistication of a data warehouse:

- **Offline operational data warehouse**
  
  Data warehouses in this stage of evolution are updated on a regular time cycle (usually daily, weekly or monthly) from the operational systems and the data is stored in an integrated reporting-oriented data

- **Offline data warehouse**
  
  Data warehouses at this stage are updated from data in the operational systems on a regular basis and the data warehouse data are stored in a data structure designed to facilitate reporting.

- **On time data warehouse**
  
  Online Integrated Data Warehousing represent the real time Data warehouses stage data in the warehouse is updated for every transaction performed on the source data

- **Integrated data warehouse**
  
  These data warehouses assemble data from different areas of business, so users can look up the information they need across other systems.\(^{[12]}\)

**Sample applications**

Some of the applications data warehousing can be used for are:

- Agriculture\(^{[13]}\)
- Call record analysis
- Churn Prediction for Telecom subscribers, Credit Card users etc.
- Decision support
- Financial forecasting
- Insurance fraud analysis
- Logistics and Inventory management
- Trend analysis
References


Further reading


External links

- Ralph Kimball articles (http://www.kimballgroup.com/html/articles.html)
Extract, transform, load

Extract, transform and load (ETL) is a process in database usage and especially in data warehousing that involves:

• Extracting data from outside sources
• Transforming it to fit operational needs (which can include quality levels)
• Loading it into the end target (database, more specifically, operational data store, data mart or data warehouse)

Extract

The first part of an ETL process involves extracting the data from the source systems. In many cases this is the most challenging aspect of ETL, as extracting data correctly will set the stage for how subsequent processes will go.

Most data warehousing projects consolidate data from different source systems. Each separate system may also use a different data organization/format. Common data source formats are relational databases and flat files, but may include non-relational database structures such as Information Management System (IMS) or other data structures such as Virtual Storage Access Method (VSAM) or Indexed Sequential Access Method (ISAM), or even fetching from outside sources such as through web spidering or screen-scraping. The streaming of the extracted data source and load on-the-fly to the destination database is another way of performing ETL when no intermediate data storage is required. In general, the goal of the extraction phase is to convert the data into a single format which is appropriate for transformation processing.

An intrinsic part of the extraction involves the parsing of extracted data, resulting in a check if the data meets an expected pattern or structure. If not, the data may be rejected entirely or in part.

Transform

The transform stage applies to a series of rules or functions to the extracted data from the source to derive the data for loading into the end target. Some data sources will require very little or even no manipulation of data. In other cases, one or more of the following transformation types may be required to meet the business and technical needs of the target database:

• Selecting only certain columns to load (or selecting null columns not to load). For example, if the source data has three columns (also called attributes), for example roll_no, age, and salary, then the extraction may take only roll_no and salary. Similarly, the extraction mechanism may ignore all those records where salary is not present (salary = null).
• Translating coded values (e.g., if the source system stores 1 for male and 2 for female, but the warehouse stores M for male and F for female)
• Encoding free-form values (e.g., mapping ”Male” to ”1”)
• Deriving a new calculated value (e.g., sale_amount = qty * unit_price)
• Sorting
• Joining data from multiple sources (e.g., lookup, merge) and deduplicating the data
• Aggregation (for example, rollup — summarizing multiple rows of data — total sales for each store, and for each region, etc.)
• Generating surrogate-key values
• Transposing or pivoting (turning multiple columns into multiple rows or vice versa)
• Splitting a column into multiple columns (e.g., putting a comma-separated list specified as a string in one column as individual values in different columns)
• Disaggregation of repeating columns into a separate detail table (e.g., moving a series of addresses in one record into single addresses in a set of records in a linked address table)
• Lookup and validate the relevant data from tables or referential files for slowly changing dimensions.
• Applying any form of simple or complex data validation. If validation fails, it may result in a full, partial or no rejection of the data, and thus none, some or all the data is handed over to the next step, depending on the rule design and exception handling. Many of the above transformations may result in exceptions, for example, when a code translation parses an unknown code in the extracted data.

Load
The load phase loads the data into the end target, usually the data warehouse (DW). Depending on the requirements of the organization, this process varies widely. Some data warehouses may overwrite existing information with cumulative information, frequently updating extract data is done on daily, weekly or monthly basis. Other DW (or even other parts of the same DW) may add new data in a historized form, for example, hourly. To understand this, consider a DW that is required to maintain sales records of the last year. Then, the DW will overwrite any data that is older than a year with newer data. However, the entry of data for any one year window will be made in a historized manner. The timing and scope to replace or append are strategic design choices dependent on the time available and the business needs. More complex systems can maintain a history and audit trail of all changes to the data loaded in the DW.

As the load phase interacts with a database, the constraints defined in the database schema — as well as in triggers activated upon data load — apply (for example, uniqueness, referential integrity, mandatory fields), which also contribute to the overall data quality performance of the ETL process.
• For example, a financial institution might have information on a customer in several departments and each department might have that customer's information listed in a different way. The membership department might list the customer by name, whereas the accounting department might list the customer by number. ETL can bundle all this data and consolidate it into a uniform presentation, such as for storing in a database or data warehouse.
• Another way that companies use ETL is to move information to another application permanently. For instance, the new application might use another database vendor and most likely a very different database schema. ETL can be used to transform the data into a format suitable for the new application to use.
• An example of this would be an Expense and Cost Recovery System (ECRS) such as used by accountancies, consultancies and lawyers. The data usually ends up in the time and billing system, although some businesses may also utilize the raw data for employee productivity reports to Human Resources (personnel dept.) or equipment usage reports to Facilities Management.
### Real-life ETL cycle

The typical real-life ETL cycle consists of the following execution steps:

1. Cycle initiation
2. Build reference data
3. Extract (from sources)
4. Validate
5. Transform (clean, apply business rules, check for data integrity, create aggregates or disaggregates)
6. Stage (load into staging tables, if used)
7. Audit reports (for example, on compliance with business rules. Also, in case of failure, helps to diagnose/repair)
8. Publish (to target tables)
9. Archive
10. Clean up

### Challenges

ETL processes can involve considerable complexity, and significant operational problems can occur with improperly designed ETL systems.

The range of data values or data quality in an operational system may exceed the expectations of designers at the time validation and transformation rules are specified. Data profiling of a source during data analysis can identify the data conditions that will need to be managed by transform rules specifications. This will lead to an amendment of validation rules explicitly and implicitly implemented in the ETL process.

Data warehouses are typically assembled from a variety of data sources with different formats and purposes. As such, ETL is a key process to bring all the data together in a standard, homogeneous environment.

Design analysts should establish the scalability of an ETL system across the lifetime of its usage. This includes understanding the volumes of data that will have to be processed within service level agreements. The time available to extract from source systems may change, which may mean the same amount of data may have to be processed in less time. Some ETL systems have to scale to process terabytes of data to update data warehouses with tens of terabytes of data. Increasing volumes of data may require designs that can scale from daily batch to multiple-day microbatch to integration with message queues or real-time change-data capture for continuous transformation and update.

### Performance

ETL vendors benchmark their record-systems at multiple TB (terabytes) per hour (or ~1 GB per second) using powerful servers with multiple CPUs, multiple hard drives, multiple gigabit-network connections, and lots of memory. The fastest ETL record is currently held by Syncsort\(^1\), Vertica and HP at 5.4TB in under an hour which is more than twice as fast as the earlier record held by Microsoft and Unisys.

In real life, the slowest part of an ETL process usually occurs in the database load phase. Databases may perform slowly because they have to take care of concurrency, integrity maintenance, and indices. Thus, for better performance, it may make sense to employ:

- Direct Path Extract method or bulk unload whenever is possible (instead of querying the database) to reduce the load on source system while getting high speed extract
- most of the transformation processing outside of the database
- bulk load operations whenever possible.

Still, even using bulk operations, database access is usually the bottleneck in the ETL process. Some common methods used to increase performance are:
• Partition tables (and indices). Try to keep partitions similar in size (watch for null values which can skew the partitioning).
• Do all validation in the ETL layer before the load. Disable integrity checking (disable constraint ...) in the target database tables during the load.
• Disable triggers (disable trigger ...) in the target database tables during the load. Simulate their effect as a separate step.
• Generate IDs in the ETL layer (not in the database).
• Drop the indices (on a table or partition) before the load - and recreate them after the load (SQL: drop index ...; create index ...).
• Use parallel bulk load when possible — works well when the table is partitioned or there are no indices. Note: attempt to do parallel loads into the same table (partition) usually causes locks — if not on the data rows, then on indices.
• If a requirement exists to do insertions, updates, or deletions, find out which rows should be processed in which way in the ETL layer, and then process these three operations in the database separately. You often can do bulk load for inserts, but updates and deletes commonly go through an API (using SQL).

Whether to do certain operations in the database or outside may involve a trade-off. For example, removing duplicates using distinct may be slow in the database; thus, it makes sense to do it outside. On the other side, if using distinct will significantly (x100) decrease the number of rows to be extracted, then it makes sense to remove duplications as early as possible in the database before unloading data.

A common source of problems in ETL is a big number of dependencies among ETL jobs. For example, job "B" cannot start while job "A" is not finished. You can usually achieve better performance by visualizing all processes on a graph, and trying to reduce the graph making maximum use of parallelism, and making "chains" of consecutive processing as short as possible. Again, partitioning of big tables and of their indices can really help.

Another common issue occurs when the data is spread between several databases, and processing is done in those databases sequentially. Sometimes database replication may be involved as a method of copying data between databases - and this can significantly slow down the whole process. The common solution is to reduce the processing graph to only three layers:

• Sources
• Central ETL layer
• Targets

This allows processing to take maximum advantage of parallel processing. For example, if you need to load data into two databases, you can run the loads in parallel (instead of loading into 1st - and then replicating into the 2nd).

Of course, sometimes processing must take place sequentially. For example, you usually need to get dimensional (reference) data before you can get and validate the rows for main "fact" tables.

Parallel processing

A recent development in ETL software is the implementation of parallel processing. This has enabled a number of methods to improve overall performance of ETL processes when dealing with large volumes of data.

ETL applications implement three main types of parallelism:

• Data: By splitting a single sequential file into smaller data files to provide parallel access.
• Pipeline: Allowing the simultaneous running of several components on the same data stream. For example: looking up a value on record 1 at the same time as adding two fields on record 2.
• Component: The simultaneous running of multiple processes on different data streams in the same job, for example, sorting one input file while removing duplicates on another file.

All three types of parallelism usually operate combined in a single job.
An additional difficulty comes with making sure that the data being uploaded is relatively consistent. Because multiple source databases may have different update cycles (some may be updated every few minutes, while others may take days or weeks), an ETL system may be required to hold back certain data until all sources are synchronized. Likewise, where a warehouse may have to be reconciled to the contents in a source system or with the general ledger, establishing synchronization and reconciliation points becomes necessary.

**Rerunnability, recoverability**

Data warehousing procedures usually subdivide a big ETL process into smaller pieces running sequentially or in parallel. To keep track of data flows, it makes sense to tag each data row with "row_id", and tag each piece of the process with "run_id". In case of a failure, having these IDs will help to roll back and rerun the failed piece.

Best practice also calls for "checkpoints", which are states when certain phases of the process are completed. Once at a checkpoint, it is a good idea to write everything to disk, clean out some temporary files, log the state, and so on.

**Virtual ETL**

As of 2010 data virtualization had begun to advance ETL processing. The application of data virtualization to ETL allowed solving the most common ETL tasks of data migration and application integration for multiple dispersed data sources. So-called Virtual ETL operates with the abstracted representation of the objects or entities gathered from the variety of relational, semi-structured and unstructured data sources. ETL tools can leverage object-oriented modeling and work with entities' representations persistently stored in a centrally located hub-and-spoke architecture. Such a collection that contains representations of the entities or objects gathered from the data sources for ETL processing is called a metadata repository and it can reside in memory\(^2\) or be made persistent. By using a persistent metadata repository, ETL tools can transition from one-time projects to persistent middleware, performing data harmonization and data profiling consistently and in near-real time.

**Best practices**

**Four-layered approach for ETL architecture design**

- Functional layer: Core functional ETL processing (extract, transform, and load).
- Operational management layer: Job-stream definition and management, parameters, scheduling, monitoring, communication and alerting.
- Audit, balance and control (ABC) layer: Job-execution statistics, balancing and controls, rejects- and error-handling, codes management.
- Utility layer: Common components supporting all other layers.

**Use file-based ETL processing where possible**

- Storage costs relatively little
- Intermediate files serve multiple purposes:
  - Used for testing and debugging
  - Used for restart and recover processing
  - Used to calculate control statistics
- Helps to reduce dependencies - enables modular programming.
- Allows flexibility for job execution and scheduling
- Better performance if coded properly, and can take advantage of parallel processing capabilities when the need arises.

**Use data-driven methods and minimize custom ETL coding**

- Parameter-driven jobs, functions, and job-control
• Code definitions and mapping in database
• Consideration for data-driven tables to support more complex code-mappings and business-rule application.

**Qualities of a good ETL architecture design**

• Performance
• Scalable
• Migratable
• Recoverable (run_id, ...)
• Operable (completion-codes for phases, re-running from checkpoints, etc.)
• Auditable (in two dimensions: business requirements and technical troubleshooting)

**Handling of non-desirable values (NULL values, erroneous values, etc.)**

See: Dealing With Nulls In The Dimensional Model (Kimball University)\(^3\)

• NULL DIMENSIONAL values
• NULL FACT values
• NULL PRIMARY and/or FOREIGN KEY values
• Erroneous or undesirable values

**Dealing with keys**

Keys are some of the most important objects in all relational databases as they tie everything together. A primary key is a column which is the identifier for a given entity, where a foreign key is a column in another table which refers a primary key. These keys can also be made up from several columns, in which case they are composite keys. In many cases the primary key is an auto generated integer which has no meaning for the business entity being represented, but solely exists for the purpose of the relational database - commonly referred to as a surrogate key.

As there will usually be more than one datasource being loaded into the warehouse the keys are an important concern to be addressed.

Your customers might be represented in several data sources, and in one their SSN (Social Security Number) might be the primary key, their phone number in another and a surrogate in the third. All of the customers information needs to be consolidated into one dimension table.

A recommended way to deal with the concern is to add a warehouse surrogate key, which will be used as foreign key from the fact table.\(^4\)

Usually updates will occur to a dimension's source data, which obviously must be reflected in the data warehouse. If the primary key of the source data is required for reporting, the dimension already contains that piece of information for each row. If the source data uses a surrogate key, the warehouse must keep track of it even though it is never used in queries or reports.

