Dr. SNS RAJALAKSHMI COLLEGE OF ARTS & SCIENCE

Coimbatore – 49.

DEPARTMENT OF COMPUTER APPLICATIONS(PG)

LECTURE NOTES

CLASS : I MCA BATCH : 2023 - 2025

COURSE : INTRODUCTION TO ARTIFICIAL INTELLIGENCE

COURSE CODE : 21PCA101 SEMESTER : I

UNIT – II

- **MACHINE LEARNING SYMBOL BASED**:

Introduction

- The ability to learn must be part of any system that would claim to possess general intelligence. Indeed, in our world of symbols and interpretation, the very notion of an unchanging intellect seems a contradiction in terms.
- Intelligent agents must be able to change through the course of their interactions with the world, as well as through the experience of their own internal states and processes. We present four chapters on machine learning, reflecting four approaches to the problem: first, the symbol-based, second, the connectionist, third, the genetic/evolutionary, and finally, the dynamic/stochastic.

- Learning is important for practical applications of artificial intelligence. Feigenbaum and McCordick (1983) have called the "knowledge engineering bottleneck" the major obstacle to the widespread use of intelligent systems.

- This "bottleneck" is the cost and difficulty of building systems using the traditional knowledge acquisition techniques One solution to this problem would be for programs to begin with a minimal 10 388 PART IV / MACHINE LEARNING amount of knowledge and learn from examples, high-level advice, or their own explorations of the application domain.

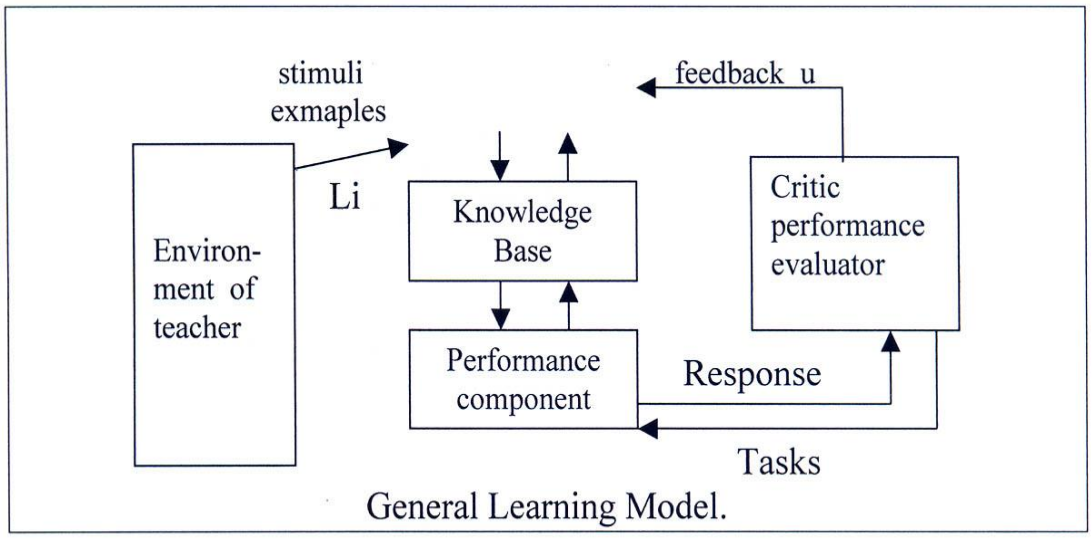
- Learning involves changes in the learner; this is clear. However, the exact nature of those changes and the best way to represent them are far from obvious. One approach model learning as the acquisition of explicitly represented domain knowledge. Based on its experience, the learner constructs or modifies expressions in a formal language, such as logic, and retains this knowledge for future use. Symbolic approaches, characterized by the algorithms of Chapter 10, are built on the assumption that the primary influence on the program's behavior is its collection of explicitly represented domain knowledge.

- we consider genetic and evolutionary learning. This model of learning we have may be seen in the human and animal systems that have evolved towards equilibration with the world. This approach to learning through adaptation is reflected in genetic algorithms, genetic programming, and artificial life research**.**

- In contrast to similarity-based methods, a learner may use prior knowledge of the domain to guide generalization. For example, humans do not require large numbers of examples to learn effectively. Often, a single example, analogy, or high-level bit of advice is sufficient to communicate a general concept.
- The effective use of such knowledge can help an agent to learn more efficiently, and with less likelihood of error. Section 10.5 examines explanation-based learning, learning by analogy and other techniques using prior knowledge to learn from limited training data.

