# Artificial intelligence analysis

[1]**RACHITA RAUL,** Gandhi Institute of Excellent Technocrats, Bhubaneswar, India
[2]**PRIYANKA MOHANTY,** Indus College of Engineering, Bhubaneswar, Odisha, India

**Abstract:** ***Tools of advancing sophistication have been created throughout many ages to benefit the human race. In many ways, digital computers are just another tool. They are capable of performing the same types of numerical and symbolic manipulations as a regular person, but more quickly and accurately. This essay provides a survey of artificial intelligence algorithms used in software and computer applications. Include knowledge-based systems. The study of simulating human mental processes in computers is known as computational intelligence, which leads to artificial intelligence. That supports a doctor's ability to deconstruct a medical diagnosis.***

**Keywords:** knowledge-based systems, computational intelligence, symbolic learning, intelligent agents, WEKA machine learning

## 1. Introduction

Over many centuries, tools of increasing sophistication have been developed to serve the human race. Digital computers are, in many respects, just another tool. They can perform the same sort of numerical and symbolic manipulations that an ordinary person can, but faster and more reliably. A more intriguing idea is whether we can build a computer (or a computer program) that can think. As Penrose (1989) has pointed out, most of us are quite happy with machines that enable us to do physical things more easily or more quickly, such as digging a hole or traveling along a freeway. We are also happy to use machines that enable us to do physical things that would otherwise be impossible, such as flying. However, the idea of a machine that can think for us is a huge leap forward in our ambitions, and one which raises many ethical and philosophical questions. Research in artificial intelligence (or simply AI) is directed toward building such a machine and improving our understanding of intelligence. Most of the definitions in the standard texts are over-complex, so here is a simple one that will suffice instead:

Artificial intelligence is the science of mimicking human mental faculties in a computer.

The ultimate achievement in this field would be to construct a machine that can mimic or exceed human mental capabilities, including reasoning, understanding, imagination, perception, recognition, creativity, and emotions. We are a long way from achieving this, but some significant successes have nevertheless been achieved.

Perhaps more importantly, in achieving these modest successes, research into artificial intelligence has resulted in the development of a family of extremely useful computing tools. These tools have enabled a range of problems to be tackled that were previously considered too difficult, and have enabled a large number of other problems to be tackled more effectively. From a pragmatic point of view, this in itself makes them interesting and useful.

The tools of AI can be roughly divided into these broad types:

- Knowledge based systems (KBSs), i.e., explicit models using words and symbols;
- Computational intelligence (CI), i.e., implicit modeling with numerical techniques; hybrids.

The first category includes techniques such as rule-based, model-based, frame-based, and case-based reasoning. As the knowledge is explicitly modeled in words and symbols, it can be read and understood by a human. Although symbolic techniques have had undoubted success in their narrow domains, they are intrinsically limited in their ability to cope only with situations that have been explicitly modeled. Although some systems allow the model to expand with experience, symbolic models are generally poor at dealing with the unfamiliar.

Computational intelligence goes some way to overcoming these difficulties by enabling the computer to build up its own model, based on observations and experience. Here the knowledge is not explicitly stated but is represented by numbers that are adjusted as the system improves its accuracy. This category includes neural networks, genetic algorithms and other optimization algorithms, as well as techniques for handling uncertainty, such as fuzzy logic.

Pinpointing the beginning of research into artificial intelligence is tricky. George Boole (1815–1864) had plenty of ideas on the mathematical analysis of thought processes, and several of his ideas have been retained in the field of AI today. However, since he had no computer, the above definition appears to rule him out as the founder of AI. Just as historians on either side of the Atlantic have different opinions of who built the first programmable computer, the same divergence of opinion occurs over the origins of AI.

British historians point to Alan Turing's article in 1950 which included the so-called Turing test to determine whether a computer displays intelligence (Turing, 1950). American historians prefer to point to the Dartmouth conference of 1956, which was explicitly billed as a study of AI and is believed to be the first published use of the term 'artificial intelligence'. As the golden jubilee of that historic event approaches, a review of the field is timely.

The figure (figure 1) below illustrates the types and relationships of the Artificial Intelligence Techniques.

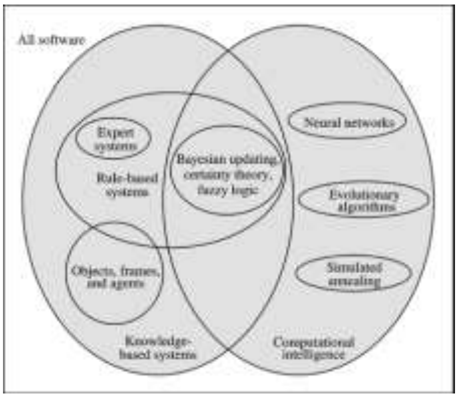


**Figure 1:** Categories of intelligent system software.

## 2. Knowledge Based Systems

### Knowledge-Based and Expert Systems

The principal difference between a knowledge-based system and a conventional program lies in its structure. In a conventional program, domain knowledge is intimately intertwined with software for controlling the application of that knowledge.

In a knowledge-based system, the two roles are explicitly separated. In the simplest case there are two modules—the knowledge module is called the knowledge base, and the control module is called the inference engine (Figure 2).

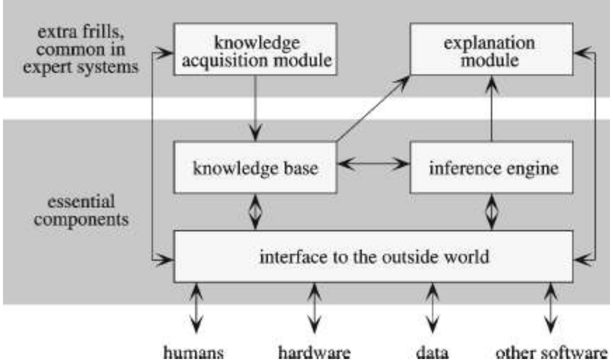


**Figure 2:** The main components of a Knowledge-based system

Within the knowledge base, the programmer expresses information about the problem to be solved. Often this information is declarative, i.e., the programmer states some facts, rules, or relationships without having to be concerned with the detail of how and when that information should be applied. These details are implicit in the inference engine. However, the domain expert will often wish to use meta-knowledge (i.e. knowledge about knowledge) to steer the inference engine. For instance, he or she may know that a plumbing diagnostic system should examine rules about valves before rules about pipes. In the most complex case, the inference engine can become a meta-knowledge-based system.

As the knowledge is represented explicitly in the knowledge base, rather than implicitly within the structure of a program, it can be entered and updated with relative ease by domain experts who may not have any programming expertise. The inference engine uses the knowledge base in a manner that can be likened to a conventional program using a data file. There is also an analogy with the brain, the control processes of which are approximately unchanging in their nature (like the inference engine), even though individual behavior is continually modified by new knowledge and experience (like updating the knowledge base).

Expert systems are a type of knowledge-based system designed to embody expertise in a particular specialized domain such as configuring computer networks or diagnosing faulty equipment. An expert system is intended to act as a human expert who can be consulted on a range of problems within his or her domain of expertise.

Typically, the user of an expert system will enter into a dialogue in which he or she describes the problem—such as the symptoms of a fault—and the expert system offers advice, suggestions, or recommendations. It is often proposed that an expert system must offer certain capabilities that mirror those of a human consultant. In particular, it is often claimed that an expert system must be capable of justifying its current line of inquiry and explaining its reasoning in arriving at a conclusion. This is the purpose of the explanation module in Figure 2.

### The Limitations of Rules

When modeling a real system, the amount of knowledge that can be represented in rules that operate on simple variables is limited. Frames provide a flexible structure for modeling complex entities, thereby allowing the creation of more flexible and versatile rules. One key use of frames is in the construction of model-based systems, which are particularly important for fault diagnosis. The links between symptoms and diagnosis are not explicitly stated but can be inferred by comparing the characteristics of a model with those of the real system.

Symbolic learning is an area in which rules can be expanded and altered in the light of experience. An important class of symbolic learning is case-based reasoning, in which previously encountered cases are stored for possible future retrieval and re-use. Finally, this section will consider some of the ways in which rules can be embellished to represent uncertainty and imprecision in the evidence, the conclusion, or the link between them.

ISSN: 2278-4632
Vol-10 Issue-09 No.03 September 2020

## 3. Frame Based Systems

Frames are data structures developed by AI researchers as a means of representing and organizing knowledge. They are similar in concept to objects, which were developed to meet the needs of software engineers. Like object-oriented systems, frame-based systems contain the ideas of classes, instances, and inheritance. For example, the class vehicle could be defined, along with subclasses car and truck. Characteristics of vehicle are inherited by car and truck classes, so that only information that is specific to the sub-class, or which overrides the inherited information, needs to be declared at the subclass level. Specific instances of classes can then be declared, e.g., my truck can be represented by an instance called my_truck. This instance inherits information from its class truck, which itself inherits from its parent class vehicle.

The attributes of a frame are sometimes called slots, into which values can be inserted. They allow us to put information onto a frame, such as the number of wheels on my truck. Thus number_of_wheels could be a slot associated with the frame instance my_truck. This slot could use the default value of 4 inherited from vehicle or it may be a locally defined value that overrides the default. The value associated with a slot can be a number, a description, a number range, a procedure, another frame, or anything allowed by the particular implementation. Some frame-based systems allow us to place multiple values in a slot. In such systems, the different pieces of information that we might want to associate with a slot are known as its facets. Each facet can have a value associated with it, as shown in Figure 3. For example, we may wish to specify limits on the number of wheels, provide a default, or calculate a value using a function known as an access function. In this example, an access function count_wheels could calculate the number of wheels when a value is not previously known.

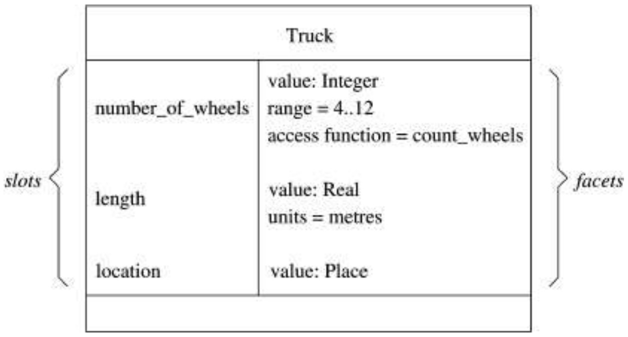


**Figure 3:** Example of a Frame-based Representation

### Model Based Reasoning

Fulton and Pepe (1990) have highlighted three major inadequacies of a purely rule-based system in the context of diagnosing faults: (a) building a complete rule set is a massive task; (b) there is uncertainty arising from whether sensor readings can be believed; and (c) maintaining the rules is a complex task because of the interdependence between them. They used these arguments to justify a model-based approach to fault diagnosis.