That is done by creating a lookup table which contains the warehouse surrogate key and the originating key.\(^5\) This way the dimension is not polluted with surrogates from various source systems, while the ability to update is preserved.

The lookup table is used in different ways depending on the nature of the source data. There are 5 types to consider,\(^6\) where three selected ones are included here:

**Type 1:**
- The dimension row is simply updated to match the current state of the source system. The warehouse does not capture history. The lookup table is used to identify which dimension row to update/overwrite.

**Type 2:**
- A new dimension row is added with the new state of the source system. A new surrogate key is assigned. Source key is no longer unique in the lookup table.

**Fully logged:**
- A new dimension row is added with the new state of the source system, while the previous dimension row is updated to reflect it is no longer active and record time of deactivation.

**Tools**

Programmers can set up ETL processes using almost any programming language, but building such processes from scratch can become complex. Increasingly, companies are buying ETL tools to help in the creation of ETL processes.\[7\]

By using an established ETL framework, one may increase one's chances of ending up with better connectivity and scalability. A good ETL tool must be able to communicate with the many different relational databases and read the various file formats used throughout an organization. ETL tools have started to migrate into Enterprise Application Integration, or even Enterprise Service Bus, systems that now cover much more than just the extraction, transformation, and loading of data. Many ETL vendors now have data profiling, data quality, and metadata capabilities. A common use case for ETL tools include converting CSV files to formats readable by relational databases. A typical translation of millions of records is facilitated by ETL tools that enable users to input csv-like data feeds/files and import it into a database with as little code as possible.

ETL Tools are typically used by a broad range of professionals - from students in computer science looking to quickly import large data sets to database architects in charge of company account management, ETL Tools have become a convenient tool that can be relied on to get maximum performance. ETL tools in most cases contain a GUI that helps users conveniently transform data as opposed to writing large programs to parse files and modify data types - which ETL tools facilitate as much as possible.

- Informatica Powercenter
- SQL Server Integration Services (included in Microsoft SQL Server product line)
- Oracle Data Integrator (earlier owned by Sunopsis)
- Pervasive Software
- Relational Junction
- Safe Software
- **SAP BusinessObjects Data Integrator** (known as **BusinessObjects Data Integrator** before the acquisition of BusinessObjects by SAP corporation)
- **SAS Data Integration Server** (in the earlier versions known as **SAS ETL Studio** (version 8) or **SAS Data Integration Studio** (version 9)
- Stone Bond Technologies
- SnapLogic;
- Syncsort DMExpress - High Performance ETL

**Open Source / Dual-licensed**

- Pentaho
- Talend Open Studio
- Scriptella
- JasperETL from JasperSoft
- CloverETL
- Benetl
- Toolsverse ETL Framework
References

[1] "New ETL World Record: 5.4 TB Loaded in Under 1 Hour - Syncsort" (http://www.syncsort.com/Portals/0/Resources/Solution/DMX_Solution_WorldRecord.pdf)
[4] (Kimball, The Data Warehouse Lifecycle Toolkit, p 332)


Star schema

In computing, the star schema (also called star-join schema, data cube, or multi-dimensional schema) is the simplest style of data warehouse schema. The star schema consists of one or more fact tables referencing any number of dimension tables. The star schema is an important special case of the snowflake schema, and is more effective for handling simpler queries. [1]

Model

A star schema classifies the attributes of an event into facts (measured numeric/time data), and descriptive dimension attributes (product id, customer name, sale date) that give the facts a context. A fact record is the nexus between the specific dimension values and the recorded facts. The facts are stored at a uniform level of detail (the grain) in the fact table. Dimension attributes are organized into affinity groups and stored in a minimal number of dimension tables.

A weather star schema that records weather data may have facts of temperature, barometric pressure, wind speed, precipitation, cloud cover, etc. and dimensions of location, date/time, reporter, etc.

Star schemas are designed to optimize user ease-of-use and retrieval performance by minimizing the number of tables to join to materialize a transaction.

A star schema is called such as it a constellation of stars, generally several bright stars (facts) surrounded by dimmer ones (dimensions).

• The fact table holds the metric values recorded for a specific event. Because of the desire to hold atomic level data, there generally are a very large number of records (billions). Special care is taken to minimize the number and size of attributes in order to constrain the overall table size and maintain performance. Fact tables generally come in 3 flavors - transaction (facts about a specific event e.g. Sale), snapshot (facts recorded at a point in time e.g. Account details at month end), and accumulating snapshot tables (e.g. month-to-date sales for a product).

• Dimension tables usually have few records compared to fact tables, but may have a very large number of attributes that describe the fact data.

Often there can be dozens to hundreds of dimension attributes describing the various facets of a fact. Dimension attributes are organized into tables of loosely related attributes that share a known or unknown affinity. Attributes of color, style, size, texture can describe a product and would be included in a product dimension table. Dimension tables include attributes that typically would be normalized into separate tables (Snowflake schema). For example, in the US a location can be identified by a zipcode that exists within a neighborhood, city, state, region. All of these
attributes would be included in a location dimension table.

On an Entity-Relationship (ER) diagram, fact tables have few distinct columns, while dimension tables have a large number of columns. However, most of the storage is used by the fact table.

Dimension tables are assigned a surrogate primary key of a simple integer that is assigned to the combination of low level attributes that form the natural key. Fact tables should also have a single surrogate primary key to allow for situations where there may be two or more facts having exactly the same set of dimension keys.

A star schema that has many dimensions is sometimes called a centipede schema.[2] Having dimensions of only a few attributes, while simpler to maintain, results in queries with many table joins and makes the star schema less easy to use.

Benefits

The primary benefit of a star schema is its simplicity for users to write, and databases to process: queries are written with simple inner joins between the facts and a small number of dimensions. Star joins are simpler than possible in snowflake schema. WHERE conditions need only to filter on the attributes desired, and aggregations are fast.

The star schema is a way to implement multidimensional database (MDDB) functionality using a mainstream relational database.

Example

Consider a database of sales, perhaps from a store chain, classified by date, store and product. The image of the schema to the right is a star schema version of the sample schema provided in the snowflake schema article.

Fact_Sales is the fact table and there are three dimension tables Dim_Date, Dim_Store and Dim_Product.

Each dimension table has a primary key on its Id column, relating to one of the columns (viewed as rows in the example schema) of the Fact_Sales table's three-column (compound) primary key (Date_Id, Store_Id, Product_Id). The non-primary key Units_Sold column of the fact table in this example represents a measure or metric that can be used in calculations and analysis. The non-primary key columns of the dimension tables represent additional attributes of the dimensions (such as the Year of the Dim_Date dimension).

For example, the following query answers how many TV sets have been sold, for each brand and country, in 1997:

```sql
SELECT
    P.Brand,
    S.Country,
    SUM(F.Units_Sold)
FROM
    Fact_Sales F
INNER JOIN Dim_Date D ON F.Date_Id = D.Id
INNER JOIN Dim_Store S ON F.Store_Id = S.Id
INNER JOIN Dim_Product P ON F.Product_Id = P.Id
```

Star schema used by example query.
WHERE
  D.Year = 1997
AND P.Product_Category = 'tv'
GROUP BY
  P.Brand,
  S.Country

References

External links
• Designing the Star Schema Database by Craig Utley (http://ciobriefings.com/Publications/WhitePapers/DesigningtheStarSchemaDatabase/tabid/101/Default.aspx)
• Star Schema for Retail Sales (http://opensourceanalytics.com/2006/04/28/sales-data-mart-dimensional-model-for-retail/)
• Stars: A Pattern Language for Query Optimized Schema (http://c2.com/ppr/stars.html)
• Star schema optimizations (http://www.dwoptimize.com/2007/06/aiming-for-stars.html)
• Fact constellation schema (http://datawarehouse4u.info/Data-warehouse-schema-architecture-fact-constellation-schema.html)
• Data Warehouses, Schemas and Decision Support Basics by Dan Power (http://www.b-eye-network.com/view/8451)

Snowflake schema

In computing, a snowflake schema is a logical arrangement of tables in a multidimensional database such that the entity relationship diagram resembles a snowflake in shape. The snowflake schema is represented by centralized fact tables which are connected to multiple dimensions.

The snowflake schema is similar to the star schema. However, in the snowflake schema, dimensions are normalized into multiple related tables, whereas the star schema’s dimensions are normalized with each dimension represented by a single table. A complex snowflake shape emerges when the dimensions of a snowflake schema are elaborate, having multiple levels of relationships, and the child tables have multiple parent tables (“forks in the road”). The “snowflaking” effect only affects the dimension tables and NOT the fact tables.
**Common uses**

Star and snowflake schemas are most commonly found in dimensional data warehouses and data marts where speed of data retrieval is more important than the efficiency of data manipulations. As such, the tables in these schemas are not normalized much, and are frequently designed at a level of normalization short of third normal form.

Deciding whether to employ a star schema or a snowflake schema should involve considering the relative strengths of the database platform in question and the query tool to be employed. Star schemas should be favored with query tools that largely expose users to the underlying table structures, and in environments where most queries are simpler in nature. Snowflake schemas are often better with more sophisticated query tools that create a layer of abstraction between the users and raw table structures for environments having numerous queries with complex criteria.

**Data normalization and storage**

Normalization splits up data to avoid redundancy (duplication) by moving commonly repeating groups of data into new tables. Normalization therefore tends to increase the number of tables that need to be joined in order to perform a given query, but reduces the space required to hold the data and the number of places where it needs to be updated if the data changes.

From a space storage point of view, the dimensional tables are typically small compared to the fact tables. This often removes the storage space benefit of snowflaking the dimension tables, as compared with a star schema.

Some database developers compromise by creating an underlying snowflake schema with views built on top of it that perform many of the necessary joins to simulate a star schema. This provides the storage benefits achieved through the normalization of dimensions with the ease of querying that the star schema provides. The tradeoff is that requiring the server to perform the underlying joins automatically can result in a performance hit when querying as well as extra joins to tables that may not be necessary to fulfill certain queries.

**Benefits of "snowballings"**

- Some OLAP multidimensional database modeling tools that use dimensional data marts as data sources are optimized for snowflake schemas.
- If a dimension is very sparse (i.e. most of the possible values for the dimension have no data) and/or a dimension has a very long list of attributes which may be used in a query, the dimension table may occupy a significant proportion of the database and snowflaking may be appropriate.
- A multidimensional view is sometimes added to an existing transactional database to aid reporting. In this case, the tables which describe the dimensions will already exist and will typically be normalized. A snowflake schema will therefore be easier to implement.
- A snowflake schema can sometimes reflect the way in which users think about data. Users may prefer to generate queries using a star schema in some cases, although this may or may not be reflected in the underlying organization of the database.
- Some users may wish to submit queries to the database which, using conventional multidimensional reporting tools, cannot be expressed within a simple star schema. This is particularly common in data mining of customer databases, where a common requirement is to locate common factors between customers who bought products meeting complex criteria. Some snowflaking would typically be required to permit simple query tools to form such a query, especially if provision for these forms of query weren't anticipated when the data warehouse was first designed.
Examples

The example schema shown to the right is a snowflaked version of the star schema example provided in the star schema article.

The following example query is the snowflake schema equivalent of the star schema example code which returns the total number of units sold by brand and by country for 1997. Notice that the snowflake schema query requires many more joins than the star schema version in order to fulfill even a simple query. The benefit of using the snowflake schema in this example is that the storage requirements are lower since the snowflake schema eliminates many duplicate values from the dimensions themselves.

```
SELECT
    B.Brand,
    G.Country,
    SUM(F.Units_Sold)
FROM
    Fact_Sales F
INNER JOIN Dim_Date D ON F.Date_Id = D.Id
INNER JOIN Dim_Store S ON F.Store_Id = S.Id
INNER JOIN Dim_Geography G ON S.Geography_Id = G.Id
INNER JOIN Dim_Product P ON F.Product_Id = P.Id
INNER JOIN Dim_Brand B ON P.Brand_Id = B.Id
INNER JOIN Dim_Product_Category C ON P.Product_Category_Id = C.Id
WHERE
    D.Year = 1997 AND C.Product_Category = 'tv'
GROUP BY
    B.Brand,
    G.Country
```

References


External links

- "Why is the Snowflake Schema a Good Data Warehouse Design?" by Mark Levene and George Loizou
- Reverse Snowflake Joins
Data mining

Data mining (the analysis step of the "Knowledge Discovery in Databases" process, or KDD),[1] is a field at the intersection of computer science and statistics,[2][3][4] is the process that attempts to discover patterns in large data sets. It utilizes methods at the intersection of artificial intelligence, machine learning, statistics, and database systems.[2] The overall goal of the data mining process is to extract information from a data set and transform it into an understandable structure for further use.[2] Aside from the raw analysis step, it involves database and data management aspects, data preprocessing, model and inference considerations, interestingness metrics, complexity considerations, post-processing of discovered structures, visualization, and online updating.[2]

The term is a buzzword, and is frequently misused to mean any form of large-scale data or information processing (collection, extraction, warehousing, analysis, and statistics) but is also generalized to any kind of computer decision support system, including artificial intelligence, machine learning, and business intelligence. In the proper use of the word, the key term is discovery, commonly defined as "detecting something new". Even the popular book "Data mining: Practical machine learning tools and techniques with Java"[5] (which covers mostly machine learning material) was originally to be named just "Practical machine learning", and the term "data mining" was only added for marketing reasons.[6] Often the more general terms "(large scale) data analysis", or "analytics" – or when referring to actual methods, artificial intelligence and machine learning – are more appropriate.

The actual data mining task is the automatic or semi-automatic analysis of large quantities of data to extract previously unknown interesting patterns such as groups of data records (cluster analysis), unusual records (anomaly detection) and dependencies (association rule mining). This usually involves using database techniques such as spatial indexes. These patterns can then be seen as a kind of summary of the input data, and may be used in further analysis or, for example, in machine learning and predictive analytics. For example, the data mining step might identify multiple groups in the data, which can then be used to obtain more accurate prediction results by a decision support system. Neither the data collection, data preparation, nor result interpretation and reporting are part of the data mining step, but do belong to the overall KDD process as additional steps.

The related terms data dredging, data fishing, and data snooping refer to the use of data mining methods to sample parts of a larger population data set that are (or may be) too small for reliable statistical inferences to be made about the validity of any patterns discovered. These methods can, however, be used in creating new hypotheses to test against the larger data populations.