➢ FRAMEWORK FOR SYMBOL BASED LEARNING

1. The data and goals of the learning task. One of the primary ways in which we characterize learning problems is according to the goals of the learner and the data it is given.
2. The concept learning algorithms begin with a collection of positive (and usually negative) examples of a target class; the goal is to infer a general definition that will allow the learner to recognize future instances of the class. In contrast to the data-intensive approach taken by these algorithms, explanation-based learning attempts to infer a general concept from a single training example and a prior base of domain-specific knowledge.
3. The conceptual clustering algorithms discussed in illustrate another variation on the induction problem: instead of a set of training instances of known categorization, these algorithms begin with a set of unclassified instances. Their task is to discover categorizations that may have some utility to the learner.



General Learning Model.

✓ **The representation of learned knowledge.**

Machine learning programs have made use of all the representation languages discussed in this text. For example, programs that learn to classify objects may represent these concepts as expressions in predicate calculus or they may use a structured representation such as frames or objects. Plans may be described as a sequence of operations or a triangle table. Heuristics may be represented as problem-solving rules.

A simple formulation of the concept learning problem represents instances of a concept as conjunctive sentences containing variables. For example, two instances of "ball" (not sufficient to learn the concept) may be represented by

size (obj1, small) ∧ color (obj1, red) ∧ shape (obj1, round)
size (obj2, large) ∧ color (obj2, red) ∧ shape (obj2, round)

The general concept of "ball" could be defined by:
size (X, Y) ∧ color (X, Z) ∧ shape (X, round)

where any sentence that unifies with this general definition represents a ball.

A set of operations. Given a set of training instances, the learner must construct a generalization, heuristic rule, or plan that satisfies its goals. This requires the ability to manipulate representations. Typical operations include generalizing orspecializing symbolic expressions, adjusting the weights in a neural network, or otherwise modifying the program's representations
In the concept learning example just introduced, a learner may generalize a definition by replacing constants with variables. If we begin with the concept:

size (obj1, small) ∧ color (obj1, red) ∧ shape (obj1, round)
replacing a single constant with a variable produces the generalizations:

size (obj1, X) ∧ color (obj1, red) ∧ shape (obj1, round)
size (obj1, small) ∧ color (obj1, X) ∧ shape (obj1, round)
size (obj1, small) ∧ color (obj1, red) ∧ shape (obj1, X)
size (X, small) ∧ color (X, red) ∧ shape (X, round)

- **The concept spaces**.: The representation language, together with the operations described above, defines a space of potential concept definitions. The learner must search this space to find the desired concept. The complexity of this concept space is a primary measure of the difficulty of a learning problem.

- Heuristic search: Learning programs must commit to a direction and order of search, as well as to the use of available training data and heuristics to search efficiently. In our example of learning the concept "ball," a plausible algorithm may take the first example as a candidate concept and generalize it to include subsequent examples. For instance, on being given the single training example

size (obj1, small) ∧ color (obj1, red) ∧ shape (obj1, round)

- The learner will make that example a candidate concept; this concept correctly classifies the only positive instance seen.

If the algorithm is given a second positive instance

size (obj2, large) ∧ color (obj2, red) ∧ shape (obj2, round)

the learner may generalize the candidate concept by replacing constants with variables as needed to form a concept that matches both instances. The result is a more general candidate concept that is closer to our target concept of "ball."

Patrick Winston's work (1975a) on learning concepts from positive and negative examples illustrates these components. His program learns general definitions of structural concepts, such as "arch," in a blocks world. The training data is a series of positive and negative examples of the concept: examples of blocks world structures that fit in the category, along with near misses. The latter are instances that almost belong to the category but fail on one property or relation. The near misses enable the program to single out features that can be used to exclude negative instances from the target concept. Figure 10.2 shows positive examples and near misses for the concept "arch."