The principle of model-based reasoning is that, rather than storing a huge collection of symptom–cause pairs in the form of rules, these pairs can be generated by applying underlying principles to the model. The model, which is often frame-based, may describe any kind of system, including physical (Fenton et al, 2001, Wotawa 2000), software (Mateis et al, 2000), medical (Montani et al, 2003) legal (Bruninghaus and Ashley, 2003), and behavioral (Koning et al, 2000) systems. This review will focus on fault diagnosis in physical systems, which are made up of fundamental components such as tubes, wires, batteries, and valves. As each of these components performs a fairly simple role, it also has a simple failure mode. For example, a wire may break and fail to conduct electricity, a tube can spring a leak, a battery can lose its charge, and a valve may become stuck. Given a model of how these components operate and interact to form a device, faults can be diagnosed by determining the effects of local malfunctions on the overall device.

The diagnostic task is to determine which nonstandard component behavior in the model could make the output values of the model match those of the physical system. When a malfunction has been detected, the single point of failure assumption is often made. This is the assumption that the malfunction has only one root cause. Such an approach is justified by Fulton and Pepe (1990) on the basis that no two failures are truly simultaneous. They argue that one failure will always follow the other either independently or as a direct result.

In summary, the key advantages of model-based reasoning for fault diagnosis are:

- A model is less cumbersome to maintain than a rule base. Real-world changes are easily reflected in changes in the model.
- The model need not waste effort looking for sensor verification. Sensors are treated identically to other components, and therefore a faulty sensor is as likely to be detected as any other fault.
- Unusual failures are just as easy to diagnose as common ones. This is not the case in a rule-based system, which is likely to be most comprehensive in the case of common faults.
- The separation of function, structure, and state may help a diagnostic system to reason about a problem that is outside its area of expertise.
- The model can simulate a physical system, for the purpose of monitoring or for verifying a hypothesis.

### Symbolic Learning

The preceding sections have discussed ways of representing knowledge and drawing inferences. It was assumed that the knowledge itself was readily available and could be expressed explicitly. However, there are many circumstances where this is not the case, such as those listed below.

- The software engineer may need to obtain the knowledge from a domain expert. This task of knowledge acquisition is extensively discussed in the literature (Xing et al, 2003), often as an exercise in psychology.

- The rules that describe a particular domain may not be known.
- The problem may not be expressible explicitly in terms of rules, facts or relationships. This category includes skills, such as welding or painting.

One way around these difficulties is to have the system learn for itself from a set of example solutions. Two approaches can be broadly recognized—symbolic learning and numerical learning. Symbolic learning describes systems that formulate and modify rules, facts, and relationships, explicitly expressed in words and symbols. In other words, they create and modify their own knowledge base. Numerical learning refers to systems that use numerical models—learning in this context refers to techniques for optimizing the numerical parameters. Numerical learning includes genetic algorithms and artificial neural networks.

A learning system is usually given some feedback on its performance. The source of this feedback is called the teacher or the oracle. Often the teacher role is fulfilled by the environment, within which the knowledge-based system is working, i.e., the reaction of the environment to a decision is sufficient to indicate whether the decision was right or wrong. Learning with a teacher is sometimes called supervised learning.

Learning can be classified as follows, where each category involves a different level of supervision:

(i) Rote learning. The system receives confirmation of correct decisions. When it produces an incorrect decision, it is "spoon-fed" with the correct rule or relationship that it should have used.
(ii) Learning from advice. Rather than being given a specific rule that should apply in a given circumstance, the system is given a piece of general advice, such as "gas is more likely to escape from a valve than from a pipe." The system must sort out for itself how to move from this high-level abstract advice to an immediately usable rule.
(iii) Learning by induction. The system is presented with sets of example data and is told the correct conclusions that it should draw from each. The system continually refines its rules and relations so as to correctly handle each new example.
(iv) Learning by analogy. The system is told the correct response to a similar, but not identical, task. The system must adapt the previous response to generate a new rule applicable to the new circumstances.
(v) Explanation-based learning (EBL). The system analyzes a set of example solutions and their outcomes to determine why each one was successful or otherwise. Explanations are generated, which are used to guide future problem solving. EBL is incorporated into PRODIGY, a general-purpose problem-solver (Minton et al, 1989).
(vi) Case-based reasoning. Any case about which the system has reasoned is filed away, together with the outcome, whether it be successful or otherwise. Whenever a new case is encountered, the system adapts its stored behavior to fit the new circumstances. Case-based reasoning is discussed in further detail.
(vii) Explorative or unsupervised learning. Rather than having an explicit goal, an explorative system continuously searches for patterns and relationships in the input data, perhaps marking some patterns as interesting and warranting further investigation. Examples of the use of unsupervised learning include:

- data mining, where patterns are sought among large or complex data sets;
- identifying clusters, possibly for compressing the data;
- learning to recognize fundamental features, such as edges, from pixel images;
- Designing products, where innovation is a desirable characteristic.

In rote learning and learning from advice, the sophistication lies in the ability of the teacher rather than the learning system. If the teacher is a human expert, these two techniques can provide an interactive means of eliciting the expert's knowledge in a suitable form for addition to the knowledge base. However, most of the interest in symbolic learning has focused on case-based reasoning, described in more detail below. Reasoning by analogy is similar to case-based reasoning, while many of the problems and solutions associated with learning by induction also apply to the other categories of symbolic learning.

## Case Based Reasoning

A characteristic of human intelligence is the ability to recall previous experience whenever a similar problem arises. This is the essence of case-based reasoning (CBR). As Riesbeck and Schank (1989) put it, a case-based reasoner solves new problems by adapting solutions that were used to solve old problems.

Consider the example of diagnosing a fault in a refrigerator. If an expert system has made a successful diagnosis of the fault, given a set of symptoms, it can file away this information for future use. If the expert system is subsequently presented with details of another faulty refrigerator of exactly the same type, displaying exactly the same symptoms in exactly the same circumstances, then the diagnosis can be completed simply by recalling the previous solution. However, a full description of the symptoms and the environment would need to be very detailed, and it is unlikely to be reproduced exactly. What we need is the ability to identify a previous case, the solution of which can be modified to reflect the slightly altered circumstances, and then saved for future use. Aamodt and Plaza (1994) have therefore proposed that CBR can be described by a four-stage cycle:

- retrieve the most similar case(s);
- reuse the case(s) to attempt to solve the problem;
- revise the proposed solution if necessary;
- retain the new solution as a part of a new case.

Such an approach is arguably a good model of human reasoning. Indeed case-based reasoning is often used in a semi-automated manner, where a human can intervene at any stage in the cycle.

## 4. Intelligent Agents

### Characteristics of an Intelligent Agent

Agent-based technologies have been growing apace, both within the world of AI and in more general software engineering. One motivation has been the rapid escalation in the quantity of information available. Software assistants-or agents-are needed to take care of specific tasks for us. For example, much of the trading on the world's stock exchanges is performed by agents that can react quickly to minor price fluctuations.

While noting that not all agents are intelligent, Wooldridge (1997) gives the following definition for an agent:

An agent is an encapsulated computer system that is situated in some environment, and that is capable of flexible, autonomous action in that environment in order to meet its design objectives.

From this definition we can see that the three key characteristics of an agent are autonomy, persistence, and the ability to interact with its environment. Autonomy refers to an agent's ability to make its own decisions based on its own expertise and circumstances, and to control its own internal state and behavior. The definition implies that an agent functions continuously within its environment, i.e., it is persistent over time. Agents are also said to be situated, i.e., they are responsive to the demands of their environment and are capable of acting upon it. Interaction with a physical environment requires perception through sensors, and action through actuators or effectors. Interaction with a purely software environment requires only access to and manipulation of data and programs.

We might reasonably expect an intelligent agent to be all of the following:

- reactive,
- goal-directed,
- adaptable,
- socially capable.

Social capability refers to the ability to cooperate and negotiate with other agents (or humans). It is quite easy to envisage an agent that is purely reactive, e.g., one whose only role is to place a warning on your computer screen when the printer has run out of paper. Likewise, modules of conventional computer code can be thought of as goal-directed in the limited sense that they have been programmed to perform a specific task regardless of their environment. Since it is autonomous, an intelligent agent can decide its own goals and choose its own actions in pursuit of those goals. At the same time, it must also be able to respond to unexpected changes in its environment. It, therefore, has to balance reactive and goal-directed behavior, typically through a mixture of problem solving, planning, searching, decision making, and learning through experience.

## 5. Computational Intelligence

WEKA (Waikato Environment for Knowledge Analysis): The Software

### Overview

An exciting and potentially far-reaching development in computer science and Artificial Intelligence is the invention and application of methods of machine learning. These enable a computer program to automatically analyze a large body of data and decide what information is most relevant. This crystallized (clustered, sorted or classified) information can then be used to automatically make predictions or to help people make decisions faster and more accurately.

### Goal

The overall goal is to build a state-of-the-art facility for developing machine learning (ML) techniques and to apply them to real-world data mining problems. Several standard ML techniques were incorporated into a software "workbench" called WEKA, for Waikato Environment for Knowledge Analysis. With WEKA, a specialist in a particular field is able to use ML to derive useful knowledge from databases that are far too large to be analyzed by hand. WEKA's users are ML researchers and industrial scientists, but it is also widely used for teaching (Schools of higher learning).

### Objectives:

i. Make Machine Learning (ML) techniques generally available;
ii. Apply them to practical problems that matter to New Zealand industry;
iii. Develop new machine learning algorithms and give them to the world;
iv. Contribute to a theoretical framework for the field.

WEKA machine learning package is publically available and presents a collection of algorithms for solving real-world data mining problems. The software is written entirely in Java (distributed under the GNU Public License) and includes a uniform interface to a number of standard ML techniques.

### Main features

i. Comprehensive set of data pre-processing tools, learning algorithms and evaluation methods.
ii. Graphical user interfaces (incl. data visualization)
iii. Environment for comparing learning algorithms

### The Explorer

As an explorer, WEKA can be functional in the follow;
i. Classification and Regression
ii. Clustering
iii. Association Rules
iv. Attribute Selection
v. Data Visualization

### History

WEKA was developed by Machine Learning Project Team at the University of Waikato in New Zealand funded by the New Zealand Government since 1993.

Late 1992 - Funding was applied for by Ian Witten
1993 - Development of the interface and infrastructure
WEKA acronym coined by Geoff Holmes
WEKA's file format "ARFF" was created by Andrew Donkin
ARFF was rumored to stand for **A**ndrew's **R**idiculous **F**ile **F**ormat

Sometime in 1994 - first internal release of WEKA
TCL/TK user interface + learning algorithms written mostly in C. It is very much beta software.