Etymology

In the 1960s, statisticians used terms like "Data Fishing" or "Data Dredging" to refer to what they considered the bad practice of analyzing data without an a-priori hypothesis. The term "Data Mining" appeared around 1990 in the database community. At the beginning of the century, there was a phrase "database mining™, trademarked by HNC, a San Diego-based company (now merged into FICO), to pitch their Data Mining Workstation,[7] researchers consequently turned to "data mining". Other terms used include Data Archaeology, Information Harvesting, Information Discovery, Knowledge Extraction, etc. Gregory Piatetsky-Shapiro coined the term "Knowledge Discovery in Databases" for the first workshop on the same topic (1989) and this term became more popular in AI and Machine Learning Community. However, the term data mining became more popular in the business and press communities.[8] Currently, Data Mining and Knowledge Discovery are used interchangeably.
Data mining

Background

The manual extraction of patterns from data has occurred for centuries. Early methods of identifying patterns in data include Bayes' theorem (1700s) and regression analysis (1800s). The proliferation, ubiquity and increasing power of computer technology has dramatically increased data collection, storage, and manipulation ability. As data sets have grown in size and complexity, direct "hands-on" data analysis has increasingly been augmented with indirect, automated data processing, aided by other discoveries in computer science, such as neural networks, cluster analysis, genetic algorithms (1950s), decision trees (1960s), and support vector machines (1990s). Data mining is the process of applying these methods with the intention of uncovering hidden patterns\[9\] in large data sets. It bridges the gap from applied statistics and artificial intelligence (which usually provide the mathematical background) to database management by exploiting the way data is stored and indexed in databases to execute the actual learning and discovery algorithms more efficiently, allowing such methods to be applied to ever larger data sets.

Research and evolution

The premier professional body in the field is the Association for Computing Machinery's Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD). Since 1989 they have hosted an annual international conference and published its proceedings,\[10\] and since 1999 have published a biannual academic journal titled "SIGKDD Explorations".\[11\]

Computer science conferences on data mining include:

- CIKM Conference – ACM Conference on Information and Knowledge Management
-DMIN Conference – International Conference on Data Mining
- DMKD Conference – Research Issues on Data Mining and Knowledge Discovery
- ECDM Conference – European Conference on Data Mining
- ECML-PKDD Conference – European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases
- EDM Conference – International Conference on Educational Data Mining
- ICDM Conference – IEEE International Conference on Data Mining
- KDD Conference – ACM SIGKDD Conference on Knowledge Discovery and Data Mining
- MLDM Conference – Machine Learning and Data Mining in Pattern Recognition
- PAKDD Conference – The annual Pacific-Asia Conference on Knowledge Discovery and Data Mining
- PAW Event – Predicitalytics World
- SDM Conference – SIAM International Conference on Data Mining (SIAM)
- SSTD Symposium – Symposium on Spatial and Temporal Databases
- WSDM Conference – ACM Conference on Web Search and Data Mining

Data mining topics are also present on many data management/database conferences such as the ICDE Conference, SIGMOD Conference and International Conference on Very Large Data Bases
Process

The **Knowledge Discovery in Databases (KDD) process** is commonly defined with the stages:

1. Selection
2. Pre-processing
3. Transformation
4. **Data Mining**
5. Interpretation/Evaluation.\(^1\)

It exists, however, in many variations on this theme, such as the Cross Industry Standard Process for Data Mining (CRISP-DM) which defines six phases:

1. Business Understanding
2. Data Understanding
3. Data Preparation
4. Modeling
5. Evaluation
6. Deployment

or a simplified process such as (1) pre-processing, (2) data mining, and (3) results validation.

Pre-processing

Before data mining algorithms can be used, a target data set must be assembled. As data mining can only uncover patterns actually present in the data, the target dataset must be large enough to contain these patterns while remaining concise enough to be mined within an acceptable time limit. A common source for data is a data mart or data warehouse. Pre-processing is essential to analyze the multivariate datasets before data mining. The target set is then cleaned. Data cleaning removes the observations containing noise and those with missing data.

Data mining

Data mining involves six common classes of tasks:\(^1\)

- Anomaly detection (Outlier/change/deviation detection) – The identification of unusual data records, that might be interesting or data errors and require further investigation.
- Association rule learning (Dependency modeling) – Searches for relationships between variables. For example a supermarket might gather data on customer purchasing habits. Using association rule learning, the supermarket can determine which products are frequently bought together and use this information for marketing purposes. This is sometimes referred to as market basket analysis.
- Clustering – is the task of discovering groups and structures in the data that are in some way or another "similar", without using known structures in the data.
- Classification – is the task of generalizing known structure to apply to new data. For example, an e-mail program might attempt to classify an e-mail as "legitimate" or as "spam".
- Regression – Attempts to find a function which models the data with the least error.
- Summarization – providing a more compact representation of the data set, including visualization and report generation.
Results validation

The final step of knowledge discovery from data is to verify that the patterns produced by the data mining algorithms occur in the wider data set. Not all patterns found by the data mining algorithms are necessarily valid. It is common for the data mining algorithms to find patterns in the training set which are not present in the general data set. This is called overfitting. To overcome this, the evaluation uses a test set of data on which the data mining algorithm was not trained. The learned patterns are applied to this test set and the resulting output is compared to the desired output. For example, a data mining algorithm trying to distinguish "spam" from "legitimate" emails would be trained on a training set of sample e-mails. Once trained, the learned patterns would be applied to the test set of e-mails on which it had not been trained. The accuracy of the patterns can then be measured from how many e-mails they correctly classify. A number of statistical methods may be used to evaluate the algorithm, such as ROC curves.

If the learned patterns do not meet the desired standards, then it is necessary to re-evaluate and change the pre-processing and data mining steps. If the learned patterns do meet the desired standards, then the final step is to interpret the learned patterns and turn them into knowledge.

Standards

There have been some efforts to define standards for the data mining process, for example the 1999 European Cross Industry Standard Process for Data Mining (CRISP-DM 1.0) and the 2004 Java Data Mining standard (JDM 1.0). Development on successors to these processes (CRISP-DM 2.0 and JDM 2.0) was active in 2006, but has stalled since. JDM 2.0 was withdrawn without reaching a final draft.

For exchanging the extracted models – in particular for use in predictive analytics – the key standard is the Predictive Model Markup Language (PMML), which is an XML-based language developed by the Data Mining Group (DMG) and supported as exchange format by many data mining applications. As the name suggests, it only covers prediction models, a particular data mining task of high importance to business applications. However, extensions to cover (for example) subspace clustering have been proposed independently of the DMG. [12]

Notable uses

Games

Since the early 1960s, with the availability of oracles for certain combinatorial games, also called tablebases (e.g. for 3x3-chess) with any beginning configuration, small-board dots-and-boxes, small-board-hex, and certain endgames in chess, dots-and-boxes, and hex; a new area for data mining has been opened. This is the extraction of human-usable strategies from these oracles. Current pattern recognition approaches do not seem to fully acquire the high level of abstraction required to be applied successfully. Instead, extensive experimentation with the tablebases – combined with an intensive study of tablebase-answers to well designed problems, and with knowledge of prior art (i.e. pre-tablebase knowledge) – is used to yield insightful patterns. Berlekamp (in dots-and-boxes, etc.) and John Nunn (in chess endgames) are notable examples of researchers doing this work, though they were not – and are not – involved in tablebase generation.

Business

Data mining in customer relationship management applications can contribute significantly to the bottom line. Rather than randomly contacting a prospect or customer through a call center or sending mail, a company can concentrate its efforts on prospects that are predicted to have a high likelihood of responding to an offer. More sophisticated methods may be used to optimize resources across campaigns so that one may predict to which channel and to which offer an individual is most likely to respond (across all potential offers). Additionally, sophisticated applications could be used to automate mailing. Once the results from data mining (potential prospect/customer and
channel/offer) are determined, this "sophisticated application" can either automatically send an e-mail or a regular mail. Finally, in cases where many people will take an action without an offer, "uplift modeling" can be used to determine which people have the greatest increase in response if given an offer. Data clustering can also be used to automatically discover the segments or groups within a customer data set.

Businesses employing data mining may see a return on investment, but also they recognize that the number of predictive models can quickly become very large. Rather than using one model to predict how many customers will churn, a business could build a separate model for each region and customer type. Then, instead of sending an offer to all people that are likely to churn, it may only want to send offers to loyal customers. Finally, the business may want to determine which customers are going to be profitable over a certain window in time, and only send the offers to those that are likely to be profitable. In order to maintain this quantity of models, they need to manage model versions and move on to automated data mining.

Data mining can also be helpful to human resources (HR) departments in identifying the characteristics of their most successful employees. Information obtained — such as universities attended by highly successful employees — can help HR focus recruiting efforts accordingly. Additionally, Strategic Enterprise Management applications help a company translate corporate-level goals, such as profit and margin share targets, into operational decisions, such as production plans and workforce levels.[13]

Another example of data mining, often called the market basket analysis, relates to its use in retail sales. If a clothing store records the purchases of customers, a data mining system could identify those customers who favor silk shirts over cotton ones. Although some explanations of relationships may be difficult, taking advantage of it is easier. The example deals with association rules within transaction-based data. Not all data are transaction based and logical, or inexact rules may also be present within a database.

Market basket analysis has also been used to identify the purchase patterns of the Alpha Consumer. Alpha Consumers are people that play a key role in connecting with the concept behind a product, then adopting that product, and finally validating it for the rest of society. Analyzing the data collected on this type of user has allowed companies to predict future buying trends and forecast supply demands.

Data mining is a highly effective tool in the catalog marketing industry. Catalogers have a rich database of history of their customer transactions for millions of customers dating back a number of years. Data mining tools can identify patterns among customers and help identify the most likely customers to respond to upcoming mailing campaigns.

Data mining for business applications is a component which needs to be integrated into a complex modeling and decision making process. Reactive business intelligence (RBI) advocates a "holistic" approach that integrates data mining, modeling, and interactive visualization into an end-to-end discovery and continuous innovation process powered by human and automated learning.[14]

In the area of decision making, the RBI approach has been used to mine knowledge that is progressively acquired from the decision maker, and then self-tune the decision method accordingly.[15]

An example of data mining related to an integrated-circuit production line is described in the paper "Mining IC Test Data to Optimize VLSI Testing."[16] In this paper, the application of data mining and decision analysis to the problem of die-level functional testing is described. Experiments mentioned demonstrate the ability to apply a system of mining historical die-test data to create a probabilistic model of patterns of die failure. These patterns are then utilized to decide, in real time, which die to test next and when to stop testing. This system has been shown, based on experiments with historical test data, to have the potential to improve profits on mature IC products.
Science and engineering

In recent years, data mining has been used widely in the areas of science and engineering, such as bioinformatics, genetics, medicine, education and electrical power engineering.

In the study of human genetics, sequence mining helps address the important goal of understanding the mapping relationship between the inter-individual variations in human DNA sequence and the variability in disease susceptibility. In simple terms, it aims to find out how the changes in an individual's DNA sequence affects the risks of developing common diseases such as cancer, which is of great importance to improving methods of diagnosing, preventing, and treating these diseases. The data mining method that is used to perform this task is known as multifactor dimensionality reduction.\[17\]

In the area of electrical power engineering, data mining methods have been widely used for condition monitoring of high voltage electrical equipment. The purpose of condition monitoring is to obtain valuable information on, for example, the status of the insulation (or other important safety-related parameters). Data clustering techniques – such as the self-organizing map (SOM), have been applied to vibration monitoring and analysis of transformer on-load tap-changers (OLTCS). Using vibration monitoring, it can be observed that each tap change operation generates a signal that contains information about the condition of the tap changer contacts and the drive mechanisms. Obviously, different tap positions will generate different signals. However, there was considerable variability amongst normal condition signals for exactly the same tap position. SOM has been applied to detect abnormal conditions and to hypothesize about the nature of the abnormalities.\[18\]

Data mining methods have also been applied to dissolved gas analysis (DGA) in power transformers. DGA, as a diagnostics for power transformers, has been available for many years. Methods such as SOM has been applied to analyze generated data and to determine trends which are not obvious to the standard DGA ratio methods (such as Duval Triangle).\[18\]

Another example of data mining in science and engineering is found in educational research, where data mining has been used to study the factors leading students to choose to engage in behaviors which reduce their learning,\[19\] and to understand factors influencing university student retention.\[20\] A similar example of social application of data mining is its use in expertise finding systems, whereby descriptors of human expertise are extracted, normalized, and classified so as to facilitate the finding of experts, particularly in scientific and technical fields. In this way, data mining can facilitate institutional memory.

Other examples of application of data mining methods are biomedical data facilitated by domain ontologies,\[21\] mining clinical trial data,\[22\] and traffic analysis using SOM.\[23\]

In adverse drug reaction surveillance, the Uppsala Monitoring Centre has, since 1998, used data mining methods to routinely screen for reporting patterns indicative of emerging drug safety issues in the WHO global database of 4.6 million suspected adverse drug reaction incidents.\[24\] Recently, similar methodology has been developed to mine large collections of electronic health records for temporal patterns associating drug prescriptions to medical diagnoses.\[25\]

Data mining has been applied software artifacts within the realm of software engineering: Mining Software Repositories.
Human rights
Data mining of government records — particularly records of the justice system (i.e. courts, prisons) — enables the discovery of systemic human rights violations in connection to generation and publication of invalid or fraudulent legal records by various government agencies.[26][27]

Medical data mining
In 2011, the case of Sorrell v. IMS Health, Inc., decided by the Supreme Court of the United States, ruled that Pharmacies may share information with outside companies. This practice was authorized under the 1st Amendment of the Constitution, protecting the "freedom of speech."[28]

Spatial data mining
Spatial data mining is the application of data mining methods to spatial data. The end objective of spatial data mining is to find patterns in data with respect to geography. So far, data mining and Geographic Information Systems (GIS) have existed as two separate technologies, each with its own methods, traditions, and approaches to visualization and data analysis. Particularly, most contemporary GIS have only very basic spatial analysis functionality. The immense explosion in geographically referenced data occasioned by developments in IT, digital mapping, remote sensing, and the global diffusion of GIS emphasizes the importance of developing data-driven inductive approaches to geographical analysis and modeling.

Data mining offers great potential benefits for GIS-based applied decision-making. Recently, the task of integrating these two technologies has become of critical importance, especially as various public and private sector organizations possessing huge databases with thematic and geographically referenced data begin to realize the huge potential of the information contained therein. Among those organizations are:

- offices requiring analysis or dissemination of geo-referenced statistical data
- public health services searching for explanations of disease clustering
- environmental agencies assessing the impact of changing land-use patterns on climate change
- geo-marketing companies doing customer segmentation based on spatial location.