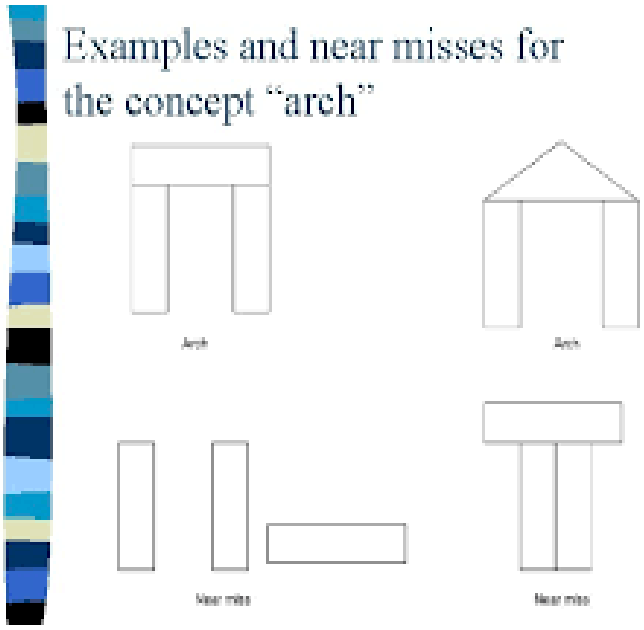


Figure 10.2 Examples and near misses for the concept arch.

- ✓ **Version space search**

Version space search (Mitchell 1978, 1979, 1982) illustrates the implementation of inductive learning as search through a concept space. Version space search takes advantage of the fact that generalization operations impose an ordering on the concepts in a space, and then uses this ordering to guide the search.

Generalization and specialization are the most common types of operations for defining a concept space. The primary generalization operations used in machine learning are:

1. Replacing constants with variables. For example,

   color (ball, red)

generalizes to
color (X, red)

2. Dropping conditions from a conjunctive expression.
   shape (X, round) ∧ size (X, small) ∧ color (X, red)
   generalizes to
   shape (X, round) ∧ color (X, red)

3. Adding a disjunct to an expression.
   shape (X, round) ∧ size (X, small) ∧ color (X, red)
   generalizes to
   shape (X, round) ∧ size (X, small) ∧ (color (X, red) ∨ color (X, blue))

4. Replacing a property with its parent in a class hierarchy. If we know that primary_color is a superclass of red, then
   color (X, red)
   generalizes to
   color (X, primary_color)

   We may think of generalization in set theoretic terms: let P and Q be the sets of sentences matching the predicate calculus expressions p and q, respectively. Expression p is more general than q iff P ⊇ Q. In the above examples, the set of sentences that match color (X, red) contains the set of elements that match color (ball, red). Similarly, in example 2, we may think of the set of rounds, red things as a superset of the set of small, red, round things. Note that the "more general than" relationship defines a partial ordering on the space of logical sentences. We express this using the "≥" symbol, where p ≥ q means that p is more general than q. This ordering is a powerful source of constraints on the search performed by a learning algorithm.

- The ID3 Decision Tree induction Algorithm

  ID3 (Quinlan 1986a), like candidate elimination, induces concepts from examples. It is particularly interesting for its representation of learned knowledge, its approach to the management of complexity, its heuristic for selecting candidate concepts, and its potential for handling noisy data. ID3 represents concepts as decision trees, a representation that allows us to determine the classification of an object by testing its values for certain properties.

  For example,
  consider the problem of estimating an individual's credit risk on the basis of such properties as credit history, current debt, collateral, and income. Figure 10.1 lists a sample of individuals with known credit risks. The decision tree of Figure 10.13 represents the classifications in Table 10.1, in that this tree can correctly classify all the objects in the table. In a decision tree, each internal node represents a test on some property, such as credit history or debt; each possible value of that property corresponds to a branch of the tree. Leaf nodes

represent classifications, such as low or moderate risk. An individual of unknown type may be classified by traversing this tree: at each internal node, test the individual's value for that property and take the appropriate branch. This continues until reaching a leaf node and the object's classification