Changes for the b1 release included (among others):

"Ambiguous and Unsupported menu commands removed."
"Crashing processes handled (in most cases :-)"
October 1996 - first public release: WEKA 2.1
July 1997 – WEKA 2.2
Schemes: 1R, T2, K*, M5, M5Class, IB1-4, FOIL, PEBLS, support for C5
Included a facility (based on UNIX makefiles) for configuring and running large scale experiments
Early 1997 - decision was made to rewrite WEKA in Java
Originated from code written by Eibe Frank for his PhD
Originally codenamed **JAWS** (**JA**va **W**eka **S**ystem)
May 1998 - WEKA 2.3
Last release of the TCL/TK-based system
Mid 1999 - WEKA 3 (100% Java) released
Version to complement the Data Mining book
Development version (including GUI)

WEKA Versions:

There are several versions of WEKA:

WEKA 3.4: "book version" compatible with description in data mining book

WEKA 3.5.5: "development version" with lots of improvements

### Projects based on WEKA

I. 45 projects currently (30/01/07) listed on the WekaWiki
II. Incorporate/wrap WEKA
   a. GRB Tool Shed - a tool to aid gamma ray burst research
   b. YALE - facility for large scale ML experiments
   c. GATE - NLP workbench with a WEKA interface
   d. Judge - document clustering and classification
   e. RWeka - an R interface to Weka
III. Extend/modify WEKA
   a. BioWeka - extension library for knowledge discovery in biology
   b. WekaMetal - metal learning extension to WEKA
   c. Weka-Parallel - parallel processing for WEKA
   d. Grid Weka - grid computing using WEKA
   e. Weka-CG - computational genetics tool library

### Limitations of WEKA

- Traditional algorithms need to have all data in main memory.
- Big datasets are an issue.

### Solution

- Incremental schemes; having the datasets in several schemes or sizes.
- Stream algorithms; the use of MOA "Massive Online Analysis" (Coincidentally, Mao is not only a streaming algorithm but a flightless bird which also is extinct!)

### Symbolic Learning Methods in WEKA

- **ID3:** uses Information Gain heuristic which is based on Shannon's entropy to build efficient decision trees. But one disadvantage with ID3 is that it over fits the training data. So, it gives rise to decision trees which are too specific and hence this approach is not noise resistant when tested on novel examples. Another disadvantage is that it cannot deal with missing attributes and requires all attributes to have nominal values. Also, it can be run only on datasets where all the attributes are nominal.

- **C4.5:** is an improved version of ID3 which prevents over-fitting of training data by pruning the decision tree when required, thus making it more noise resistant.

- **J48:** J48 (Quinlan, 1992) implements Quinlan's C4.5 algorithm (Quinlan, 1993) for generating a pruned or unpruned C4.5 decision tree. C4.5 is an extension of Quinlan's earlier ID3 algorithm. The decision trees generated by J48 can be used for classification. J48 builds decision trees from a set of labeled training data using the concept of information entropy. It uses the fact that each attribute of the data can be used to make a decision by splitting the data into smaller subsets. J48 examines the normalized information gain (difference in entropy) that results from choosing an attribute for splitting the data. To make the decision, the attribute with the highest normalized information gain is used. Then the algorithm recurs on the smaller subsets. The splitting procedure stops if all instances in a subset belong to the same class. Then a leaf node is created in the decision tree telling to choose that class. But it can also happen that none of the features give any information gain. In this case J48 creates a decision node higher up in the tree using the expected value of the class. J48 can handle both continuous and discrete attributes, training data with missing attribute values and attributes with differing costs. Further it provides an option for pruning trees after creation.

### Neural Network Method in WEKA

**a) Multi-Layer Perceptron (MLP):** Multilayer Perceptron is a layered network comprising of input nodes, hidden nodes and output nodes. The error values are back propagated from the output nodes to the input nodes via the hidden nodes. Considerable time is required to build a neural network but once it is done, classification is quite fast. Neural networks are robust to noisy data as long as too many epochs are not considered since they do not over fit the training data. In

Weka, nominal attributes are automatically converted to numeric ones for neural network learning methods. So, preprocessing is not required in this type of datasets. Numeric Datasets are those which contain few nominal and few numeric attributes.

**Differences between Symbolic Learning and Neural Network**

The most often application of both neural network and symbolic learning systems is the inductive (the system is presented with sets of example data and is told the correct conclusions that it should draw from each) acquisition of concepts from examples. The system continually refines its rules and relations so as to correctly handle each new example. (Hopgood, 2002)

1 Symbolic learning describes systems that formulate and modify rules, facts, and relationships, explicitly expressed in words and symbols. In other words, they create and modify their own knowledge base, the system constructs a decision tree from a set of training objects; hence they are knowledge based systems while the Numerical learning refers to systems that use numerical models; learning in this context refers to techniques for optimizing the numerical parameters such as weights and bias with back propagated error (The error values are back propagated from the output nodes to the input nodes via the hidden nodes). They are computational methods in Artificial intelligence (AI).

2 Using algorithms that construct decision trees for symbolic methods and for networks the use of back propagation to determine appropriate weights as in Table 1 (Results extracted from Weiss and Kapouleas) (Quinlan), it was discovered that both methods performed well but Neural networks took more CPU time in Thyroid domain training.

3 In all, Back propagation (Neural Network) usually requires a great deal more computation compared to Decision Tree (Symbolic Learning). (Quinlan; Shravya)

4 However in general, the predictive accuracy of both approaches is roughly the same with back propagation often slightly more accurate. (Quinlan; Shravya)

5 ID3 (symbolic learning) is that it over fits the training data. So, it gives rise to decision trees which are too specific and hence this approach is not noise resistant when tested on novel (new) examples though it was corrected in C4.5 which prevents over-fitting of training data by pruning the decision tree when required. Another disadvantage is that it cannot deal with missing attributes and requires all attributes to have nominal values while Neural Networks are robust to noisy data as long as too many epochs are not considered since they do not over fit the training data. In other words, Back propagation is more adaptive in a noisy datasets but Symbolic learning tends to perform better in a noise free datasets. (Shravya)

## 6. Validation

- **Percentage Split (Holdout Method)**: It is the simplest kind of cross validation. The data set is separated into two sets, called the training set and the testing set. The function approximator fits a function using the training set only. Then the function approximator is asked to predict the output values for the data in the testing set (it has never seen these output values before). The errors it makes are accumulated as before to give the mean absolute test set error, which is used to evaluate the model. The advantage of this method is that it is usually preferable to the residual method and takes no longer to compute. However, its evaluation can have a high variance. The evaluation may depend heavily on which data points end up in the training set and which end up in the test set, and thus the evaluation may be significantly different depending on how the division is made. In general, the data will be split up randomly into training data and test data. In the experiments conducted, the data will be split such that training data comprises 66% of the entire data and the rest is used for testing. (Shravya, Schneider, 1997)

Holdout (Percentage split in WEKA) validation is not strictly cross-validation, because the data never are crossed over. Observations are chosen randomly from the initial sample to form the testing data, and the remaining observations are retained as the training data. Normally, less than a third of the initial sample is used for testing data. (Martin, 2009)

- **K-fold Cross-validation**: In general, is one way to improve over the holdout method. The data is split into k disjoint subsets and one of it is used as testing data and the rest of them are used as training data. This is continued till every subset has been used once as a testing dataset. In other words, the data set is divided into k subsets, and the holdout method is repeated k times. Each time, one of the k subsets is used as the test set and the other k-1 subsets are put together to form a training set. Then the average error across all k trials is computed. The advantage of this method is that it matters less how the data gets divided. Every data point gets to be in a test set exactly once, and gets to be in a training set k-1 times. The variance of the resulting estimate is reduced as k is increased. The disadvantage of this method is that the training algorithm has to be rerun from scratch k times, which means it takes k times as much computation to make an evaluation. A variant of this method is to randomly divide the data into a test and training set k different times. The advantage of doing this is that you can independently choose how large each test set is and how many trials you average over. In the experiments conducted, 5-fold cross validation was done. (Shravya, Schneider, 1997)

## 7. Artificial Neural Networks (Ann)

Artificial Neural Networks are a programming paradigm that seek to emulate the microstructure of the brain, and are used extensively in artificial intelligence problems from simple pattern-recognition tasks, to advanced symbolic manipulation. (Noriega, 2005)

The Multilayer Perceptron is an example of an artificial neural network that is used extensively for the solution of a number of different problems, including pattern recognition and interpolation. It is a development of the Perceptron neural network model, that was originally developed in the early 1960s but found to have serious limitations.

An Artificial Neural Network (ANN) consists of a collection of processing units called neurons that are highly interconnected according to a given topology. ANNs have the ability of learning-by-example and generalization from limited, noisy, and incomplete data. They have been successfully employed in a broad spectrum of data-intensive applications (Xiaonan et al, 2008). In this section, we will review their contributions and performance on intrusion detection domain. This section is organized by the types of ANNs illustrated in fig below

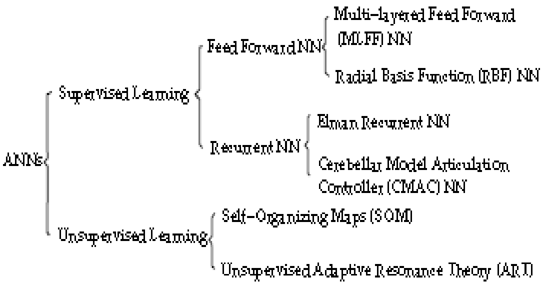

ANNs
Supervised Learning
Feed Forward NN
Multi-layered Feed Forward (MLFF) NN
Radial Basis Function (RBF) NN
Recurrent NN
Elman Recurrent NN
Cerebellar Model Articulation Controller (CMAC) NN
Unsupervised Learning
Self-Organizing Maps (SOM)
Unsupervised Adaptive Resonance Theory (ART)

**Figure 4:** ANN Hierarchy

**History and Theoretical Background** (Noriega, 2005)

### Biological Basis of Neural Networks

Artificial Neural Networks attempt to model the functioning of the human brain. The human brain for example consists of billions of individual cells called neurons. It is believed by many (the issue is contentious) that all knowledge and experience is encoded by the connections that exist between neurons. Given that the human brain consists of such a large number of neurons (so many that it is impossible to count them with any certainty), the quantity and nature of the connections between neurons is, at present levels of understanding, almost impossible to assess.

### Understanding the Neuron

Intelligence is arguably encoded at the connections between neurons (the synapses), but before examining what happens at these connections, we need to understand how the neuron functions.

Modern computers use a single, highly complex processing unit (eg. Intel Pentium) which performs a large number of different functions. All of the processing on a conventional computer is handled by this single unit, which processes commands at great speed.