Challenges
Geospatial data repositories tend to be very large. Moreover, existing GIS datasets are often splintered into feature and attribute components that are conventionally archived in hybrid data management systems. Algorithmic requirements differ substantially for relational (attribute) data management and for topological (feature) data management.[29] Related to this is the range and diversity of geographic data formats, which present unique challenges. The digital geographic data revolution is creating new types of data formats beyond the traditional "vector" and "raster" formats. Geographic data repositories increasingly include ill-structured data, such as imagery and geo-referenced multi-media.[30]

There are several critical research challenges in geographic knowledge discovery and data mining. Miller and Han[31] offer the following list of emerging research topics in the field:

- **Developing and supporting geographic data warehouses (GDW’s):** Spatial properties are often reduced to simple aspatial attributes in mainstream data warehouses. Creating an integrated GDW requires solving issues of spatial and temporal data interoperability — including differences in semantics, referencing systems, geometry, accuracy, and position.

- **Better spatio-temporal representations in geographic knowledge discovery:** Current geographic knowledge discovery (GKD) methods generally use very simple representations of geographic objects and spatial relationships. Geographic data mining methods should recognize more complex geographic objects (i.e. lines and polygons) and relationships (i.e. non-Euclidean distances, direction, connectivity, and interaction through
attributed geographic space such as terrain). Furthermore, the time dimension needs to be more fully integrated into these geographic representations and relationships.

- **Geographic knowledge discovery using diverse data types**: GKD methods should be developed that can handle diverse data types beyond the traditional raster and vector models, including imagery and geo-referenced multimedia, as well as dynamic data types (video streams, animation).

**Sensor data mining**

Wireless sensor networks can be used for facilitating the collection of data for spatial data mining for a variety of applications such as air pollution monitoring. A characteristic of such networks is that nearby sensor nodes monitoring an environmental feature typically register similar values. This kind of data redundancy due to the spatial correlation between sensor observations inspires the techniques for in-network data aggregation and mining. By measuring the spatial correlation between data sampled by different sensors, a wide class of specialized algorithms can be developed to develop more efficient spatial data mining algorithms.

**Visual data mining**

In the process of turning from analogical into digital, large data sets have been generated, collected, and stored discovering statistical patterns, trends and information which is hidden in data, in order to build predictive patterns. Studies suggest visual data mining is faster and much more intuitive than is traditional data mining.

**Music data mining**

Data mining techniques, and in particular co-occurrence analysis, has been used to discover relevant similarities among music corpora (radio lists, CD databases) for the purpose of classifying music into genres in a more objective manner.

**Surveillance**

Data mining has been used to stop terrorist programs under the U.S. government, including the Total Information Awareness (TIA) program, Secure Flight (formerly known as Computer-Assisted Passenger Prescreening System (CAPPS II)), Analysis, Dissemination, Visualization, Insight, Semantic Enhancement (ADVISE), and the Multi-state Anti-Terrorism Information Exchange (MATRIX). These programs have been discontinued due to controversy over whether they violate the 4th Amendment to the United States Constitution, although many programs that were formed under them continue to be funded by different organizations or under different names.

In the context of combating terrorism, two particularly plausible methods of data mining are "pattern mining" and "subject-based data mining".

**Pattern mining**

"Pattern mining" is a data mining method that involves finding existing patterns in data. In this context patterns often means association rules. The original motivation for searching association rules came from the desire to analyze supermarket transaction data, that is, to examine customer behavior in terms of the purchased products. For example, an association rule "beer ⇒ potato chips (80%)" states that four out of five customers that bought beer also bought potato chips.

In the context of pattern mining as a tool to identify terrorist activity, the National Research Council provides the following definition: "Pattern-based data mining looks for patterns (including anomalous data patterns) that might be associated with terrorist activity — these patterns might be regarded as small signals in a large ocean of noise." Pattern Mining includes new areas such a Music Information Retrieval (MIR) where patterns seen both in the temporal and non temporal domains are imported to classical knowledge discovery search methods.
Subject-based data mining

"Subject-based data mining" is a data mining method involving the search for associations between individuals in data. In the context of combating terrorism, the National Research Council provides the following definition: "Subject-based data mining uses an initiating individual or other datum that is considered, based on other information, to be of high interest, and the goal is to determine what other persons or financial transactions or movements, etc., are related to that initiating datum."[41]

Knowledge grid

Knowledge discovery "On the Grid" generally refers to conducting knowledge discovery in an open environment using grid computing concepts, allowing users to integrate data from various online data sources, as well make use of remote resources, for executing their data mining tasks. The earliest example was the Discovery Net,[43][44] developed at Imperial College London, which won the “Most Innovative Data-Intensive Application Award” at the ACM SC02 (Supercomputing 2002) conference and exhibition, based on a demonstration of a fully interactive distributed knowledge discovery application for a bioinformatics application. Other examples include work conducted by researchers at the University of Calabria, who developed a Knowledge Grid architecture for distributed knowledge discovery, based on grid computing.[45][46]

Reliability/Validity

Data mining can be misused, and can also unintentionally produce results which appear significant but which do not actually predict future behavior and cannot be reproduced on a new sample of data. See Data snooping, Data dredging.

Challenges

In four annual surveys of data miners,[47] data mining practitioners consistently identify three key challenges that they face more than any others, specifically (a) dirty data, (b) explaining data mining to others, and (c) unavailability of data/difficult access to data. In the 2010 survey data miners also shared their experiences in overcoming these particular challenges.[48]

Privacy concerns and ethics

Some people believe that data mining itself is ethically neutral.[49] It is important to note that the term "data mining" has no ethical implications, but is often associated with the mining of information in relation to peoples' behavior (ethical and otherwise). To be precise, data mining is a statistical method that is applied to a set of information (i.e. a data set). Associating these data sets with people is an extreme narrowing of the types of data that are available in today's technological society. Examples could range from a set of crash test data for passenger vehicles, to the performance of a group of stocks. These types of data sets make up a great proportion of the information available to be acted on by data mining methods, and rarely have ethical concerns associated with them. However, the ways in which data mining can be used can in some cases and contexts raise questions regarding privacy, legality, and ethics.[50] In particular, data mining government or commercial data sets for national security or law enforcement purposes, such as in the Total Information Awareness Program or in ADVISE, has raised privacy concerns.[51][52]

Data mining requires data preparation which can uncover information or patterns which may compromise confidentiality and privacy obligations. A common way for this to occur is through data aggregation. Data aggregation involves combining data together (possibly from various sources) in a way that facilitates analysis (but that also might make identification of private, individual-level data deductive or otherwise apparent).[53] This is not data mining per se, but a result of the preparation of data before – and for the purposes of – the analysis. The threat to an individual's privacy comes into play when the data, once compiled, cause the data miner, or anyone who has
access to the newly compiled data set, to be able to identify specific individuals, especially when the data were originally anonymous.

It is recommended that an individual is made aware of the following before data are collected\(^{[53]}\):

- the purpose of the data collection and any (known) data mining projects
- how the data will be used
- who will be able to mine the data and use the data and their derivatives
- the status of security surrounding access to the data
- how collected data can be updated.

In America, privacy concerns have been addressed to some extent by the US Congress via the passage of regulatory controls such as the Health Insurance Portability and Accountability Act (HIPAA). The HIPAA requires individuals to give their "informed consent" regarding information they provide and its intended present and future uses. According to an article in *Biotech Business Week*, "'[i]n practice, HIPAA may not offer any greater protection than the longstanding regulations in the research arena,' says the AAHC. More importantly, the rule's goal of protection through informed consent is undermined by the complexity of consent forms that are required of patients and participants, which approach a level of incomprehensibility to average individuals."\(^{[54]}\) This underscores the necessity for data anonymity in data aggregation and mining practices.

Data may also be modified so as to become anonymous, so that individuals may not readily be identified.\(^{[53]}\) However, even "de-identified"/"anonymized" data sets can potentially contain enough information to allow identification of individuals, as occurred when journalists were able to find several individuals based on a set of search histories that were inadvertently released by AOL.\(^{[55]}\)

**Software**

**Free open-source data mining software and applications**

2. Chemicalize.org: A chemical structure miner and web search engine.
3. ELKI: A university research project with advanced cluster analysis and outlier detection methods written in the Java language.
4. GATE: a natural language processing and language engineering tool.
5. JHepWork: Java cross-platform data analysis framework developed at Argonne National Laboratory.
6. KNIME: The Konstanz Information Miner, a user friendly and comprehensive data analytics framework.
7. NLTK (Natural Language Toolkit): A suite of libraries and programs for symbolic and statistical natural language processing (NLP) for the Python language.
9. R: A programming language and software environment for statistical computing, data mining, and graphics. It is part of the GNU project.
11. UIMA: The UIMA (Unstructured Information Management Architecture) is a component framework for analyzing unstructured content such as text, audio and video – originally developed by IBM.
13. ML-Flex: A software package that enables users to integrate with third-party machine-learning packages written in any programming language, execute classification analyses in parallel across multiple computing nodes, and produce HTML reports of classification results.

In 2010, the open source R language overtook other tools to become the application used by more data miners (43%) than any other, according to a well-known annual survey.\(^{[47]}\)
Commercial data-mining software and applications

- Microsoft Analysis Services: data mining software provided by Microsoft
- SAS: Enterprise Miner – data mining software provided by the SAS Institute.
- STATISTICA: Data Miner – data mining software provided by StatSoft.
- Oracle Data Mining: data mining software by Oracle.
- LIONsolver: an integrated software application for data mining, business intelligence, and modeling that implements the Learning and Intelligent OptimizatioN (LION) approach

According to Rexer's Annual Data Miner Survey in 2010, IBM SPSS Modeler, STATISTICA Data Miner, and the R received the strongest satisfaction ratings.[47]

Marketplace surveys

Several researchers and organizations have conducted reviews of data mining tools and surveys of data miners. These identify some of the strengths and weaknesses of the software packages. They also provide an overview of the behaviors, preferences and views of data miners. Some of these reports include:

- Annual Rexer Analytics Data Miner Surveys[47]
- Forrester Research 2010 Predictive Analytics and Data Mining Solutions report[56]
- Gartner 2008 "Magic Quadrant" report[57]
- Haughton et al.’s 2003 Review of Data Mining Software Packages in *The American Statistician*[58]
- Robert A. Nisbet's 2006 Three Part Series of articles "Data Mining Tools: Which One is Best For CRM?"[59]
- 2011 Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery[60]

References

[6] Bouckaert, Remco R.; Frank, Eibe; Hall, Mark A.; Holmes, Geoffrey; Pfahringer, Bernhard; Reutemann, Peter; Witten, Ian H. (2010). "WEKA Experiences with a Java open-source project". *Journal of Machine Learning Research* 11: 2533–2541. "the original title, "Practical machine learning", was changed [...] The term "data mining" was [added] primarily for marketing reasons."

[16] Fountain, Tony; Dietterich, Thomas; and Sudyka, Bill (2000); Mining IC Test Data to Optimize VLSI Testing (http://web.engr. oregonstate.edu/~tgd/publications/kdd2000-diff.pdf), in Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM Press, pp. 18–25


[20] Superby Aguirre, Juan Francisco; Vandamme, Jean-Philippe; Meskens, Nadine. "Determination of factors influencing the achievement of the first-year university students using data mining methods". Workshop on Educational Data Mining 2006.


[31] Miller, Harvey J.; and Han, Jiawei (eds.) (2001); Geographic Data Mining and Knowledge Discovery, London, GB: Taylor & Francis


This citation will be automatically completed in the next few minutes. You can jump the queue or expand by hand (http://en.wikipedia.org/wiki/Template:cite_doi_/10.3390/s8063601/preloads/Template:Cite_doi/preload&editintro=Template:Cite_doi/editintro&action=edit)


[34] Zhao, Kaidi; and Liu, Bing; Tirpark, Thomas M.; and Weinim, Xiao; A Visual Data Mining Framework for Convenient Identification of Useful Knowledge (http://dl.acm.org/citation.cfm?id=1106390)


[37] Government Accountability Office, Data Mining: Early Attention to Privacy in Developing a Key DHS Program Could Reduce Risks, GAO-07-289 (February 2007), Washington, DC

[38] Secure Flight Program report (http://www.msbc.msn.com/id/20604775/), MSNBC


[40] Agrawal, Rakesh; Mannila, Heikki; Srikant, Ramakrishnan; Toivonen, Hannu; and Verkamo, A. Inkeri; Fast discovery of association rules, in Advances in knowledge discovery and data mining, MIT Press, 1996, pp. 307–328


Further reading

- Cabena, Peter; Hadjninian, Pablo; Stadler, Rolf; Verhees, Jaap; and Zanasi, Alessandro (1997); *Discovering Data Mining: From Concept to Implementation*, Prentice Hall, ISBN 0-13-743980-6


• Poncelet, Pascal; Masseglia, Florent; and Teisseire, Maguelonne (editors) (October 2007); "Data Mining Patterns: New Methods and Applications", Information Science Reference, ISBN 978-1-59904-162-9
• Tan, Pang-Ning; Steinbach, Michael; and Kumar, Vipin (2005); Introduction to Data Mining, ISBN 0-321-32136-7
• Weiss, Sholom M.; and Indurkhya, Nitin (1998); Predictive Data Mining, Morgan Kaufmann
• Witten, Ian H.; Frank, Eibe; Hall, Mark A. (30 January 2011). Data Mining: Practical Machine Learning Tools and Techniques (3 ed.). Elsevier. ISBN 978-0-12-374856-0. (See also Free Weka software)
• Ye, Nong (2003); The Handbook of Data Mining, Mahwah, NJ: Lawrence Erlbaum

External links
• Data Mining Software (http://www.dmoz.org/Computers/Software/Databases/Data_Mining/) at the Open Directory Project

Cross Industry Standard Process for Data Mining

CRISP-DM stands for Cross Industry Standard Process for Data Mining[1]. It is a data mining process model that describes commonly used approaches that expert data miners use to tackle problems. Polls conducted in 2002, 2004, and 2007 show that it is the leading methodology used by data miners.[2][3][4]

Major phases
CRISP-DM breaks the process of data mining into six major phases[5]:
• Business Understanding
• Data Understanding
• Data Preparation
• Modeling
• Evaluation
• Deployment

History
CRISP-DM was conceived in 1996. In 1997 it got underway as a European Union project under the ESPRIT funding initiative. The project was led by four companies: SPSS, Teradata, Daimler_AG and OHRA[6].