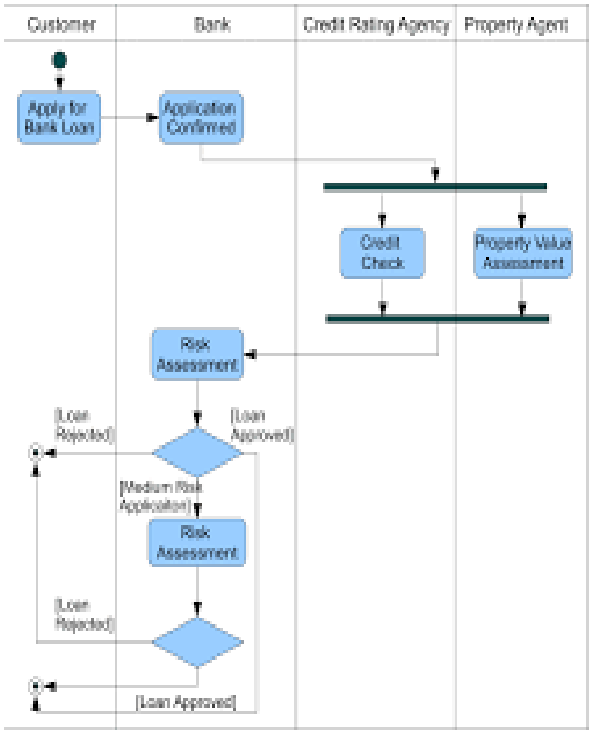
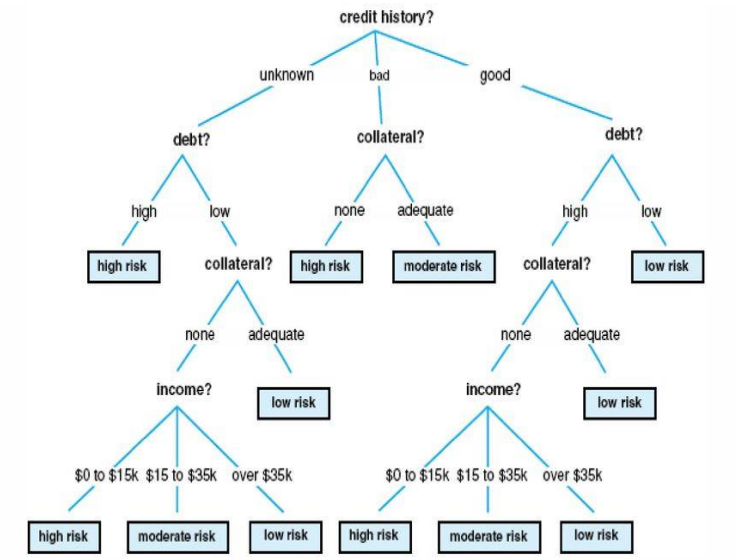


Fig:10.1 Data from credit history of loan applications

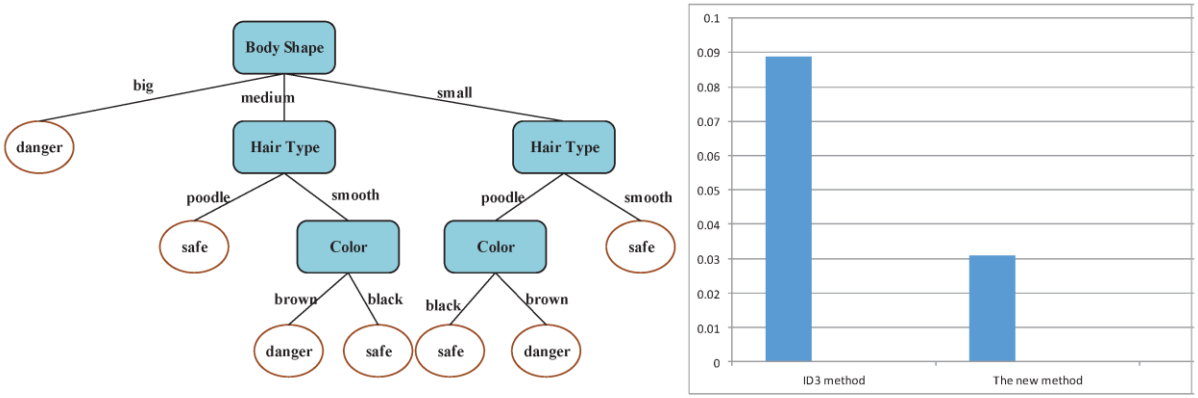
Fig 10.13 A decision tree for credit risk assessment.