The human brain is different in that it has billions of simple processing units (neurons). Each of these units is slow when compared to say a Pentium 4, but only ever performs one simple task. A neuron activates (fires) or remains inactive. One may observe in this a kind of binary logic, where activation may be denoted by a '1' and inactivation by a '0'. Neurons can be modeled as simple switches therefore, the only problem remains in understanding what determines whether a neuron fires.

Neurons can be modeled as simple input-output devices, linked together in a network. Input is received from neurons found lower down a processing chain, and the output transmitted to neurons higher up the chain. When a neuron fires, information is passed up the processing chain. This innate simplicity makes neurons fairly straightforward entities to model; it is in modeling the connections that the greatest challenges occur.

### Understanding the Connections (Synapses)

When real neurons fire or are activated, they transmit chemicals (neurotransmitters) to the next group of neurons up the processing chain alluded to in the previous subsection. These neurotransmitters form the input to the next neuron, and constitute the messages neurons send to each other. These messages can assume one of three different forms.

- Excitation - Excitatory neurotransmitters increase the likelihood of the next neuron in the chain to fire.
- Inhibition - Inhibitory neurotransmitters decrease the likelihood of the next neuron to fire.
- Potentiation - Adjusting the sensitivity of the next neurons in the chain to excitation or inhibition (this is the learning mechanism).

If we can model neurons as simple switches, we model connections between neurons as matrices of numbers (called weights), such that positive weights indicate excitation, negative weights indicate inhibition. How learning is modelled depends on the paradigm used.

### Modelling Learning

Using artificial neural networks it is impossible to model the full complexity of the brain of anything other than the most basic living creatures, and generally ANNs will consist of at most a few hundred (or few thousand) neurons, and very limited connections between them. Nonetheless quite small neural networks have been used to solve what have been quite difficult computational problems.

Generally Artificial Neural Networks are basic input and output devices, with the neurons organized into layers. Simple Perceptrons consist of a layer of input neurons, coupled with a layer of output neurons, and a single layer of weights between them, as shown in Figure 5.

The learning process consists of finding the correct values for the weights between the input and output layer. The schematic representation given in Figure 5 is often how neural nets are depicted in the literature, although mathematically it is useful to think of the input and output layers as vectors of values(I and O respectively), and the weights as a matrix.

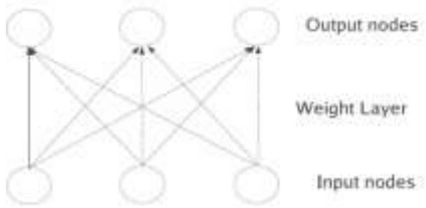


**Figure 5:** Simple Perceptron Architecture

We define the weight matrix Wio as an i X o matrix, where i is the number of input nodes, and o is the number of output nodes. The network output is calculated as follows.

$$O - f(IWio) \qquad (1)$$

Generally data is presented at the input layer, the network then processes the input by multiplying it by the weight layer. The result of this multiplication is processed by the output layer nodes, using a function that determines whether or not the output node fires.

The process of finding the correct values for the weights is called the learning rule, and the process involves initializing the weight matrix to a set of random numbers between -1 and +1. Then as the network learns, these values are changed until it has been decided that the network has solved the problem. Finding the correct values for the weights is effected using a learning paradigm called supervised learning. Supervised learning is sometimes referred to as training.

Data is used to train the network; this constitutes input data for which the correct output is known. Starting with random weights, an input pattern is presented to the network; it makes an initial guess as to what the correct output should be.

During the training phase, the difference between the guess made by the network and the correct value for the output is assessed, and the weights are changed in order to minimize the error. The error minimization technique is based on traditional gradient descent techniques. While this may sound frighteningly mathematical, the actual functions used in neural networks to make the corrections to the weights are chosen because of their simplicity, and the implementation of the algorithm is invariably uncomplicated.

### The Activation Function

The basic model of a neuron used in Perceptrons and MLPs is the McCulloch-Pitts model, which dates from the late 1940s. This modeled a neuron as a simple threshold function.

$$f(x) = \begin{cases} 1 & x > o \\ 0 \ otherwise \end{cases} \qquad (2)$$

This activation function was used in the Perceptron neural network model, and as can be seen this is a relatively straightforward activation function to implement.

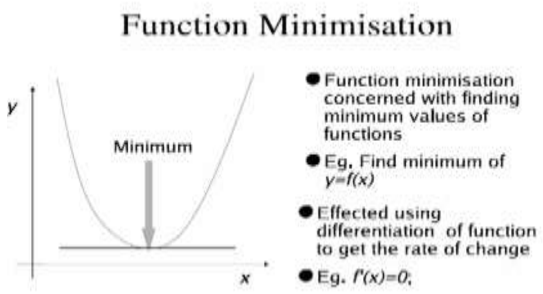
### The Learning Rule

The perceptron learning rule is comparatively straightforward. Starting with a matrix of random weights, we present a training pattern to the network, and calculate the network output. We determine an error function E

$$E(O) = (T - O) \qquad (3)$$

where in this case, T is the target output vector for a training input to the ANN. In order to determine how the weights should change, this function has to minimize. What this means is find the point at which the function reaches its minimum value. The assumption we make about the error function is that if we were to plot all of its potential values into a graph, it would be shaped like a bowl, with sides sloping down to a minimum value at the bottom.

In order to find the minimum values of a function differentiation is used. Differentiation is used to give the rate at which functions change, and is often defined as the tangent on a curve at a particular point. If our function is perfectly bowl shaped, then there will only be one point at which the minimum value of a function has a tangent of zero (i.e. have a perfectly at tangent), and that is at its minimum point (see Figure 6.)


Function Minimisation

- Function minimisation concerned with finding minimum values of functions
- Eg. Find minimum of y=f(x)
- Effected using differentiation of function to get the rate of change
- Eg. f'(x)=0;

**Figure 6:** Function Minimization using Differentiation

In neural network programming the intention is to assess the effect of the weights on the overall error function. We can take Equation 3 and combine it with Equation 1 to obtain the following.

$$E(O) = (T - O) = T - f(IWio) \qquad (4)$$

We then differentiate the error function with respect to the weight matrix. The discussion on Multilayer Perceptrons will look at the issues of function minimization in greater detail. Function minimization in the Simple Perceptron Algorithm is very straightforward. We consider the error each individual output node, and add that error to the weights feeding into that node. The Perceptron learning algorithm works as follows.

1. Initialize the weights to random values on the interval [-1, 1].
2. Present an input pattern to the network.
3. Calculate the network output.
4. for each node n in the output layer...
(a) Calculate the error $E_n = T_n - O_n$
(b) Add $E_n$ to all of the weights that connect to node n (add $E_n$ to column n of the weight matrix.
5. Repeat the process from 2 for the next pattern in the training set.

This is the essence of the perceptron algorithm. It can be shown that this technique minimizes the error function. In its current form it will work, but the time taken to converge to a solution (i.e. the time taken to find the minimum value) may be unpredictable because adding the error to the weight matrix is something of a 'blunt instrument' and results in the weights gaining high values if several iterations are required to obtain a solution. This is akin to taking large steps around the bowl in order to find the minimum value; if smaller steps are taken we are more likely to find the bottom.

In order to control the convergence rate, and reduce the size of the steps being taken, a parameter called a learning rate is used. This parameter is set to a value that is less than unity, and means that the weights are updated in smaller steps (using a fraction of the error). The weight update rule becomes the following.

$$Wio(t+1) = Wio(t) + \epsilon En \qquad (5)$$

which means that the weight value at iteration t+1 of the algorithm is equivalent to a fraction of the error $\epsilon En$ added to the weight value at iteration t.

**Supervised Learning** (Xiaonan et al, 2008)

Feed Forward Neural Networks: Feed forward neural networks are the first and arguably the simplest type of artificial neural networks devised. Two types of feed forward neural networks are commonly used in modeling either normal or intrusive patterns. Multi-layered Feed Forward (MLFF) Neural Networks MLFF networks use various learning techniques, the most popular being back-propagation (MLFF-BP). Network traffic is another indispensable data source.

MLFF-BP can also be used as a multi-class classifier (MCC). MCC neural networks can either have multiple output neurons or assemble multiple binary neural network classifiers. Apparently, the latter is more flexible than the former when facing a new class. Except for the BP learning algorithm, there are many other learning options in MLFF networks. S. Mukkamala and A. H. Sung (2003) compared 12 different learning algorithms on the KDD99 dataset, and found that resilient back propagation achieved the best performance among the neural networks in terms of accuracy (97.04%) and training time (67 epochs).

Radial Basis Function Neural Networks: Radial Basis Function (RBF) neural networks are another widely used type of feed forward neural networks. Since they perform classification by measuring the distances between inputs and the centers of the RBF hidden neurons, they are much faster than time consuming back-propagation, and more suitable for problems with large sample size [S. Cayzer and J. Smith, 2006]. Other than being a classifier, the RBF network was also used to fuse results from multiple classifiers [S. Cayzer and J. Smith, 2006]. It outperformed five different decision fusion functions, such as Dempster-Shafer combination and Weighted Majority Vote.

Comparison between MLFF-BP and RBF networks Since RBF and MLFF-BP networks are widely used, a comparison between them is naturally required. [Jiang et al, 2003] and [Zhang et al, 2003] compared the RBF and MLFF-BP networks for misuse and anomaly detection on the KDD99 dataset. Their experiments have shown that for misuse detection, BP has a slightly better performance than RBF in terms of detection rate and false positive rate, but requires longer training time, while for anomaly detection, the RBF network improves the performance with a high detection rate and a low false positive rate, and requires less training time (cutting it down from hours to minutes). All in all, RBF networks achieve better performance.

Another interesting comparison has been made between the binary and decimal input encoding schemes of MLFFBP and RBF [Liu et al, 2002]. The results showed that binary encoding has lower error rates than decimal encoding, because decimal encoding only computes the frequency without considering the order of system calls. Decimal encoding, however, handles noise well and the classifiers can be trained with fewer data. Furthermore, there are fewer input nodes in decimal encoding than in binary encoding, which decreases the training and testing time and simplifies the network structure.

**Unsupervised Learning** (Xiaonan et al, 2008)

Self-Organizing Maps and Adaptive Resonance Theory are two typical unsupervised neural networks. Similar to statistical clustering algorithms, they group objects by similarity. They are suitable for intrusion detection tasks in that normal behavior is densely populated around one or two centers, while abnormal behavior and intrusions appear in sparse regions of the pattern space outside of normal clusters.

Self-Organizing Maps: Self-organizing maps (SOM), also known as Kohonen maps, are single-layer feed forward networks where outputs are clustered in a low dimensional (usually 2D or 3D) grid. It preserves topological relationships of input data according to their similarity. SOM can function as a data pre-processor to cluster input data. Other classification algorithms, such as feed forward neural networks, were trained using the outputs from the SOM. Sometimes SOMs map data from different classes into one neuron. SOMs are the most popular neural networks to be trained for anomaly detection tasks; an example is a proposed multi-layer detection framework, where the first layer used a SOM to cluster the payload, and compressed it into a single feature.