This core consortium brought different experiences to the project: ISL, later acquired and merged into SPSS Inc. The computer giant NCR Corporation produced the Teradata data warehouse and its own data mining software. Daimler-Benz had a significant data mining team. OHRA[6], an insurance company, was just starting to explore the potential use of data mining.

The first version of the methodology was released as CRISP-DM 1.0[7] in 1999.
CRISP-DM 2.0

In July 2006 the consortium announced that it was going to start the process of working towards a second version of CRISP-DM. On 26 September 2006, the CRISP-DM SIG [8] met to discuss potential enhancements for CRISP-DM 2.0 and the subsequent roadmap. However, these efforts appear to be stalled. The SIG has not met, updated the CRISP website, or communicated anything to members since early 2007. As of June 22, 2011, the website redirects to an IBM page about SPSS.

Advantages

• Industry neutral
• Tool neutral
• Closely related to the Knowledge Discovery in Databases Process Model
• Anchors the data mining process

References


External links

• CRoss Industry Standard Process for Data Mining Blog (http://crispmdblog.wordpress.com/)
• Le site des dataminers (http://lesitedesdataminers.free.fr/02_PAGES_WEB/conduite_projet_crisp_dm.html)
  Article publié par Pascal BIZZARI, Mai 2009
• The Data Mining Group (DMG) (http://www.dmg.org): The DMG is an independent, vendor led group which develops data mining standards, such as the Predictive Model Markup Language (PMML)
Statistical classification

In machine learning and statistics, classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. The individual observations are analyzed into a set of quantifiable properties, known as various explanatory variables, features, etc. These properties may variously be categorical (e.g. "A", "B", "AB" or "O", for blood type), ordinal (e.g. "large", "medium" or "small"), integer-valued (e.g. the number of occurrences of a part word in an email) or real-valued (e.g. a measurement of blood pressure). Some algorithms work only in terms of discrete data and require that real-valued or integer-valued data be discretized into groups (e.g. less than 5, between 5 and 10, or greater than 10). An example would be assigning a given email into "spam" or "non-spam" classes or assigning a diagnosis to a given patient as described by observed characteristics of the patient (gender, blood pressure, presence or absence of certain symptoms, etc.).

An algorithm that implements classification, especially in a concrete implementation, is known as a classifier. The term "classifier" sometimes also refers to the mathematical function, implemented by a classification algorithm, that maps input data to a category.

In the terminology of machine learning, classification is considered an instance of supervised learning, i.e. learning where a training set of correctly-identified observations is available. The corresponding unsupervised procedure is known as clustering (or cluster analysis), and involves grouping data into categories based on some measure of inherent similarity (e.g. the distance between instances, considered as vectors in a multi-dimensional vector space).

Terminology across fields is quite varied. In statistics, where classification is often done with logistic regression or a similar procedure, the properties of observations are termed explanatory variables (or independent variables, regressors, etc.), and the categories to be predicted are known as outcomes, which are considered to be possible values of the dependent variable. In machine learning, the observations are often known as instances, the explanatory variables are termed features (grouped into a feature vector), and the possible categories to be predicted are classes.

There is also some argument over whether classification methods that do not involve a statistical model can be considered "statistical". Other fields may use different terminology: e.g. in community ecology, the term "classification" normally refers to cluster analysis, i.e. a type of unsupervised learning, rather than the supervised learning described in this article.

Relation to other problems

Classification and clustering are examples of the more general problem of pattern recognition, which is the assignment of some sort of output value to a given input value. Other examples are regression, which assigns a real-valued output to each input; sequence labeling, which assigns a class to each member of a sequence of values (for example, part of speech tagging, which assigns a part of speech to each word in an input sentence); parsing, which assigns a parse tree to an input sentence, describing the syntactic structure of the sentence; etc.

A common subclass of classification is probabilistic classification. Algorithms of this nature use statistical inference to find the best class for a given instance. Unlike other algorithms, which simply output a "best" class, probabilistic algorithms output a probability of the instance being a member of each of the possible classes. The best class is normally then selected as the one with the highest probability. However, such an algorithm has numerous advantages over non-probabilistic classifiers:

• It can output a confidence value associated with its choice (in general, a classifier that can do this is known as a confidence-weighted classifier)
• Correspondingly, it can abstain when its confidence of choosing any particular output is too low
• Because of the probabilities output, probabilistic classifiers can be more effectively incorporated into larger machine-learning tasks, in a way that partially or completely avoids the problem of error propagation.
**Frequentist procedures**

Early work on statistical classification was undertaken by Fisher,[1][2] in the context of two-group problems, leading to Fisher's linear discriminant function as the rule for assigning a group to a new observation.[3] This early work assumed that data-values within each of the two groups had a multivariate normal distribution. The extension of this same context to more than two-groups has also been considered with a restriction imposed that the classification rule should be linear.[3][4] Later work for the multivariate normal distribution allowed the classifier to be nonlinear:[5] several classification rules can be derived based on slight different adjustments of the Mahalanobis distance, with a new observation being assigned to the group whose centre has the lowest adjusted distance from the observation.

**Bayesian procedures**

Unlike frequentist procedures, Bayesian classification procedures provide a natural way of taking into account any available information about the relative sizes of the sub-populations associated with the different groups within the overall population.[6] Bayesian procedures tend to be computationally expensive and, in the days before Markov chain Monte Carlo computations were developed, approximations for Bayesian clustering rules were devised.[7]

Some Bayesian procedures involve the calculation of group membership probabilities: these can be viewed as providing a more informative outcome of a data analysis than a simple attribution of a single group-label to each new observation.

**Binary and multiclass classification**

Classification can be thought of as two separate problems - binary classification and multiclass classification. In binary classification, a better understood task, only two classes are involved, whereas multiclass classification involves assigning an object to one of several classes.[8] Since many classification methods have been developed specifically for binary classification, multiclass classification often requires the combined use of multiple binary classifiers.

**Feature vectors**

Most algorithms describe an individual instance whose category is to be predicted using a feature vector of individual, measurable properties of the instance. Each property is termed a feature, also known in statistics as an explanatory variable (or independent variable, although in general different features may or may not be statistically independent). Features may variously be binary ("male" or "female"); categorical (e.g. "A", "B", "AB" or "O", for blood type); ordinal (e.g. "large", "medium" or "small"); integer-valued (e.g. the number of occurrences of a particular word in an email); or real-valued (e.g. a measurement of blood pressure). If the instance is an image, the feature values might correspond to the pixels of an image; if the instance is a piece of text, the feature values might be occurrence frequencies of different words. Some algorithms work only in terms of discrete data and require that real-valued or integer-valued data be discretized into groups (e.g. less than 5, between 5 and 10, or greater than 10).

The vector space associated with these vectors is often called the feature space. In order to reduce the dimensionality of the feature space, a number of dimensionality reduction techniques can be employed.
Linear classifiers

A large number of algorithms for classification can be phrased in terms of a linear function that assigns a score to each possible category $k$ by combining the feature vector of an instance with a vector of weights, using a dot product. The predicted category is the one with the highest score. This type of score function is known as a linear predictor function and has the following general form:

$$\text{score}(\mathbf{X}_i, k) = \mathbf{\beta}_k \cdot \mathbf{X}_i,$$

where $\mathbf{X}_i$ is the feature vector for instance $i$, $\mathbf{\beta}_k$ is the vector of weights corresponding to category $k$, and $\text{score}(\mathbf{X}_i, k)$ is the score associated with assigning instance $i$ to category $k$. In discrete choice theory, where instances represent people and categories represent choices, the score is considered the utility associated with person $i$ choosing category $k$.

Algorithms with this basic setup are known as linear classifiers. What distinguishes them is the procedure for determining (training) the optimal weights/coefficients and the way that the score is interpreted.

Examples of such algorithms:
- Logistic regression and multinomial logit
- Probit regression
- The perceptron algorithm
- Support vector machines
- Linear discriminant analysis

Algorithms

The most widely used classifiers are the neural network (multi-layer perceptron), support vector machines, k-nearest neighbours, Gaussian mixture model, Gaussian, naive Bayes, decision tree and RBF classifiers.

Examples of classification algorithms include:
- Linear classifiers
  - Fisher’s linear discriminant
  - Logistic regression
  - Naive Bayes classifier
  - Perceptron
  - Support vector machines
    - Least squares support vector machines
- Quadratic classifiers
- Kernel estimation
  - k-nearest neighbor
- Boosting (meta-algorithm)
- Decision trees
  - Random forests
- Neural networks
- Gene Expression Programming
- Bayesian networks
- Hidden Markov models
- Learning vector quantization
- Proaftn
Evaluation

Classifier performance depends greatly on the characteristics of the data to be classified. There is no single classifier that works best on all given problems (a phenomenon that may be explained by the no-free-lunch theorem). Various empirical tests have been performed to compare classifier performance and to find the characteristics of data that determine classifier performance. Determining a suitable classifier for a given problem is however still more an art than a science.

The measures precision and recall are popular metrics used to evaluate the quality of a classification system. More recently, receiver operating characteristic (ROC) curves have been used to evaluate the tradeoff between true- and false-positive rates of classification algorithms.

As a performance metric, the uncertainty coefficient has the advantage over simple accuracy in that it is not affected by the relative sizes of the different classes. Further, it will not penalize an algorithm for simply rearranging the classes.

Application domains

Classification problems has many applications. In some of these it is employed as a data mining procedure, while in others more detailed statistical modeling is undertaken.

- Computer vision
  - Medical imaging and medical image analysis
  - Optical character recognition
  - Video tracking
- Drug discovery and development
  - Toxicogenomics
  - Quantitative structure-activity relationship
- Geostatistics
- Speech recognition
- Handwriting recognition
- Biometric identification
- Biological classification
- Statistical natural language processing
- Document classification
- Internet search engines
- Credit scoring
- Pattern recognition

References

External links

- TOOLDIAG Pattern recognition toolbox (http://sites.google.com/site/tooldiag/).
- kNN and Potential energy (http://www.math.le.ac.uk/people/ag153/homepage/KNN/KNN3.html) (Applet), University of Leicester

Clustering analysis

Cluster analysis or clustering is the task of assigning a set of objects into groups (called clusters) so that the objects in the same cluster are more similar (in some sense or another) to each other than to those in other clusters.

Clustering is a main task of explorative data mining, and a common technique for statistical data analysis used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, and bioinformatics.

Cluster analysis itself is not one specific algorithm, but the general task to be solved. It can be achieved by various algorithms that differ significantly in their notion of what constitutes a cluster and how to efficiently find them. Popular notions of clusters include groups with low distances among the cluster members, dense areas of the data space, intervals or particular statistical distributions. Clustering can therefore be formulated as a multi-objective optimization problem. The appropriate clustering algorithm and parameter settings (including values such as the distance function to use, a density threshold or the number of expected clusters) depend on the individual data set and intended use of the results. Cluster analysis as such is not an automatic task, but an iterative process of knowledge discovery or interactive multi-objective optimization that involves trial and failure. It will often be necessary to modify preprocessing and parameters until the result achieves the desired properties.

Besides the term clustering, there are a number of terms with similar meanings, including automatic classification, numerical taxonomy, botryology (from Greek βότρυς "grape") and typological analysis. The subtle differences are often in the usage of the results: while in data mining, the resulting groups are the matter of interest, in automatic classification primarily their discriminative power is of interest. This often leads to misunderstandings between researchers coming from the fields of data mining and machine learning, since they use the same terms and often the same algorithms, but have different goals.

Clusters and clusterings

The notion of a "cluster" varies between algorithms and is one of the many decisions to take when choosing the appropriate algorithm for a particular problem. At first the terminology of a cluster seems obvious: a group of data objects. However, the clusters found by different algorithms vary significantly in their properties, and understanding these "cluster models" is key to understanding the differences between the various algorithms. Typical cluster models include:
• Connectivity models: for example hierarchical clustering builds models based on distance connectivity.
• Centroid models: for example the k-means algorithm represents each cluster by a single mean vector.
• Distribution models: clusters are modeled using statistic distributions, such as multivariate normal distributions used by the Expectation-maximization algorithm.
• Density models: for example DBSCAN and OPTICS defines clusters as connected dense regions in the data space.
• Subspace models: in Biclustering (also known as Co-clustering or two-mode-clustering), clusters are modeled with both cluster members and relevant attributes.
• Group models: some algorithms (unfortunately) do not provide a refined model for their results and just provide the grouping information.
• Graph-based models: a clique, i.e., a subset of nodes in a graph such that every two nodes in the subset are connected by an edge can be considered as a prototypical form of cluster. Relaxations of the complete connectivity requirement (a fraction of the edges can be missing) are known as quasi-cliques.

A "clustering" is essentially a set of such clusters, usually containing all objects in the data set. Additionally, it may specify the relationship of the clusters to each other, for example a hierarchy of clusters embedded in each other. Clusterings can be roughly distinguished in:
• hard clustering: each object belongs to a cluster or not
• soft clustering (also: fuzzy clustering): each object belongs to each cluster to a certain degree (e.g. a likelihood of belonging to the cluster)

There are also finer distinctions possible, for example:
• strict partitioning clustering: here each object belongs to exactly one cluster
• strict partitioning clustering with outliers: objects can also belong to no cluster, and are considered outliers.
• overlapping clustering (also: alternative clustering, multi-view clustering): while usually a hard clustering, objects may belong to more than one cluster.
• hierarchical clustering: objects that belong to a child cluster also belong to the parent cluster
• subspace clustering: while an overlapping clustering, within a uniquely defined subspace, clusters are not expected to overlap.

**Clustering algorithms**

Clustering algorithms can be categorized based on their cluster model, as listed above. The following overview will only list the most prominent examples of clustering algorithms, as there are probably a few dozen (if not over 100) published clustering algorithms. Not all provide models for their clusters and can thus not easily be categorized. An overview of algorithms explained in Wikipedia can be found in the list of statistics algorithms.

**Connectivity based clustering (hierarchical clustering)**

Connectivity based clustering, also known as hierarchical clustering, is based on the core idea of objects being more related to nearby objects than to objects farther away. As such, these algorithms connect "objects" to form "clusters" based on their distance. A cluster can be described largely by the maximum distance needed to connect parts of the cluster. At different distances, different clusters will form, which can be represented using a dendrogram, which explains where the common name "hierarchical clustering" comes from: these algorithms do not provide a single partitioning of the data set, but instead provide an extensive hierarchy of clusters that merge with each other at certain distances. In a dendrogram, the y-axis marks the distance at which the clusters merge, while the objects are placed along the x-axis such that the clusters don't mix.