Unlike other unsupervised approaches, SOMs are useful to visualize the analysis which took advantage of topology-preserving and dimensionality reducing properties of SOMs. Although SOM shows very high accuracy in usage, the training procedure suffers from a high computational overhead, especially when the size of the training set is over 10,000.

Adaptive Resonance Theory (ART): The Adaptive Resonance Theory (ART) embraces a series of neural network models that perform unsupervised or supervised learning, pattern recognition, and prediction, since it has been invented by Stephen Grossberg in 1976. Unsupervised learning models include ART-1, ART-2, ART-3, and Fuzzy ART. Various supervised ones are named with the suffix “MAP”, such as ARTMAP, Fuzzy ARTMAP, and Gaussian ARTMAP. Compared with SOMs who cluster data objects based on the absolute distance, ARTs cluster objects based on the relative similarity of input patterns to the weight vector.

In comparing the performance of ARTs and SOMs, the results showed that ART nets have better intrusion detection performance than SOMs on either offline or online data in Intrusion Detection.

Fuzzy ART nets combine fuzzy set theory and adaptive resonance theory. This combination is faster and more stable than ART nets alone in responding to arbitrary input sequences. Liao et al (2007) and Durgin et al (2005) are two examples of using Fuzzy ART to detect anomalies. Liao et al: deployed Fuzzy ART in an adaptive learning framework which is suitable for dynamic changing environments. Normal behavior changes are efficiently accommodated while anomalous activities can still be identified. Durgin et al (2005) investigated in detail the capabilities of SOMs and Fuzzy ARTs. Both SOMs and Fuzzy ARTs show promise in detecting network abnormal behavior. The sensitivity of Fuzzy ARTs seems to be much higher than that of SOMs.

**Summary**

All these research works took advantage of ANNs’ ability to generalize from limited, noisy, and incomplete data. Some researchers attempted to address disadvantages of ANNs as well such as long training time and retraining.

To further correct some of the disadvantages, the following practice has been proven useful in ANNs:

- Datasets and features. Neural networks only recognize whatever is fed to them in the form of inputs. Although they have the ability of generalization, they are still unable to recognize unseen patterns sometimes. One cause of this difficulty is incomplete training sets. To address this problem, randomly generated anomalous inputs are inserted into the training set with the purpose of exposing the network to more patterns, hence making the training sets more complete. Selecting good feature sets is another way to improve performance.

## 8. Fuzzy Sets

The past decades have witnessed a rapid growth in the number and variety of applications of fuzzy logic. Fuzzy logic, dealing with the vague and imprecise, is appropriate for intrusion detection for two major reasons. First, the intrusion detection problem involves many numeric attributes in collected audit data, and various derived statistical measures. Building models directly on numeric data causes high detection errors. For example, an intrusion that deviates only slightly from a model may not be detected or a small change in normal behavior may cause a false alarm. Second, the security itself includes fuzziness, because the boundary between the normal and anomaly is not well defined.

**Genetic Algorithm** (Mahmud et al., 2009)

Genetic algorithm (GA) is an adaptive heuristic search method for solving optimization problems. It was formally introduced in the United States in the 1970s by John Holland at the University of Michigan (Goldberg, 1989). They have a solid basis in genetics and evolutionary biological systems. Genetic Algorithms comprise a kind of effective searching and optimizing technique that outperforms most of traditional methods. In particular, GAs work very well on combinatorial problems such as reduct finding in rough set theory. Furthermore, finding the minimal reducts is a NP-hard problem [Abraham et al, 2007b]. Hence, GA is a good candidate as a methodology for finding minimal reducts.

In classical GA, individuals are encoded as binary strings of the attributes ((e.g. 0100110100 ⇔{a2; a5; a6; a8}). Each individual represents a set of attributes generated by mutation, crossover and selection procedures using some fitness criteria. Individuals with maximal fitness are highly probable to be reducts but there is no full guarantee.

Parallel Genetic Algorithm, PGA was first attempted by Grefenstette. Parallelism refers to many processors, with distributed operational load. Each GA is a good candidate for parallelization. Processor may independently work with different parts of a search space and evolve new generations in parallel. This helps to find out the optimum solution for the complex problems by searching massive populations and increases quality of the solutions by overcoming premature convergence. There are many types of Parallel Genetic Algorithm taxonomies [Abraham ett al, 2007a]. One of the most ingenious taxonomies is the Island Model (IM) [Abraham et al, 2007b], where processors are globally controlled by message passing within Master-Slave architecture. Master processor sends "START" signal to the slave processors to start generations and continue sending "MIGRATION" message to partially exchange the best chromosomes between the processors. So the worst chromosomes are replaced by the best received ones. Time between two consecutive MIGRATION signals is called the migration step; percentage of the best chromosomes is called migration percentage. Migrations should occur after a time period long enough for allowing development of good characteristics in each subpopulation.

## 9. Simulated Annealing

**Definition:** A technique to finding a good solution to an optimization problem by trying random variations of the current solution is called Simulated Annealing. A worse variation is accepted as the new solution with a probability that decreases as the computation proceeds. The slower the cooling schedule, or rate of decrease, the more likely the algorithm is to find an optimal or near-optimal solution.

**History**

A simple Monte Carlo simulation samples the possible states of a system by randomly choosing new parameters. At the end of the simulation, the collection, or ensemble, of randomly chosen points in search space gives you information about this space.

For example, the web page Simple Monte Carlo Simulation gives an example of a unit square containing one-quarter of a unit circle whose center is in the lower left corner. The search space is the unit square, and any point in this space can be in one of two possible states; inside of the quarter-circle, or outside. Each point in the search space is determined by the value of two parameters, its x- and y-coordinate. The possible values for each parameter can be any real number in the range [0.0, 1.0]. Each step in the simulation consists of choosing random, allowed values for both of the parameters. This generates a point in the search space that is associated with one of the two states. At the end of the simulation, there will be an ensemble of N points, of which $N_{in}$ are inside of the quarter-circle. The ratio of $N_{in}$ to N is just the ratio of the area inside the quarter-circle to the area of the unit square.

Therefore, a simple Monte Carlo simulation randomly selects a point somewhere is the search space and all points are used to find out information about the search space. This procedure has use in some problems, like the one described above for finding the area of certain regions, but does not give physically realistic results when the search space represents an energy surface. For example, assume that the simulation studies a collection of M helium atoms in a cube. The position of each atom is described by three parameters that give its coordinates within the cube. The energy of this system is given by the sum of all pair-wise interaction energies. If you wanted to calculate the average energy of this system, a simple Monte Carlo simulation should not be used. This is because a random placement of the M atoms may, at some point of the simulation, place two of the atoms so close together that their interaction energy is virtually infinite. This adds an infinite energy to the ensemble of atom distributions and produces an infinite average energy. In the real world, two helium atoms would never get that close together. Therefore, a modification to the simple Monte Carlo simulation needs to be made so that unrealistic samples are not placed into the ensemble. Such a modification was proposed in 1953 by Nicholas Metropolis and coworkers (Metropolis, 1953). This modified procedure is known as a Metropolis Monte Carlo simulation.

In contrast with the simple Monte Carlo simulation, a new point in search space is sampled by making a slight change to the current point. In the example used here, a new orientation of the helium atoms is created by making a random, small change to each atom's coordinates. If the energy of this new orientation is less than that of the old, this orientation is added to the ensemble. If the energy rises, a Boltzmann acceptance criterion is used. If the energy rise is small enough, the new orientation is added to the ensemble. Conversely, if the energy rise is too large, the new orientation is rejected and the old orientation is again added to the ensemble (see Metropolis Monte Carlo Simulation for more details).

By using this acceptance probability one can prove that the average of any property, such as the energy, over the ensemble is equal to the Boltzmann average of this property as determined by the Boltzmann Distribution Law, for a sufficiently large ensemble. What is unique about this Boltzmann acceptance probability is that the temperature of the system must be used. Therefore, the Boltzmann average of a property is the expected value of this property at the given temperature.

In 1983, Kirkpatrick and coworkers (1983) proposed a method of using a Metropolis Monte Carlo simulation to find the lowest energy (most stable) orientation of a system. Their method is based upon the procedure used to make the strongest possible glass. This procedure heats the glass to a high temperature so that the glass is a liquid and the atoms can move relatively freely. The temperature of the glass is slowly lowered so that at each temperature the atoms can move enough to begin adopting the most stable orientation. If the glass is cooled slowly enough, the atoms are able to "relax" into the most stable orientation. This slow cooling process is known as annealing, and so their method is known as Simulated Annealing.

### Simulated Annealing (Algorithm)

A Simulated Annealing optimization starts with a Metropolis Monte Carlo simulation at a high temperature. This means that a relatively large percentage of the random steps that result in an increase in the energy will be accepted. After a sufficient number of Monte Carlo steps, or attempts, the temperature is decreased. The Metropolis Monte Carlo simulation is then continued. This process is repeated until the final temperature is reached.

A Simulated Annealing program consists of a pair of nested DO-loops. The outer-most loop sets the temperature and the inner-most loop runs a Metropolis Monte Carlo simulation at that temperature. The way in which the temperature is decreased is known as the cooling schedule. In practice, two different cooling schedules are predominantly used; a linear cooling schedule ($T_{new}=T_{old}-dT$) and a proportional cooling schedule ($T_{new}=C\times T_{old}$) where C<1.0. These are not the only possible cooling schedules; they are just the ones that appear the most in the literature.

As described in more detail in the discussion of a Metropolis Monte Carlo simulation, a more difficult aspect is to determine who long to run this simulation at each temperature. This depends upon the maximum size of the Monte Carlo step at each temperature. While a pure Metropolis Monet Carlo simulation attempts to reproduce the correct Boltzmann distribution at a given temperature, the inner-loop of a Simulated Annealing optimization only needs to be run long enough to explore the regions of search space that should be reasonably populated. This allows for a reduction in the number of Monte Carlo steps at each temperature, but the balance between the maximum step size and the number of Monte Carlo steps is often difficult to achieve, and depends very much on the characteristics of the search space or energy landscape.

Simulated annealing has been used in various combinatorial optimization problems and has been particularly successful in circuit design problems (see Kirkpatrick et al. 1983).

## 10. Benefits of AI

This chapter has reviewed a range of AI techniques. Whether the resultant systems display true intelligence remains questionable. Nevertheless, the following practical benefits have stemmed from the development of AI techniques:

**Reliability and Consistency**: An AI system makes decisions that are consistent with its input data and its knowledge base (for a knowledge-based system) or numerical parameters (for a computational intelligence technique). It may, therefore, be more reliable than a person, particularly where repetitive mundane judgments have to be made.

**Automation**: In many applications, such as visual inspection on a production line, judgmental decision-making has to be performed repeatedly. A well-designed AI system ought to be able to deal with the majority of such cases, while highlighting any that lie beyond the scope of its capabilities. Therefore, only the most difficult cases, which are normally the most interesting, are deferred to a person.

**Speed**: AI systems are designed to automatically make decisions that would otherwise require human reasoning, judgment, expertise, or common sense. Any lack of true intelligence is compensated by the system's processing speed. An AI system can make decisions informed by a wealth of data and information that a person would have insufficient time to assimilate.

**Improved Domain Understanding**: The process of constructing a knowledge-based system requires thedecision-making criteria to be clearly identified and assessed. This process frequently leads to a better understanding of the problem being tackled. Similar benefits can be obtained by investigating the decision-making criteria used by the computational intelligence techniques.

**Knowledge Archiving**: The knowledge base is a repository for the knowledge of one or more people. When these people move on to new jobs, some of their expert knowledge is saved in the knowledge base, which continues to evolve after their departure.

**New Approaches to Software Engineering**: Since AI systems are supposed to be flexible and adaptable, development is usually based upon continuous refinements of an initial prototype. This is the prototype–test–refine cycle, which applies to both knowledge-based systems and computational intelligence techniques.

The key stages in the development of a system are:

- decide the requirements;
- design and implement a prototype;
- continuously test and refine the prototype.

Rapid prototyping and iterative development have gained respectability across most areas of software engineering in recent years, replacing the traditional linear "waterfall process" of meticulous specification, analysis, and design phases prior to implementation and testing.

## 11. Applications of AI

Some approaches are pre-specified and structured, while others specify only low-level behavior, leaving the intelligence to emerge through complex interactions. Some approaches are based on the use of knowledge expressed in words and symbols, whereas others use only mathematical and numerical constructions.

Overall, the tools and techniques of AI are ingenious, practical, and useful. If these were the criteria by which the successes of AI were measured, it would be heralded as one of the most accomplished technological fields. However, human mental faculties are incredibly complex and have proved to be extremely difficult to mimic.

Nevertheless, the techniques presented here have undoubtedly advanced humankind's progress towards the construction of an intelligent machine. AI research has made significant advances from both ends of the intelligence spectrum but a gap still exists in the middle. Building a system that can make sensible decisions about unfamiliar situations in everyday, non-specialist domains remains difficult. This development requires progress in simulating behaviors that humans take for granted—specifically perception, language, common sense, and adaptability. Some of the areas where AI has been successfully applied are as follow:

### Game playing

You can buy machines that can play master level chess for a few hundred dollars. There is some AI in them, but they play well against people mainly through brute force computation--looking at hundreds of thousands of positions. To beat a world champion by brute force and known reliable heuristics requires being able to look at 200 million positions per second.

### Speech recognition

In the 1990s, computer speech recognition reached a practical level for limited purposes. Thus United Airlines has replaced its keyboard tree for flight information by a system using speech recognition of flight numbers and city names. It is quite convenient. On the other hand, while it is possible to instruct some computers using speech, most users have gone back to the keyboard and the mouse as still more convenient.

### Understanding natural language

Just getting a sequence of words into a computer is not enough. Parsing sentences is not enough either. The computer has to be provided with an understanding of the domain the text is about, and this is presently possible only for very limited domains.

### Computer vision

The world is composed of three-dimensional objects, but the inputs to the human eye and computers' TV cameras are two dimensional. Some useful programs can work solely in two dimensions, but full computer vision requires partial three-dimensional information that is not just a set of two-dimensional views. At present there are only limited ways of representing three-dimensional information directly, and they are not as good as what humans evidently use.

**Expert systems**

A ``knowledge engineer'' interviews experts in a certain domain and tries to embody their knowledge in a computer program for carrying out some task. How well this works depends on whether the intellectual mechanisms required for the task are within the present state of AI. When this turned out not to be so, there were many disappointing results. One of the first expert systems was MYCIN in 1974, which diagnosed bacterial infections of the blood and suggested treatments. It did better than medical students or practicing doctors, provided its limitations were observed. Namely, its ontology included bacteria, symptoms, and treatments and did not include patients, doctors, hospitals, death, recovery, and events occurring in time. Its interactions depended on a single patient being considered.

Since the experts consulted by the knowledge engineers knew about patients, doctors, death, recovery, etc., it is clear that the knowledge engineers forced what the experts told them into a predetermined framework. In the present state of AI, this has to be true. The usefulness of current expert systems depends on their users having common sense.

**Heuristic classification**

One of the most feasible kinds of expert system given the present knowledge of AI is to put some information in one of a fixed set of categories using several sources of information. An example is advising whether to accept a proposed credit card purchase. Information is available about the owner of the credit card, his record of payment and also about the item he is buying and about the establishment from which he is buying it (e.g., about whether there have been previous credit card frauds at this establishment).Heuristic classifications can also be applied in Medical fields like pathology, biometrics, pharmacology, etc.

## 12. Support Vector Machine (SVM)

This review of Support Vector Machines attempts to detail the background of Support Vector Machines, their strengths in certain tasks and explains their usefulness to Semantic Classification and Learning tasks. Support Vector Machine (SVM) is a discriminative classifier that learns the decision surface through a process of discrimination and with good generalization characteristics. The approach is a systematic, reproducible and properly motivated by statistical learning theory. Training involves optimization of a convex cost function: there are no false local minima to complicate the learning process. SVMs are the most well-known of a class of algorithms that use the idea of kernel substitution and which are broadly referred to as Kernel Methods. Support Vector Machines (SVM) are a hot topic in current research, being used in a variety of research not just to solve a multitude of different learning and classification problems but also due to its high classification performance (Chapman, 2004; Scholkopf, 1997). For instance the following examples;

- Protein Structure Prediction,
- Land Cover Classification,
- Network Intrusion Detection (Yao et all, 1996),
- Handwriting Recognition,
- Electricity Fraud Prediction (Ahmad, 2007),
- Detecting Steganography in digital images,
- Breast Cancer Diagnosis and Prognosis,
- Particle and Quark-Flavour Identification in High Energy Physics,
- 3D Computer Vision Object Detection,
- Combustion Engine Knock Detection,
- Protein Sequence Transitions,
- Detecting Protein Homologies,
- Text Categorization (Joachims, 1998),
- Predicting time series data,
- Micro array Gene Expression Classification,
- Database Marketing,
- Image Retrieval.

**Advantages of SVM**

The main success of the SVM is its good generalization ability i.e. it can easily distribute data in its feature space and a missing data usually does not affect its solution and output. It also has high classification performance which can be applied in solving various different learning and classification problems. Through statistical learning, it is proven that the bounds on the generalization error or future points not on the training set can be obtained. These bounds are a function of the misclassification error on the training data and terms that measure the complexity or the capacity of the classification function. The size of the margin is not directly dependent on the dimensionality of the data and as such the reason for good performance even for a very high dimensional data (i.e., with a very large number of attributes).

Another major advantage of the SVM approach is its flexibility. Using a basic concept of maximizing margins, duality and kernels, the paradigm can be adapted to many types of inference problems. These maximizations could be changing the norm used for regularization, i.e., how the margin is measured, one can produce a linear program (LP) model for classification; adapting the technique by substituting the kernel functions and forming a dual Lagrangian to do unsupervised learning task of novelty detection and finally, two slack variables are introduced where one computes for underestimating function and the other, overestimating function. For points inside the tube, the slack variables are zero and progressively increase for points outside the tube according to the loss function used. The same strategy of computing the Lagrangian dual and adding kernel functions is then used to construct nonlinear regression functions for regression tasks.

Other appealing features of SVM include the following:

1. SVM are a rare example of a methodology where geometric intuition, elegant mathematics, theoretical guarantees and practical algorithms meet.

2. They present a general methodology for many types of problems where they are applied to many types or wide range of classification, regression and novelty detection tasks. However, they can also be applied to other areas such as operator inversion and unsupervised learning.

3. The method eliminates many of the problems with other inference methodologies like neural networks and decision trees.

a. There are no problems with local minima. One can construct highly nonlinear classification and regression without worrying about getting stuck at local minima.

b. There are few model parameters to choose for example, if one chooses to produce a radial basic function (RBF) machine for classification, one need only two parameters: the penalty parameter for miscalculation and the number of the gaussian function. The number of the basic function is automatically selected by the SVM.

c. The final results are stable, reproducible and largely independent of the specific algorithm used to optimise the SVM model. If two users apply the same SVM model with the same parameters to the same data, they will get the same solution modulo numeric issues. Compared with ANN, the results are dependent on the particular algorithm and starting point used.

4. Robust optimization algorithms exist for solving SVM models. The problems are formulated as mathematical programming models so a state-of-the-art research from the area can be readily applied. Results have been reported in the literature for classification problems with millions of data points.

5. The method is relatively simple to use. One needs not to be a SVM expert to successfully apply existing SVM software to new problems.

6. There are many successful applications of SVM. They have proven to be robust to noise and perform well on many tasks.

**Kernels Used in SVM.**

In this section we consider a situation where the two classes cannot be reasonably separated with a linear discriminant function and nonlinear discriminant function must be used. Figure 9 illustrates two linearly non-separable situations. In a) it is clear that a classifier with a linear discriminant function performs poorly while in b) the classes overlap and the optimal discriminant function is at least roughly linear. In practice, most real-world classification problems are at least linearly non-separable but, in addition to this, the optimal discriminant function is often nonlinear. Note that using a nonlinear discriminant function of course does not guarantee zero training error

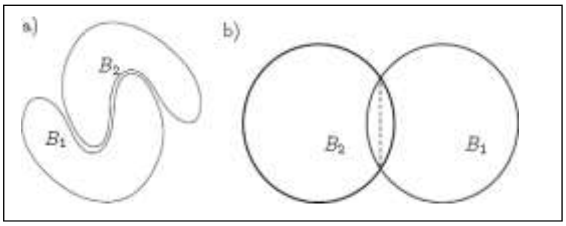


**Figure 7:** In a) the optimal discriminant function is nonlinear while the optimal classifiers have no errors. In b) the optimal discriminant function is linear while the classes overlap and thus optimal classifier is not error free.