Connectivity based clustering is a whole family of methods that differ by the way distances are computed. Apart from the usual choice of distance functions, the user also needs to decide on the linkage criterion (since a cluster
Cluster analysis

consists of multiple objects, there are multiple candidates to compute the distance to) to use. Popular choices are known as single-linkage clustering (the minimum of object distances), complete linkage clustering (the maximum of object distances) or UPGMA ("Unweighted Pair Group Method with Arithmetic Mean", also known as average linkage clustering). Furthermore, hierarchical clustering can be agglomerative (starting with single elements and aggregating them into clusters) or divisive (starting with the complete data set and dividing it into partitions).

While these methods are fairly easy to understand, the results are not always easy to use, as they will not produce a unique partitioning of the data set, but a hierarchy the user still needs to choose appropriate clusters from. The methods are not very robust towards outliers, which will either show up as additional clusters or even cause other clusters to merge (known as "chaining phenomenon", in particular with single-linkage clustering). In the general case, the complexity is $O(n^3)$, which makes them too slow for large data sets. For some special cases, optimal efficient methods (of complexity $O(n^2)$) are known: SLINK\textsuperscript{1} for single-linkage and CLINK\textsuperscript{2} for complete-linkage clustering. In the data mining community these methods are recognized as a theoretical foundation of cluster analysis, but often considered obsolete. They did however provide inspiration for many later methods such as density based clustering.

### Linkage clustering examples

---

**Single-linkage on Gaussian data.** At 35 clusters, the biggest cluster starts fragmenting into smaller parts, while before it was still connected to the second largest due to the single-link effect.

![Single-linkage on Gaussian data](image1)

**Single-linkage on density-based clusters.** 20 clusters extracted, most of which contain single elements, since linkage clustering does not have a notion of "noise".

![Single-linkage on density-based clusters](image2)
Centroid-based clustering

In centroid-based clustering, clusters are represented by a central vector, which may not necessarily be a member of the data set. When the number of clusters is fixed to \( k \), \( k \)-means clustering gives a formal definition as an optimization problem: find the \( k \) cluster centers and assign the objects to the nearest cluster center, such that the squared distances from the cluster are minimized.

The optimization problem itself is known to be NP-hard, and thus the common approach is to search only for approximate solutions. A particularly well known approximative method is Lloyd's algorithm,\(^3\) often actually referred to as "\( k \)-means algorithm". It does however only find a local optimum, and is commonly run multiple times with different random initializations. Variations of \( k \)-means often include such optimizations as choosing the best of multiple runs, but also restricting the centroids to members of the data set (\( k \)-medoids), choosing medians (\( k \)-medians clustering), choosing the initial centers less randomly (\( k \)-means++) or allowing a fuzzy cluster assignment (Fuzzy \( c \)-means).

Most \( k \)-means-type algorithms require the number of clusters - \( k \) - to be specified in advance, which is considered to be one of the biggest drawbacks of these algorithms. Furthermore, the algorithms prefer clusters of approximately similar size, as they will always assign an object to the nearest centroid. This often leads to incorrectly cut borders in between of clusters (which is not surprising, as the algorithm optimized cluster centers, not cluster borders).

\( K \)-means has a number of interesting theoretical properties. On one hand, it partitions the data space into a structure known as Voronoi diagram. On the other hand, it is conceptually close to nearest neighbor classification and as such popular in machine learning. Third, it can be seen as a variation of model based classification, and Lloyd's algorithm as a variation of the Expectation-maximization algorithm for this model discussed below.

\[ \text{k-Means clustering examples} \]

---

K-means separates data into Voronoi-cells, which assumes equal-sized clusters (not adequate here)

K-means cannot represent density-based clusters
**Distribution-based clustering**

The clustering model most closely related to statistics is based on distribution models. Clusters can then easily be defined as objects belonging most likely to the same distribution. A nice property of this approach is that this closely resembles the way artificial data sets are generated: by sampling random objects from a distribution.

While the theoretical foundation of these methods is excellent, they suffer from one key problem known as overfitting, unless constraints are put on the model complexity. A more complex model will usually always be able to explain the data better, which makes choosing the appropriate model complexity inherently difficult.

The most prominent method is known as expectation-maximization algorithm (or short: *EM-clustering*). Here, the data set is usually modeled with a fixed (to avoid overfitting) number of Gaussian distributions that are initialized randomly and whose parameters are iteratively optimized to fit better to the data set. This will converge to a local optimum, so multiple runs may produce different results. In order to obtain a hard clustering, objects are often then assigned to the Gaussian distribution they most likely belong to, for soft clusterings this is not necessary.

Distribution-based clustering is a semantically strong method, as it not only provides you with clusters, but also produces complex models for the clusters that can also capture correlation and dependence of attributes. However, using these algorithms puts an extra burden on the user: to choose appropriate data models to optimize, and for many real data sets, there may be no mathematical model available the algorithm is able to optimize (e.g. assuming Gaussian distributions is a rather strong assumption on the data).

**EM clustering examples**

On Gaussian-distributed data, EM works well, since it uses Gaussians for modelling clusters.

Density-based clusters cannot be modeled using Gaussian distributions.
Density-based clustering

In density-based clustering, clusters are defined as areas of higher density than the remainder of the data set. Objects in these sparse areas - that are required to separate clusters - are usually considered to be noise and border points.

The most popular density based clustering method is DBSCAN. In contrast to many newer methods, it features a well-defined cluster model called "density-reachability". Similar to linkage based clustering, it is based on connecting points within certain distance thresholds. However, it only connects points that satisfy a density criterion, in the original variant defined as a minimum number of other objects within this radius. A cluster consists of all density-connected objects (which can form a cluster of an arbitrary shape, in contrast to many other methods) plus all objects that are within these objects' range. Another interesting property of DBSCAN is that its complexity is fairly low - it requires a linear number of range queries on the database - and that it will discover essentially the same results (it is deterministic for core and noise points, but not for border points) in each run, therefore there is no need to run it multiple times. OPTICS is a generalization of DBSCAN that removes the need to choose an appropriate value for the range parameter $\varepsilon$, and produces a hierarchical result related to that of linkage clustering. DeLi-Clu, Density-Link-Clustering combines ideas from single-linkage clustering and OPTICS, eliminating the $\varepsilon$ parameter entirely and offering performance improvements over OPTICS by using an R-tree index.

The key drawback of DBSCAN and OPTICS is that they expect some kind of density drop to detect cluster borders. Moreover they can not detect intrinsic cluster structures which are prevalent in the majority of real life data. A variation of DBSCAN, EnDBSCAN efficiently detects such kinds of structures. On data sets with, for example, overlapping Gaussian distributions - a common use case in artificial data - the cluster borders produced by these algorithms will often look arbitrary, because the cluster density decreases continuously. On a data set consisting of mixtures of Gaussians, these algorithms are nearly always outperformed by methods such as EM clustering, that are able to precisely model this kind of data.

density-based clustering examples

Newer developments

In recent years considerable effort has been put into improving algorithm performance of the existing algorithms. Among them are CLARANS (Ng and Han, 1994), and BIRCH (Zhang et al., 1996). With the recent need to process larger and larger data sets (also known as big data), the willingness to treat semantic meaning of the generated clusters for performance has been increasing. This led to the development of pre-clustering methods such as canopy clustering, which can process huge data sets efficiently, but the resulting "clusters" are merely a rough
pre-partitioning of the data set to then analyze the partitions with existing slower methods such as k-means clustering. Various other approaches to clustering have been tried such as seed based clustering.[13]

For high-dimensional data, many of the existing methods fail due to the curse of dimensionality, which renders particular distance functions problematic in high-dimensional spaces. This led to new clustering algorithms for high-dimensional data that focus on subspace clustering (where only some attributes are used, and cluster models include the relevant attributes for the cluster) and correlation clustering that also looks for arbitrary rotated ("correlated") subspace clusters that can be modeled by giving a correlation of their attributes. Examples for such clustering algorithms are CLIQUE[14] and SUBCLU.[15]

Ideas from density-based clustering methods (in particular the DBSCAN/OPTICS family of algorithms) have been adopted to subspace clustering (HiSC,[16] hierarchical subspace clustering and DiSH[17]) and correlation clustering (HiCO,[18] hierarchical correlation clustering, 4C[19] using "correlation connectivity" and ERiC[20] exploring hierarchical density-based correlation clusters).

Several different clustering systems based on mutual information have been proposed. One is Marina Meilă's variation of information metric;[21] another provides hierarchical clustering.[22] Using genetic algorithms, a wide range of different fit-functions can be optimized, including mutual information.[23] Also message passing algorithms, a recent development in Computer Science and Statistical Physics, has led to the creation of new types of clustering algorithms.[24]

**Evaluation of clustering results**

Evaluation of clustering results sometimes is referred to as cluster validation.

There have been several suggestions for a measure of similarity between two clusterings. Such a measure can be used to compare how well different data clustering algorithms perform on a set of data. These measures are usually tied to the type of criterion being considered in assessing the quality of a clustering method.

**Internal evaluation**

When a clustering result is evaluated based on the data that was clustered itself, this is called internal evaluation. These methods usually assign the best score to the algorithm that produces clusters with high similarity within a cluster and low similarity between clusters. One drawback of using internal criteria in cluster evaluation is that high scores on an internal measure do not necessarily result in effective information retrieval applications.[25] Additionally, this evaluation is biased towards algorithms that use the same cluster model. For example k-Means clustering naturally optimizes object distances, and a distance-based internal criterion will likely overrate the resulting clustering.

The following methods can be used to assess the quality clustering algorithms based on internal criterion:

- **Davies–Bouldin index**

  The Davies–Bouldin index can be calculated by the following formula:

  \[
  DB = \frac{1}{n} \sum_{i=1}^{n} \max_{i \neq j} \left( \frac{\sigma_i + \sigma_j}{d(c_i, c_j)} \right)
  \]

  where \(n\) is the number of clusters, \(c_x\) is the centroid of cluster \(x\), \(\sigma_x\) is the average distance of all elements in cluster \(x\) to centroid \(c_x\), and \(d(c_i, c_j)\) is the distance between centroids \(c_i\) and \(c_j\). Since algorithms that produce clusters with low intra-cluster distances (high intra-cluster similarity) and high inter-cluster distances (low inter-cluster similarity) will have a low Davies–Bouldin index, the clustering algorithm that produces a collection of clusters with the smallest Davies–Bouldin index is considered the best algorithm based on this criterion.

- **Dunn index** (J. C. Dunn 1974)
The Dunn index aims to identify dense and well-separated clusters. It is defined as the ratio between the minimal inter-cluster distance to maximal intra-cluster distance. For each cluster partition, the Dunn index can be calculated by the following formula \[^{[26]}\]:

\[
D = \min_{1 \leq i \leq n} \left\{ \min_{1 \leq j \leq n, i \neq j} \left\{ \frac{d(i, j)}{\max_{1 \leq k \leq n} d(k)} \right\} \right\}
\]

where \(d(i, j)\) represents the distance between clusters \(i\) and \(j\), and \(d(k)\) measures the intra-cluster distance of cluster \(k\). The inter-cluster distance \(d(i, j)\) between two clusters may be any number of distance measures, such as the distance between the centroids of the clusters. Similarly, the intra-cluster distance \(d(k)\) may be measured in a variety ways, such as the maximal distance between any pair of elements in cluster \(k\). Since internal criterion seek clusters with high intra-cluster similarity and low inter-cluster similarity, algorithms that produce clusters with high Dunn index are more desirable.

**External evaluation**

In external evaluation, clustering results are evaluated based on data that was not used for clustering, such as known class labels and external benchmarks. Such benchmarks consist of a set of pre-classified items, and these sets are often created by human (experts). Thus, the benchmark sets can be thought of as a gold standard for evaluation. These types of evaluation methods measure how close the clustering is to the predetermined benchmark classes. However, it has recently been discussed whether this is adequate for real data, or only on synthetic data sets with a factual ground truth, since classes can contain internal structure, the attributes present may not allow separation of clusters or the classes may contain anomalies.\[^{[27]}\] Additionally, from a knowledge discovery point of view, the reproduction of known knowledge may not necessarily be the intended result.\[^{[27]}\]

Some of the measures of quality of a cluster algorithm using external criterion include:

- **Rand measure** (William M. Rand 1971)\[^{[28]}\]

  The Rand index computes how similar the clusters (returned by the clustering algorithm) are to the benchmark classifications. One can also view the Rand index as a measure of the percentage of correct decisions made by the algorithm. It can be computed using the following formula:

  \[
  RI = \frac{TP + TN}{TP + FP + FN + TN}
  \]

  where \(TP\) is the number of true positives, \(TN\) is the number of true negatives, \(FP\) is the number of false positives, and \(FN\) is the number of false negatives. One issue with the Rand index is that false positives and false negatives are equally weighted. This may be an undesirable characteristic for some clustering applications. The F-measure addresses this concern.

- **F-measure**

  The F-measure can be used to balance the contribution of false negatives by weighting recall through a parameter \(\beta \geq 0\). Let precision and recall be defined as follows:

  \[
  P = \frac{TP}{TP + FP}
  \]

  \[
  R = \frac{TP}{TP + FN}
  \]

  where \(P\) is the precision rate and \(R\) is the recall rate. We can calculate the F-measure by using the following formula\[^{[25]}\]:

  \[
  F_\beta = \frac{(\beta^2 + 1) \cdot P \cdot R}{\beta^2 \cdot P + R}
  \]

  Notice that when \(\beta = 0\), \(F_0 = P\). In other words, recall has no impact on the F-measure when \(\beta = 0\), and increasing \(\beta\) allocates an increasing amount of weight to recall in the final F-measure.
• **Pair-counting F-Measure** is the F-Measure applied to the set of object pairs, where objects are paired with each other when they are part of the same cluster. This measure is able to compare clusterings with different numbers of clusters.

• **Jaccard index**

  The Jaccard index is used to quantify the similarity between two datasets. The Jaccard index takes on a value between 0 and 1. An index of 1 means that the two dataset are identical, and an index of 0 indicates that the datasets have no common elements. The Jaccard index is defined by the following formula:

  \[ J(A, B) = \frac{|A \cap B|}{|A \cup B|} \]

  This is simply the number of unique elements common to both sets divided by the total number of unique elements in both sets.

• **Fowlkes–Mallows index** (E. B. Fowlkes & C. L. Mallows 1983\[29\])

  The Fowlkes-Mallows index computes the similarity between the clusters returned by the clustering algorithm and the benchmark classifications. The higher the value of the Fowlkes-Mallows index the more similar the clusters and the benchmark classifications are. It can be computed using the following formula:

  \[ FM = \sqrt{\frac{TP}{TP + FP} \cdot \frac{TP}{TP + FN}} \]

  where \(TP\)is the number of true positives, \(FP\)is the number of false positives, and \(FN\) is the number of false negatives.