We will map the vectors $X_i$, i = 1,…, n, into a new space in the hope that the optimal separating hyperplane in new space performs better classifications than the optimal hyperplane in the original space. More specifically the mapping that will be considered is of the form

$$\Phi(X) = \left(\sqrt{\lambda 1 \psi 1(X)}, \sqrt{\lambda 2 \psi 2}, \ldots\right),$$

Where $\lambda i$ and $\psi i$ are the eigenvalues and the normalized eigen functions of an integral operator $\mathcal{L}k: f \mapsto \int K(v) f(v) dv$. In the SVM literature the space $\mathcal{H}$ is often called a feature space and the $\Phi(Xi)$'s are called feature vectors. Calculating the feature vectors can be computationally expensive, or even impossible, if the dimension of feature space is high or infinite. It should be noted that in the SVM algorithm all the calculations involving the $\Phi(Xi)$'s appear as inner products. Instead of explicitly mapping the vectors into a high dimensional feature space and computing the inner product there it is under certain condition possible to use a function K (**u**, **v**) whose value directly gives the inner product between two vectors $\Phi(u)$ and $\Phi(v)$. Direct consequence is that using K the inner products can be computed roughly the same time in the feature space and in the original space. In the literature the function K ( , ) is usually called a kernel.

**Some well-known kernels**

There are also some difficulties associated with the mapping $\Phi$ and the kernel K. Usually it is very difficult or even impossible to find a mapping that corresponds to a particular kernel and, vice versa, it is difficult to find a kernel that corresponds to some particular mapping. The selection of a kernel function is an important problem in applications although there is no theory to tell which kernel to use. Moreover, it can be difficult to check that some particular kernel satisfies Mercer's conditions, since these conditions must hold for every $f \in$ $L_2$(C). In the following some well-known and widely used kernels are presented. Selection of the kernel, perhaps from among the presented kernels, is usually based on experience and knowledge about the classification problem at hand, and also on theoretical considerations. The problem of choosing the kernel on the basis of theoretical considerations is discussed in the next subsection.

### 1. Polynomial kernel

The polynomial kernel of degree q is of the form

$$K\,(u,v) = (u^T v + c)^q$$

Where c is some non-negative constant, usually c = 1. Using of a generalized inner-product instead of the standard inner-product. In this case, the kernel is

$$K\,(u,v) = \left(\Sigma_i \frac{u_i v_i}{\sigma_i} + c\right)^q$$

Where the vector $\sigma$ is such that the function satisfies the Mercer's condition. When c is positive the kernel is called inhomogeneous and, correspondingly, homogeneous when c = 0. The inhomogeneous kernel avoids problems with the Hessians becoming zero in numerical calculations.

### 2. Sigmoid-function

The sigmoid kernel is of the form

$$K\,(u,v) = \tanh(v\,(u^T v) + a)$$

And it satisfies the Mercer condition only for certain values of the parameters $v$ and $a$. Currently there are no theoretical results on the parameter values that satisfy the Mercer condition and proper values are found by empirical means. When the sigmoid kernel is used with the SVM one can regard it as a two-layer neural network. In two-layer neural network the vector **X** is mapped by the first layer into the vector **F** = (F1,…., FN), where $F_n = \tanh(v\,(z_n^T x) + a)$, $n' =$ 1,…,N, and the dimension of this vector is called the number of hidden units. In the second layer the sign of weighted sum of elements of F is calculated by using weights $\Upsilon n'$.

The major difference between the SVM and a two-layer neural network is in different optimization criterion: in the SVM the goal is to find the optimal hyperplane that maximizes the margin (in the feature space), while in a two-layer neural network the optimization criterion usually is to minimize the empirical risk associated with some loss function, typically the mean squared error. It should be pointed out that quite often in neural networks the optimal network architecture is unknown. If one uses the sigmoid-function with SVM then such problems are avoided, since the number of hidden units (the number of support vectors), the centers in the hidden layer (weights , that is, support vectors) and the vector of weights in the output layer are all determined automatically in the linearly separable case.

### 3. Radial basis function

The Gaussian kernel, known also as the radial basis function, is of the form

$$K\,(u,v) = exp\left(-\frac{\|u-v\|^2}{\sigma^2}\right)$$

Where σ stands for $a$ window width. It is also possible to have different window widths for different vectors, that is, to use a vector σ.

Using the Gaussian kernel with the SVM, the number of basis functions (the number of support vectors), the centers (the $x_i$'s corresponding to the nonzero Lagrangian multipliers, i.e., support vectors) and the weights in the output layer $(\psi a_i)$ of the RBF-network are all determined automatically. Furthermore, in some situations it can be useful to use the centers given by the SVM in an RBF-network if no other information is available for an optimal placing of the centers. It should be pointed out again that the RBF-SVM and the RBF-network use the different optimization criteria.

**Selecting the kernel and the parameters**

When a kernel is used it is often unclear what the properties of the mapping and the feature space are. It is always possible to make a mapping into a potentially very high dimensional space and to produce a classifier with no classification errors on the training set. However, then the performance of the classifier can be poor. On the other hand, it is possible that a classifier with an infinite dimensional feature space performs well. Thus, the dimension of the feature space is not the essential quantity when choosing the right kernel.

This is opposite to the usual curse of dimensionality problem. One could try to select the kernel on the basis of some functional analytic criteria, say using covering numbers that were shown to be important in the upper bounds of the error

On the other hand, Vapnik argued that on the basis of experiments the choice between the kernels presented in the previous subsection does not make a big difference in empirical performance. The more important, and usually the more difficult problem is the selection of the parameters of kernel function. This problem could be solved using a (leave-one-out) cross-validation procedure but quite often with real-world sized training sets this is computationally very costly or even impossible since the quadratic optimization problem of the SVM algorithm is computationally rather demanding.

One approach would be to use the linear approximation in the cross-validation to make the parameter selection faster. Recently, some more advanced approaches have been proposed. Various kernel dependent upper bounds are given on the leave-one-out error of the SVM. These upper bounds are then differentiated with respect to kernel parameters and then, by using some optimization algorithm (for example Newton- Rhapson -method), the best values for a kernel are found.

## 13. Conclusion

Support Vector Machines (SVM) is a method of calculating the optimal separating hyperplane in the feature space. Optimal separating hyperplane is defined as the maximum-margin hyperplane in the higher dimensional feature space.

The use of the maximum-margin hyperplane is motivated by statistical learning theory, which provides a probabilistic test error bound which is minimized when the margin is maximized. The parameters of the maximum-margin hyperplane are derived by solving a quadratic programming (QP) optimization problem. There exist several specialized algorithms for quickly solving the QP problem that arises from SVMs.

The original SVM was a linear classifier. However, Vapnik suggested using the kernel trick (originally proposed by Aizerman et al., 1964). In the kernel trick, each dot product used in a linear algorithm is replaced with a non-linear kernel function. This causes the linear algorithm to operate in a different space. For SVMs, using the kernel trick makes the maximum margin hyperplane be fit in a feature space. The feature space is a non-linear map from the original input space, usually of much higher dimensionality than the original input space. In this way, non-linear SVMs can be created. If the kernel used is a radial basis function, the corresponding feature space is a Hilbert space of infinite dimension.

Least Squares SVM (LS-SVM) simplifies the formulation by replacing the inequality constraint in SVM with an equality constraint. This approach significantly reduces the complexity and the computation times, solving a set of linear equations instead of solving the QP problem.

## Maturity of Artificial Intelligence In Medicine

The earliest work in medical artificial intelligence (AI) dates to the early 1970s, when the field of AI was about 15 years old (the phrase ‘‘artificial intelligence’’ had been first coined at a famous Dartmouth College conference in 1956 (Hoopgood, 2002). Early AI in medicine (AIM) researchers had discovered the applicability of AI methods to life sciences, most visibly in the Dendral experiments (Lindsay et Al, 1980) of the late 1960s and early 1970s, which brought together computer scientists (e.g., Edward Feigenbaum), chemists (e.g., Carl Djerassi), geneticists (e.g., Joshua Lederberg), and philosophers of science (e.g., Bruce Buchanan) in collaborative work that demonstrated the ability to represent and utilize expert knowledge in symbolic form.

There was an explosive interest in biomedical applications of AI during the 1970s, catalyzed in part by the creation of the SUMEX-AIM Computing Resource (Freiherr, 1980) at Stanford University, and a sister facility at Rutgers University, which took advantage of the nascent ARPANET to make computing cycles available to a national (and eventually international) community of researchers applying AI methods to problems in biology and medicine. Several early AIM systems including Internist-1 (Miller et. Al., 1982), CASNET (Weiss et. Al, 1978), and MYCIN (Shortliffe, 1976), were developed using these shared national resources, supported by the Division of Research Resources at the National Institutes of Health.

The general AI research community was fascinated by the applications being developed in the medical world, noting that significant new AI methods were emerging as AIM researchers struggled with challenging biomedical problems. In fact, by 1978, the leading journal in the field (Artificial Intelligence, Elsevier, Amsterdam) had devoted a special issue (Sridharan, 1978) solely to AIM research papers. Over the next decade, the community continued to grow, and with the formation of the American Association for Artificial Intelligence in 1980, a special subgroup on medical applications (AAAI-M) was created. It was against this background that Ted Shortliffe was asked to address the June 1991 conference of the organization that had become known as Artificial Intelligence in Medicine Europe (AIME), held in Maastricht, The Netherlands. By that time the field was in the midst of ‘‘AI winter’’ (Wikipedia, 2008), although the introduction of personal computers and high-performance workstations was enabling new types of AIM research and new models for technology dissemination. In that talk, he attempted to look back on the progress of AI in medicine to date, and to anticipate the major challenges for the decade ahead. A paper based on that talk was later published in Artificial Intelligence in Medicine (Shortliffe, 1993).

## Medical Data Mining

Human medical data are at once the most rewarding and difficult of all biological data to mine and analyze. Humans are the most closely watched species on earth. Human subjects can provide observations that cannot easily be gained from animal studies, such as visual and auditory sensations, the perception of pain, discomfort, hallucinations, and recollection of possibly relevant prior traumas and exposures. Most animal studies are short-term, and therefore cannot track long-term disease processes of medical interest, such as preneoplasia or atherosclerosis. With human data, there is no issue of having to extrapolate animal observations to the human species.

Some three-quarter billions of persons living in North America, Europe, and Asia have at least some of their medical information collected in electronic form, at least transiently. These subjects generate volumes of data that an animal experimentalist can only dream of. On the other hand, there are ethical, legal, and social constraints on data collection and distribution, that do not apply to non-human species, and that limit the scientific conclusions that may be drawn. The major points of uniqueness of medical data may be organized under four general headings:

- Heterogeneity of medical data
  - Volume and complexity of medical data
  - Physician’s interpretation
  - Sensitivity and specificity analysis
  - Poor mathematical characterization
  - Canonical form
- Ethical, legal, and social issues
  - Data ownership
  - Fear of lawsuits
  - Privacy and security of human data
  - Expected benefits
  - Administrative issues
- Statistical philosophy
  - Ambush in statistics
  - Data mining as a superset of statistics
  - Data mining and knowledge discovery process
- Special status of medicine: Finally, medicine has a special status in science, philosophy, and daily life. The outcomes of

medical care are life-or-death, and they apply to everybody. Medicine is a necessity, not merely an optional luxury, pleasure, or convenience.

In summary, data mining in medicine is distinct from that in other fields, because the data are heterogeneous; special ethical, legal, and social constraints apply to private medical information; statistical methods must address these heterogeneity and social issues; and because medicine itself has a special status in life.

Data from medical sources are voluminous, but they come from many different sources, not all commensurate structure or quality. The physician's interpretations are an essential component of these data. The accompanying mathematical models are poorly characterized compared to the physical sciences. Medicine is far, far from the intellectual gold standard of a canonical form for its basic concepts.

The ethical, legal, and social limitations on medical data mining relate to privacy and security considerations, fear of lawsuits, and the need to balance the expected benefits of research against any inconvenience or possible injury to the patient. Methods of medical data mining must address the heterogeneity of data sources, data structures, and the pervasiveness of missing values for both technical and social reasons.

The natural history of disease affects statistical hypotheses in an unknown way. Statistical hypothesis tests often take the form of an ambush or a contest with a winner and a loser. The relevance of this model to the natural processes of medicine is questionable. For all its perils, medical data mining can also be the most rewarding. For an appropriately formulated scientific question, thousands of data-elements can be brought to bear on finding a solution. For an appropriately formulated medical question, finding an answer could mean extending a life, or giving comfort to an ill person. These potential rewards more than compensate for the many extraordinary difficulties along the pathway to success. For more info, see Cios and Moore (2002)

## References

[1] Wa'el M. Mahmud, Hamdy N.Agiza, and Elsayed Radwan, Intrusion Detection Using Rough Sets based Parallel Genetic Algorithm Hybrid Model, Proceedings of the World Congress on Engineering and Computer Science 2009 Vol II WCECS 2009, October 20-22, 2009, San Francisco, USA

[2] Mariusz Nowostawski and Riccardo Poli, "Parallel genetic algorithm taxonomy" Knowledge-Based Intelligent Information Engineering Systems, 1999, Third International Conference Volume, Issue, Dec 1999 Page(s) 88 – 92.

[3] Mohammad M. Rahman1, Dominik Slezak, and Jakub Wroblewski, "Parallel island model for attribute reduction" Lecture Notes in Computer Science. 2005.

[4] S. Mukkamala and A. H. Sung, A comparative study of techniques for intrusion detection, In Proceedings of 15th IEEE International Conference on Tools with Artificial Intelligence, pages 570– 577. IEEE Press, 3-5 Nov. 2003.

[5] S. Cayzer and J. Smith, Gene libraries: Coverage, efficiency and diversity. In H. Bersini and J. Carneiro, editors, Artificial Immune Systems, volume 4163 of Lecture Notes in Computer Science, pages 136–149, Springer Berlin/Heidelberg, 2006.

[6] J. Jiang, C. Zhang, and M. Kame, RBF-based real-time hierarchical intrusion detection systems, In Proceedings of the International Joint Conference on Neural Networks (IJCNN '03), volume 2, pages 1512–1516, Portland, OR, USA, 20-24 July 2003. IEEE Press

[7] C. Zhang, J. Jiang, and M. Kamel. Comparison of BPL and RBF network in intrusion detection system, In G. Wang, Q. Liu, Y. Yao, and A. Skowron, editors, Proceedings of the 9th International Conference on Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing (RSFDGrC '03), 26-29 May, Chongqing, China, volume [8]2639 of Lecture Notes in Computer Science, chapter Proceedings of the 9th International Conference on Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing (RSFDGrC '03), pages 466–470. Springer Berlin / Heidelberg, 2003

[8] Z. Liu, G. Florez, and S. M. Bridges. A comparison of input representations in neural networks: A case study in intrusion detection. In Proceedings of the International Joint Conference on Neural Networks (IJCNN '02), volume 2, pages 1708–1713, Honolulu, HI, USA, 12-17 May 2002. IEEE Press.

[9] Y. Liao, V. R. Vemuri, and A. Pasos, Adaptive anomaly detection with evolving connectionist systems. Journal of Network and Computer Applications, 30(1):60–80, 2007. Special Issue on Network and Information Security: A Computational Intelligence Approach.

[10] N. A. Durgin and P. Zhang, Profile-based adaptive anomaly detection for network security, Technical report, Sandia National Laboratories, 2005.

[11] A. Abraham, C. Grosan, and C. Martin-Vide, Evolutionary design of intrusion detection programs. International Journal of Network Security, 4(3):328–339, 2007. (a)

[12] A. Abraham, R. Jain, J. Thomas, and S. Y. Han. D-SCIDS: Distributed soft computing intrusion detection system. Journal of Network and Computer Applications, 30(1):81–98, 2007. (b)

[13] Leonardo Noriega, Multilayer Perceptron Tutorial, School of Computing Staffordshire University November 17, 2005

[14] Quinlan J.R.; Comparing Connectionist and Symbolic Learning Methods, Basser Department of Computer Science, University of Sydney

[15] Shravya R. K.; A Comparative Evaluation of Symbolic Learning Methods and Neural Learning Methods, Department of Computer Science, University of Maryland, College Park

[16] Hopgood A. A (2002), Intelligent Systems for Engineers and Scientists, CRC Press, pp 158-175, 206-233.

[17] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009), The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1.

[18] Jeff Schneider, Cross Validation, Feb 7, 1997, http://www.cs.cmu.edu/~schneide/tut5/node42.html

[19] Martin D. S., Data Mining - Classification II, Tutorial 07, Friday 24th April 2009.

[20] Quinlan R. J. (1992): Learning with Continuous Classes. In: 5th Australian Joint Conference on Artificial

Intelligence, Singapore, 343-348.

[21] Quinlan R. J. (1993): C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers, San Mateo, CA.

[22] Hopgood, Adrian A., the State of Artificial Intelligence, School of Computing & Informatics Nottingham Trent University, Burton Street, Nottingham, NG1 4BU, UK. 2005.

[23] Penrose R., The Emperor's New Mind, Oxford University Press, London, 1989

[24] Turing A.M., "Computing machinery and intelligence", Mind 59 (1950) 433–460

[25] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth. A.H. Teller and E. Teller, J. Chem. Phys. 21 (1953) 1087-1092.

[26] Kirkpatrick, S., C. D. Gelatt Jr., M. P. Vecchi, "Optimization by Simulated annealing", Science, 220, 4598, 671-680, 1983.

[27] Krzysztof J. Cios and G. William Moore, Uniqueness of medical data mining, Artificial Intelligence in Medicine 26 (2002) 1–24

[28] Lindsay R. K., Buchanan B. G., Feigenbaum E. A. and Lederberg J., Applications of artificial intelligence for organic chemistry: the DENDRAL Project. New York:
McGraw-Hill; 1980

[29] History of Artificial Intelligence; http://en.wikipedia.org/wiki/History_of_artificial_intelligence (Accessed June 1, 2008)

[30] Freiherr G. The seeds of artificial intelligence: SUMEX-AIM (1980). U.S. G.P.O; DHEW publication no.(NIH) 80-2071, Washington, D.C.; U.S. Dept. of Health, Education, and Welfare, Public Health Service, National Institutes of Health; 1980.

[31] Miller R. A., Pople H. E. and Myers J. D. Internist-1: an experimental computer-based diagnostic consultant for general internal medicine. New England Journal of Medicine 1982; 307(8): 468-76.

[32] Weiss S. M., Kulikowski C. A., Amarel S. and Safir A., A model-based method for computer-aided medical decision making, Artificial Intelligence 1978; 11: 145-72.

[33] Shortliffe E. H., Computer-based medical consultations: MYCIN. New York: Elsevier; 1976.

[34] Sridharan N. S., Guest editorial. Artificial Intelligence 1978:11 (1-2); 1-4.

[35] Shortliffe E. H., The adolescence of AI in medicine: will the field come of age in the 90s? Artificial Intelligence in Medicine 1993; 5: 93-106

[36] Fulton S.L. and Pepe C.O., "An introduction to model-based reasoning", AI Expert (January 1990) 48–55

[37] Fenton W.G., Mcginnity T.M. and Maguire L.P., "Fault diagnosis of electronic systems using intelligent techniques: a review", IEEE Transactions on Systems Man and Cybernetics Part C—Applications and Reviews 31 (2001) 269–281.

[38] Wotawa F., "Debugging VHDL designs using model-based reasoning", Artificial Intelligence in Engineering 14 (2000) 331–351.

[39] Mateis C., Stumptner M. and Wotawa F., "Locating bugs in Java programs—first results of the Java diagnosis experiments project", in: Lecture Notes in Artificial Intelligence, vol. 1821, Springer-Verlag, Berlin/New York, 2000, pp. 174–183.

[40] Montani S., Magni P., Bellazzi R., Larizza C., Roudsari A.V. and Carson E.R., "Integrating model-based decision support in a multi-modal reasoning system for managing type 1 diabetic patients", Artificial Intelligence in Medicine 29 (2003) 131–151.

[41] Bruninghaus S. and Ashley K.D., Combining case-based and model-based reasoning for predicting the outcome of legal cases, in: Lecture Notes in Artificial Intelligence, vol. 2689, Springer-Verlag, Berlin/New York, 2003, pp. 65–79.

[42] Koning K., Bredeweg B., Breuker J. and Wielinga B., "Model-based reasoning about learner behaviour", Artificial Intelligence 117 (2000) 173–229

[43] Xing H., Huang S.H. and Shi J., "Rapid development of knowledge-based systems via integrated knowledge acquisition", Artificial Intelligence for Engineering Design Analysis and Manufacturing 17 (2003) 221–234

[44] Minton S., Carbonell J.G., Knoblock C.A., Kuokka D.R., Etzioni O. and Gil Y., "Explanation-based learning: a problem-solving perspective", Artificial Intelligence 40 (1989) 63–118.

[45] Aamodt A. and Plaza E., "Case-based reasoning—foundational issues, methodological variations, and system approaches", AI Communications 7 (1994) 39–59.

[46] Wooldridge M.J., "Agent-based software engineering", IEE Proc. Software Engineering 144 (1997) 26–37.

[47] M.A. Aizerman, E.M. Braverman and L.I. Rozonoer, "Theoretical Foundations of the Potential Function Method in Pattern Recognition Learning"; A

[48] Shelly Xiaonan Wu and Wolfgang Banzhaf, The Use of Computational intelligence in Intrusion Detection Systems: A Review, Technical Report #2008-05. November 2008.

[49] Riesbeck C.K. and Schank R.C., Inside Case-based Reasoning, Lawrence Erlbaum Associates, Hillsdale, NJ, 1989