• **Confusion matrix**

  A confusion matrix can be used to quickly visualize the results of a classification (or clustering) algorithm. It shows how different a cluster is from the gold standard cluster.

• **Mutual Information** is an information theoretic measure of how much information is shared between a clustering and a ground-truth classification that can detect a non-linear similarity between two clusterings.

  Adjusted mutual information is the corrected-for-chance variant of this that has a reduced bias for varying cluster numbers.

### Applications

Biology, computational biology and bioinformatics

Plant and animal ecology cluster analysis is used to describe and to make spatial and temporal comparisons of communities (assemblages) of organisms in heterogeneous environments; it is also used in Systematics to generate artificial Phylogeny or clusters of organisms (individuals) at the species, genus or higher level that share a number of attributes. Transcriptome clustering is used to build groups of genes with related expression patterns (also known as coexpressed genes). Often such groups contain functionally related proteins, such as enzymes for a specific metabolic pathway, or genes that are co-regulated. High throughput experiments using expressed sequence tags (ESTs) or DNA microarrays can be a powerful tool for genome annotation, a general aspect of genomics. Sequence analysis clustering is used to group homologous sequences into list of gene families. This is a very important concept in bioinformatics, and evolutionary biology in general. See evolution by gene duplication. High-throughput genotyping platforms clustering algorithms are used to automatically assign genotypes. Human genetic clustering The similarity of genetic data is used in clustering to infer population structures.

Medicine

Medical imaging On PET scans, cluster analysis can be used to differentiate between different types of tissue (biology) and blood in a three dimensional image. In this application, actual position does not matter, but the voxel intensity is considered as a coordinate vector, with a dimension for each image that was taken over
time. This technique allows, for example, accurate measurement of the rate a radioactive tracer is delivered to the area of interest, without a separate sampling of arterial blood, an intrusive technique that is most common today. IMRT segmentation Clustering can be used to divide a fluence map into distinct regions for conversion into deliverable fields in MLC-based Radiation Therapy.

Business and marketing Market research Cluster analysis is widely used in market research when working with multivariate data from Statistical surveys and test panels. Market researchers use cluster analysis to partition the general population of consumers into market segments and to better understand the relationships between different groups of consumers/potential customers, and for use in market segmentation, positioning (marketing) Product positioning, New product development and Selecting test markets. Grouping of shopping items Clustering can be used to group all the shopping items available on the web into a set of unique products. For example, all the items on eBay can be grouped into unique products. (eBay doesn't have the concept of a Stock-keeping unitSKU)

World wide web Social network analysis In the study of social networks, clustering may be used to recognize communities within large groups of people. Search result grouping In the process of intelligent grouping of the files and websites, clustering may be used to create a more relevant set of search results compared to normal search engines like Google. There are currently a number of web based clustering tools such as Clusty. Slippy map optimization Flickr's map of photos and other map sites use clustering to reduce the number of markers on a map. This makes it both faster and reduces the amount of visual clutter.

Computer science Software evolution Clustering is useful in software evolution as it helps to reduce legacy properties in code by reforming functionality that has become dispersed. It is a form of restructuring and hence is a way of directly preventative maintenance. Image segmentation Clustering can be used to divide a digital image into distinct regions for border detection or object recognition. Evolutionary algorithms Clustering may be used to identify different niches within the population of an evolutionary algorithm so that reproductive opportunity can be distributed more evenly amongst the evolving species or subspecies. Recommender systems Recommender systems are designed to recommend new items based on a user's tastes. They sometimes use clustering algorithms to predict a user's preferences based on the preferences of other users in the user's cluster. Markov chain Monte Carlo methods Clustering is often utilized to locate and characterize extrema in the target distribution.

Social science Crime analysis Cluster analysis can be used to identify areas where there are greater incidences of particular types of crime. By identifying these distinct areas or "hot spots" where a similar crime has happened over a period of time, it is possible to manage law enforcement resources more effectively. Educational data mining Cluster analysis is for example used to identify groups of schools or students with similar properties.

Others Mathematical chemistry To find structural similarity, etc., for example, 3000 chemical compounds were clustered in the space of 90 topological index topological indices. Basak S.C., Magnuson V.R., Niemi C.J., Regal R.R. "Determining Structural Similarity of Chemicals Using Graph Theoretic Indices". Discr. Appl. Math., 19, 1988: 17-44. Climatology To find weather regimes or preferred sea level pressure atmospheric patterns. Huth R. et al. "Classifications of Atmospheric Circulation Patterns: Recent Advances and Applications". Ann. N.Y. Acad. Sci., 1146, 2008: 105-152 Petroleum geology Cluster analysis is used to reconstruct missing bottom hole core data or missing log curves in order to evaluate reservoir properties. Physical geography The clustering of chemical properties in different sample locations.
Cluster analysis

References

This citation will be automatically completed in the next few minutes. You can jump the queue or expand by hand (http://en.wikipedia.org/wiki/Template:cite_doi/10.1109/TIT.1982.1056489?preload=Template:Cite_doi/preload&editintro=Template:Cite_doi/editintro&editaction=edit)
This citation will be automatically completed in the next few minutes. You can jump the queue or expand by hand (http://en.wikipedia.org/wiki/Template:cite_doi/10.1007/11731139_16?preload=Template:Cite_doi/preload&editintro=Template:Cite_doi/editintro&editaction=edit)
This citation will be automatically completed in the next few minutes. You can jump the queue or expand by hand (http://en.wikipedia.org/wiki/Template:cite_doi/10.1007/s10618-005-1396-1?preload=Template:Cite_doi/preload&editintro=Template:Cite_doi/editintro&editaction=edit)
[16] doi: 10.1007/11871637_42
This citation will be automatically completed in the next few minutes. You can jump the queue or expand by hand (http://en.wikipedia.org/wiki/Template:cite_doi/10.1007/11871637_42?preload=Template:Cite_doi/preload&editintro=Template:Cite_doi/editintro&editaction=edit)
[17] doi: 10.1007/978-3-540-71703-4_15
This citation will be automatically completed in the next few minutes. You can jump the queue or expand by hand (http://en.wikipedia.org/wiki/Template:cite_doi/10.1007/978-3-540-71703-4_15?preload=Template:Cite_doi/preload&editintro=Template:Cite_doi/editintro&editaction=edit)
[18] doi: 10.1109/SSDBM.2006.35
This citation will be automatically completed in the next few minutes. You can jump the queue or expand by hand (http://en.wikipedia.org/wiki/Template:cite_doi/10.1109/2fssdbm.2006.35?preload=Template:Cite_doi/preload&editintro=Template:Cite_doi/editintro&editaction=edit)
This citation will be automatically completed in the next few minutes. You can jump the queue or expand by hand (http://en.wikipedia.org/wiki/Template:cite_doi/10.1145/1007568.1007620?preload=Template:Cite_doi/preload&editintro=Template:Cite_doi/editintro&editaction=edit)
[20] doi: 10.1109/SSDBM.2007.21
This citation will be automatically completed in the next few minutes. You can jump the queue or expand by hand (http://en.wikipedia.org/wiki/Template:cite_doi/10.1109/2fssdbm.2007.21?preload=Template:Cite_doi/preload&editintro=Template:Cite_doi/editintro&editaction=edit)
Cluster analysis

148

action=edit


Association rule learning

In data mining, association rule learning is a popular and well researched method for discovering interesting relations between variables in large databases. Piatetsky-Shapiro[1] describes analyzing and presenting strong rules discovered in databases using different measures of interestingness. Based on the concept of strong rules, Rakesh Agrawal et al.[2] introduced association rules for discovering regularities between products in large scale transaction data recorded by point-of-sale (POS) systems in supermarkets. For example, the rule \{onions, potatoes\} \Rightarrow \{burger\} found in the sales data of a supermarket would indicate that if a customer buys onions and potatoes together, he or she is likely to also buy hamburger meat. Such information can be used as the basis for decisions about marketing activities such as, e.g., promotional pricing or product placements. In addition to the above example from market basket analysis association rules are employed today in many application areas including Web usage mining, intrusion detection and bioinformatics. As opposed to sequence mining, association rule learning typically does not consider the order of items either within a transaction or across transactions.

Definition
Following the original definition by Agrawal et al.\textsuperscript{[2]} the problem of association rule mining is defined as: Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of $n$ binary attributes called items. Let $D = \{t_1, t_2, \ldots, t_m\}$ be a set of transactions called the database. Each transaction in $D$ has a unique transaction ID and contains a subset of the items in $I$. A rule is defined as an implication of the form $X \Rightarrow Y$ where $X, Y \subseteq I$ and $X \cap Y = \emptyset$. The sets of items (for short itemsets) $X$ and $Y$ are called antecedent (left-hand-side or LHS) and consequent (right-hand-side or RHS) of the rule respectively.

To illustrate the concepts, we use a small example from the supermarket domain. The set of items is $I = \{\text{milk}, \text{bread}, \text{butter}, \text{beer}\}$ and a small database containing the items (1 codes presence and 0 absence of an item in a transaction) is shown in the table to the right. An example rule for the supermarket could be $\{\text{butter}, \text{bread}\} \Rightarrow \{\text{milk}\}$ meaning that if butter and bread are bought, customers also buy milk.

Note: this example is extremely small. In practical applications, a rule needs a support of several hundred transactions before it can be considered statistically significant, and datasets often contain thousands or millions of transactions.

**Useful Concepts**

To select interesting rules from the set of all possible rules, constraints on various measures of significance and interest can be used. The best-known constraints are minimum thresholds on support and confidence.

- The support $\text{supp}(X)$ of an itemset $X$ is defined as the proportion of transactions in the data set which contain the itemset. In the example database, the itemset $\{\text{milk}, \text{bread}, \text{butter}\}$ has a support of $1/5 = 0.2$ since it occurs in 20% of all transactions (1 out of 5 transactions).

- The confidence of a rule is defined $\text{conf}(X \Rightarrow Y) = \frac{\text{supp}(X \cup Y)}{\text{supp}(X)}$. For example, the rule $\{\text{milk, bread}\} \Rightarrow \{\text{butter}\}$ has a confidence of $0.2/0.4 = 0.5$ in the database, which means that for 50% of the transactions containing milk and bread the rule is correct (50% of the times a customer buys milk and bread, butter is bought as well). Be careful when reading the expression: here $\text{supp}(X \cup Y)$ means "support for occurrences of transactions where $X$ and $Y$ both appear", not "support for occurrences of transactions where either $X$ or $Y$ appears", the latter interpretation arising because set union is equivalent to logical disjunction. The argument of $\text{supp}()$ is a set of preconditions, and thus becomes more restrictive as it grows (instead of more inclusive).

- Confidence can be interpreted as an estimate of the probability $P(Y|X)$, the probability of finding the RHS of the rule in transactions under the condition that these transactions also contain the LHS.\textsuperscript{[3]}

- The lift of a rule is defined as $\text{lift}(X \Rightarrow Y) = \frac{\text{supp}(X \cup Y)}{\text{supp}(X) \times \text{supp}(Y)}$ or the ratio of the observed support to that expected if $X$ and $Y$ were independent. The rule $\{\text{milk, bread}\} \Rightarrow \{\text{butter}\}$ has a lift of $\frac{0.2}{0.4 \times 0.4} = 1.25$. 

**Example database with 4 items and 5 transactions**

<table>
<thead>
<tr>
<th>transaction ID</th>
<th>milk</th>
<th>bread</th>
<th>butter</th>
<th>beer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
The conviction of a rule is defined as $\text{conv}(X \Rightarrow Y) = \frac{1 - \text{supp}(Y)}{1 - \text{conf}(X \Rightarrow Y)}$. The rule $\{\text{milk, bread}\} \Rightarrow \{\text{butter}\}$ has a conviction of $\frac{1 - 0.4}{1 - 0.5} = 1.2$, and can be interpreted as the ratio of the expected frequency that $X$ occurs without $Y$ (that is to say, the frequency that the rule makes an incorrect prediction) if $X$ and $Y$ were independent divided by the observed frequency of incorrect predictions. In this example, the conviction value of 1.2 shows that the rule $\{\text{milk, bread}\} \Rightarrow \{\text{butter}\}$ would be incorrect 20% more often (1.2 times as often) if the association between $X$ and $Y$ was purely random chance.

Process

Association rules are usually required to satisfy a user-specified minimum support and a user-specified minimum confidence at the same time. Association rule generation is usually split up into two separate steps:

1. First, minimum support is applied to find all frequent itemsets in a database.
2. Second, these frequent itemsets and the minimum confidence constraint are used to form rules.

While the second step is straightforward, the first step needs more attention.

Finding all frequent itemsets in a database is difficult since it involves searching all possible itemsets (item combinations). The set of possible itemsets is the power set over $I$ and has size $2^n - 1$ (excluding the empty set which is not a valid itemset). Although the size of the powerset grows exponentially in the number of items $n$ in $I$, efficient search is possible using the downward-closure property of support (also called anti-monotonicity) which guarantees that for a frequent itemset, all its subsets are also frequent and thus for an infrequent itemset, all its supersets must also be infrequent. Exploiting this property, efficient algorithms (e.g., Apriori and Eclat) can find all frequent itemsets.

History

The concept of association rules was popularised particularly due to the 1993 article of Agrawal, which has acquired more than 6000 citations according to Google Scholar, as of March 2008, and is thus one of the most cited papers in the Data Mining field. However, it is possible that what is now called “association rules” is similar to what appears in the 1966 paper on GUHA, a general data mining method developed by Petr Hájek et al.

Alternative measures of interestingness

Next to confidence also other measures of interestingness for rules were proposed. Some popular measures are:

- All-confidence
- Collective strength
- Conviction
- Leverage
- Lift (originally called interest)
A definition of these measures can be found here\cite{15}. Several more measures are presented and compared by Tan et al.\cite{16} Looking for techniques that can model what the user has known (and using this models as interestingness measures) is currently an active research trend under the name of "Subjective Interestingness"

**Statistically sound associations**

One limitation of the standard approach to discovering associations is that by searching massive numbers of possible associations to look for collections of items that appear to be associated, there is a large risk of finding many spurious associations. These are collections of items that co-occur with unexpected frequency in the data, but only do so by chance. For example, suppose we are considering a collection of 10,000 items and looking for rules containing two items in the left-hand-side and 1 item in the right-hand-side. There are approximately 1,000,000,000,000 such rules. If we apply a statistical test for independence with a significance level of 0.05 it means there is only a 5% chance of accepting a rule if there is no association. If we assume there are no associations, we should nonetheless expect to find 50,000,000,000 rules. Statistically sound association discovery\cite{17}\cite{18} controls this risk, in most cases reducing the risk of finding any spurious associations to a user-specified significance level.

**Algorithms**

Many algorithms for generating association rules were presented over time.

Some well known algorithms are Apriori, Eclat and FP-Growth, but they only do half the job, since they are algorithms for mining frequent itemsets. Another step needs to be done after to generate rules from frequent itemsets found in a database.

**Apriori algorithm**

Apriori\cite{6} is the best-known algorithm to mine association rules. It uses a breadth-first search strategy to count the support of itemsets and uses a candidate generation function which exploits the downward closure property of support.

**Eclat algorithm**

Eclat\cite{7} is a depth-first search algorithm using set intersection.

**GUHA procedure ASSOC**

GUHA is a general method for exploratory data analysis that has theoretical foundations in observational calculi\cite{19}. The ASSOC procedure\cite{20} is a GUHA method which mines for generalized association rules using fast bitstrings operations. The association rules mined by this method are more general than those output by apriori, for example "items" can be connected both with conjunction and disjunctions and the relation between antecedent and consequent of the rule is not restricted to setting minimum support and confidence as in apriori: an arbitrary combination of supported interest measures can be used.
OPUS search

OPUS is an efficient algorithm for rule discovery that, in contrast to most alternatives, does not require either monotone or anti-monotone constraints such as minimum support.[21] Initially used to find rules for a fixed consequent,[21][22] it has subsequently been extended to find rules with any item as a consequent.[23] OPUS search is the core technology in the popular Magnum Opus[24] association discovery system.

Lore

A famous story about association rule mining is the "beer and diaper" story. A purported survey of behavior of supermarket shoppers discovered that customers (presumably young men) who buy diapers tend also to buy beer. This anecdote became popular as an example of how unexpected association rules might be found from everyday data. There are varying opinions as to how much of the story is true.[25] Daniel Powers says:[25]

In 1992, Thomas Blischok, manager of a retail consulting group at Teradata, and his staff prepared an analysis of 1.2 million market baskets from about 25 Osco Drug stores. Database queries were developed to identify affinities. The analysis "did discover that between 5:00 and 7:00 p.m. that consumers bought beer and diapers". Osco managers did NOT exploit the beer and diapers relationship by moving the products closer together on the shelves.

Other types of association mining

Contrast set learning is a form of associative learning. Contrast set learners use rules that differ meaningfully in their distribution across subsets.[26]

Weighted class learning is another form of associative learning in which weight may be assigned to classes to give focus to a particular issue of concern for the consumer of the data mining results.

K-optimal pattern discovery provides an alternative to the standard approach to association rule learning that requires that each pattern appear frequently in the data.

Mining frequent sequences uses support to find sequences in temporal data.[27]

Generalized Association Rules hierarchical taxonomy (concept hierarchy)

Quantitative Association Rules categorial and quantitative data[28]

Interval Data Association Rules e.g. partition the age into 5-year-increment ranged

Maximal Association Rules

Sequential Association Rules temporal data e.g. first buy computer, then CD-Roms, then a webcam.

References


[2] Agrawal, Rakesh; Imieliński, Tomasz; Swami, Arun; Mining Association Rules Between Sets of Items in Large Databases, SIGMOD Conference 1993:207-216


[5] Pei, Jian; Han, Jiawei; and Lakshmanan, Laks V. S.; Mining frequent itemsets with convertible constraints, in Proceedings of the 17th International Conference on Data Engineering, April 2–6, 2001, Heidelberg, Germany, 2001, pages 433-442

[6] Agrawal, Rakesh; and Srikant, Ramakrishnan; Fast algorithms for mining association rules in large databases (http://takesh.agrawal-family.com/papers/vldb94apriori.pdf), in Bocca, Jorge B.; Jarke, Matthias; and Zaniolo, Carlo; editors, Proceedings of the 20th International Conference on Very Large Data Bases (VLDB), Santiago, Chile, September 1994, pages 487-499

Association rule learning

Implementations

Bibliographies

External links

Implementations

SIPINA (http://eric.univ-lyon2.fr/~ricco/sipina.html), a free, academic data mining software which includes a model for association rule learning.

Pervasive DataRush (http://www.pervasivedatarush.com), a data mining platform for big data, includes association rule mining.

KXEN, a commercial Data Mining software (http://www.KXEN.com)
• Silverlight widget for live demonstration of association rule mining using Apriori algorithm (http://codeding.com/?article=13)
• RapidMiner, a free Java data mining software suite (Community Edition: GNU)
• Orange, a free data mining software suite, module orngAssoc (http://www.ailab.si/orange/doc/modules/orngAssoc.htm)
• Ruby implementation (AHR) (http://ai4r.org)
• arules (http://cran.r-project.org/package=arules), a package for mining association rules and frequent itemsets with R
• C. Borgelt's implementation of Apriori and Eclat (http://www.borgelt.net/fpm.html)
• Frequent Itemset Mining Implementations Repository (FIMI) (http://fimi.cs.helsinki.fi/)
• Frequent pattern mining implementations from Bart Goethals (http://adrem.ua.ac.be/~goethals/software/)
• Weka (http://www.cs.waikato.ac.nz/ml/weka/), a collection of machine learning algorithms for data mining tasks written in Java
• KNIME an open source workflow oriented data preprocessing and analysis platform
• Zaki, Mohammed J.; Data Mining Software (http://www.cs.rpi.edu/~zaki/software/)
• Magnum Opus (http://lispminer.vse.cz), mines for generalized (GUHA) association rules (uses bitstrings, not apriori algorithm)
• Ferda Dataminer (http://ferda.sourceforge.net), an extensible visual data mining platform, implements GUHA procedures ASSOC and features multirelational data mining
• STATISTICA (http://www.statsoft.com), commercial statistics software with an Association Rules module
• SPMF (http://www.philippe-fournier-viger.com/spmf/), Java implementations of more than 40 algorithms for frequent itemsets mining, association rule mining and sequential pattern mining. Includes a simple user interface and source code distributed under the GPL license.
• ARtool (http://www.cs.umb.edu/~laur/ARtool/), GPL Java association rule mining application with GUI, offering implementations of multiple algorithms for discovery of frequent patterns and extraction of association rules (includes Apriori and FPgrowth)

Open Standards

• Association Rules in PMML (http://www.dmg.org/v4-0/AssociationRules.html)
Sequence mining

Sequence mining is a topic of data mining concerned with finding statistically relevant patterns between data examples where the values are delivered in a sequence. It is usually presumed that the values are discrete, and thus time series mining is closely related, but usually considered a different activity. Sequence mining is a special case of structured data mining.

There are several key traditional computational problems addressed within this field. These include building efficient databases and indexes for sequence information, extracting the frequently occurring patterns, comparing sequences for similarity, and recovering missing sequence members. In general, sequence mining problems can be classified as string mining which is typically based on string processing algorithms and itemset mining which is typically based on association rule learning.

String Mining

String mining typically deals with a limited alphabet for items that appear in a sequence, but the sequence itself may be typically very long. Examples of an alphabet can be those in the ASCII character set used in natural language text, nucleotide bases ‘A’, ‘G’, ‘C’ and ‘T’ in DNA sequences, or amino acids for protein sequences. In biology applications analysis of the arrangement of the alphabet in strings can be used to examine gene and protein sequences to determine their properties. Knowing the sequence of letters of a DNA a protein is not an ultimate goal in itself. Rather, the major task is to understand the sequence, in terms of its structure and biological function. This is typically achieved first by identifying individual regions or structural units within each sequence and then assigning a function to each structural unit. In many cases this requires comparing a given sequence with previously studied ones. The comparison between the strings becomes complicated when insertions, deletions and mutations occur in a string.

A survey and taxonomy of the key algorithms for sequence comparison for bioinformatics is presented in the paper String Mining in Bioinformatics which include:

- **Repeat-related problems**: that deal with operations on single sequences and can be based on exact string matching or approximate string matching methods for finding dispersed fixed length and maximal length repeats, finding tandem repeats, and finding unique subsequences and missing (un-spelled) subsequences.

- **Alignment problems**: that deal with comparison between strings by first aligning one or more sequences; examples of popular methods include BLAST for comparing a single sequence with multiple sequences in a database, and ClustalW for multiple alignments. Alignment algorithms can be based on either exact or approximate methods, and can also be classified as global alignments, semi-global alignments and local alignment. See sequence alignment.

Itemset Mining

Some problems in sequence mining lend themselves discovering frequent itemsets and the order they appear, for example, one is seeking rules of the form “if a [customer buys a car], he or she is likely to [buy insurance] within 1 week”, or in the context of stock prices, “if [Nokia up and Ericsson Up], it is likely that [Motorolla up and Samsung up] within 2 days”. Traditionally, itemset mining is used in marketing applications for discovering regularities between frequently co-occurring items in large transactions. For example, by analysing transactions of customer shopping baskets in a supermarket, one can produce a rule which reads “if a customer buys onions and potatoes together, he or she is likely to also buy hamburger meat in the same transaction”.

A survey and taxonomy of the key algorithms for item set mining is presented in the paper Frequent pattern mining: current status and future directions.
Sequence mining

The two common techniques that are applied to sequence databases for frequent itemset mining are the influential apriori algorithm and the more-recent FP-Growth technique.

References


[2] http://scholar.google.co.uk/scholar?q=String+Mining+in+Bioinformatics&hl=en&as_sdt=1%2C5&as_sdtp=on

[3] String M. Abouelhoda, M. Ghanem. String Mining in Bioinformatics. In M. M. Gaber (Editor) Scientific Data Mining and Knowledge Discovery. Springer2009 (http://scholar.google.co.uk/scholar?q=String+Mining+in+Bioinformatics&hl=en&as_sdt=1,5&as_sdtp=on)

[4] http://scholar.google.co.uk/scholar?hl=en&q=Frequent+pattern+mining%3A+current+status+and+future+directions+&as_sdt=0%2C5&as_ylo=&as_vis=0


Anomaly detection

Anomaly detection, also referred to as outlier detection[1] refers to detecting patterns in a given data set that do not conform to an established normal behavior.[2] The patterns thus detected are called anomalies and often translate to critical and actionable information in several application domains. Anomalies are also referred to as outliers, change, deviation, surprise, aberrant, peculiarity, intrusion, etc.

In particular in the context of abuse and network intrusion detection, the interesting objects are often not rare objects, but unexpected bursts in activity. This pattern does not adhere to the common statistical definition of an outlier as a rare object, and many outlier detection methods (in particular unsupervised methods) will fail on such data, unless it has been aggregated appropriately. Instead, a cluster analysis algorithm may be able to detect the micro clusters formed by these patterns.

Three broad categories of anomaly detection techniques exist. Unsupervised anomaly detection techniques detect anomalies in an unlabeled test data set under the assumption that the majority of the instances in the data set are normal by looking for instances that seem to fit least to the remainder of the data set. Supervised anomaly detection techniques require a data set that has been labeled as "normal" and "abnormal" and involves training a classifier (the key difference to many other statistical classification problems is the inherent unbalanced nature of outlier detection). Semi-supervised anomaly detection techniques construct a model representing normal behavior from a given normal training data set, and then testing the likelihood of a test instance to be generated by the learnt model.

Applications

Anomaly detection is applicable in a variety of domains, such as intrusion detection, fraud detection, fault detection, system health monitoring, event detection in sensor networks, and detecting eco-system disturbances. It is often used in preprocessing to remove anomalous data from the dataset. In supervised learning, removing the anomalous data from the dataset often results in a statistically significant increase in accuracy.[3][4]

Popular techniques

Several anomaly detection techniques have been proposed in literature. Some of the popular techniques are:

- Distance based techniques (k-nearest neighbor, Local Outlier Factor[5]).
- One Class Support Vector Machines.
- Replicator Neural Networks.
- Cluster analysis based outlier detection.
- Pointing at records that deviate from association rules
• Conditional Anomaly Concept [6]
• Control charts such as the real-time contrasts chart.

Application to data security

Anomaly detection was proposed for Intrusion detection systems (IDS) by Dorothy Denning in 1986. [7] Anomaly detection for IDS is normally accomplished with thresholds and statistics, but can also be done with Soft computing, and inductive learning [8]. Types of statistics proposed by 1999 included profiles of users, workstations, networks, remote hosts, groups of users, and programs based on frequencies, means, variances, covariances, and standard deviations. [9] The counterpart of Anomaly detection in Intrusion detection is Misuse Detection.

Time series outlier detection

Parametric tests to find outliers in time series are implemented in almost all statistical packages: Demetra+, for example, uses the most popular ones. One way to detect anomalies in time series is a simple non parametric method called washer. [10] It uses a non parametric test to find one or more outliers in a group of even very short time series. The group must have a similar behaviour, as explained more fully below. An example is that of municipalities cited in the work of Dahlberg and Johanssen (2000). [11] Swedish municipalities expenditures between 1979 and 1987 represent 256 time series. If you consider three years such as, for example, 1981,1982 and 1983, you have 256 simple polygonal chains made of two lines segments. Every couple of segments can approximate a straight line or a convex downward (or convex upward) simple polygonal chain. The idea is to find outliers among the couples of segments that performs in a too much different way from the other couples. In the washer procedure every couple of segments is represented by an index and a non parametric test (Sprent test [12]) is applied to the unknown distribution of those indices. [13] For implementing washer methodology you can download an open source R (programming language) function with a simple numeric example. [14]

References

   This citation will be automatically completed in the next few minutes. You can jump the queue or expand by hand (http://en.wikipedia.org/wiki/Template:cite_doi_/10.1145.2f335191.335388?preload=Template:Cite_doi preload&editintro=Template:Cite_doi/editintro&action=edit)
[13] https://sites.google.com/site/andreaventurini65/home/outlier-detection
Image Sources, Licenses and Contributors


Image:Synapse Self-Organizing Map.png Source: http://en.wikipedia.org/w/index.php?title=File:Synapse_Self-Organizing_Map.png License: Creative Commons Attribution-ShareAlike 2.5 Contributors: Original uploader was Demoir at en.wikipedia


File:FrequentItems.png Source: http://en.wikipedia.org/w/index.php?title=File:FrequentItems.png License: Creative Commons Attribution-ShareAlike 3.0 Contributors: Xodorap00 (talk). Original uploader was Xodorap00 at en.wikipedia