In general, the size of the tree necessary to classify a given set of examples varies according to the order with which properties are tested. Figure 10.14 shows a tree that is considerably simpler than that of Figure 10.13 but that also classifies correctly the examples in Figure10.1. Given a set of training instances and a number of different decision trees that correctly classify them, we may ask which tree has the greatest likelihood of correctlclassifying unseen instances of the population. The ID3 algorithm assumes that this is the simplest decision tree that covers all the training examples. The rationale for this assumption is the time-honored heuristic of preferring simplicity and avoiding unnecessary assumptions. This principle, known as Occam's Razor, was first articulated by the medieval logician William of Occam in 1324:

✓ Evaluating ID3

- Although the ID3 algorithm produces simple decision trees, it is not obvious that such trees will be effective in predicting the classification of unknown examples. ID3 has been evaluated in both controlled tests and applications and has proven to work well in practice.
- Has evaluated ID3's performance on the problem of learning to classify boards in a chess endgame (Quinlan 1983). The endgame involved white, playing with a king and a rook, against black, playing with a king and a knight. ID3's goal was to learn to recognize boards that led to a loss for black within three moves. The attributes were different high-level properties of boards, such as "an inability to move the king safely." The test used 23 such attributes.
- These results are supported by further empirical studies and by anecdotal results from further applications. Variations of ID3 have been developed to deal with such problems as noise and excessively large training sets. For more details, see Quinlan (1986a, b).



(a) (b)

✓ The evaluation of ID3

Decision Tree Data Issues: Bagging, Boosting

Quinlan (1983) was the first to suggest the use of information theory to produce subtrees in decision tree learning and his work was the basis for our presentation. Our examples were clean, however, and their use straightforward. There are a number of issues that we did not address, each of which often occurs in a large data set:

1. The data is bad. This can happen when two (or more) identical attribute sets give different results. What can we do if we have no a priori reason to get rid of data?

2. Data from some attribute sets is missing, perhaps because it is too expensive to obtain. Do we extrapolate? Can we create a new value "unknown?" How can we smooth over this irregularity?
3. Some of the attribute sets are continuous. We handled this by breaking the continuous value "income" into convenient subsets of values, and then used these groupings. Are there better approaches?
4. The data set may be too large for the learning algorithm. How do you handle this?

- Bagging produces replicate training sets by sampling with replacement from the training instances. Boosting uses all instances at each replication, but maintains a weight for each instance in the training set.
- This weight is intended to reflect that vector's importance. When the weights are adjusted, different classifiers are produced, since the weights cause the learner to focus on different instances. In either case, the multiple classifiers produced are combined by voting to form a composite classifier. In bagging, each component classifier has the same vote, while boosting assigns different voting strengths to the component classifiers on the basis of their accuracy

➢ KNOWLEDGE AND LEARNING

- ID3 and the candidate elimination algorithm generalize on the basis of regularities in training data. Such algorithms are often referred to as similarity based, in that generalization is primarily a function of similarities across training examples.
- The biases employed by these algorithms are limited to syntactic constraints on the form of learned knowledge; they make no strong assumptions about the semantics of the domains. In this CHAPTER 10 / MACHINE LEARNING: SYMBOL-BASED 423 section, we examine algorithms, such as explanation-based learning, that use prior domain knowledge to guide generalization.

✓ Meta-DENDRAL

- Meta-DENDRAL (Buchanan and Mitchell 1978) is one of the earliest and still one of the best examples of the use of knowledge in inductive learning. Meta-DENDRAL acquires rules to be used by the DENDRAL program for analyzing mass spectrographic data. DENDRAL infers the structure of organic molecules from their chemical formula and mass spectrographic data.
- A mass spectrograph bombards molecules with electrons, causing some of the chemical bonds to break. Chemists measure the weight of the resulting pieces and interpret these results to gain insight into the structure of the compound. DENDRAL employs knowledge in the form of rules for interpreting mass spectrographic data. The premise of a DENDRAL rule is a graph of some portion of a molecular structure. The conclusion of the rule is that graph with the location of the cleavage indicated.
- Meta-DENDRAL infers these rules from spectrographic results on molecules of known structure. Meta-DENDRAL is given the structure of a known compound, along with the mass and relative abundance of the fragments produced by spectrography. It interprets these, constructing an account of where the breaks occurred. These explanations of breaks in specific molecules are used as examples for constructing general rules
- These explanations become the set of positive instances for a rule induction program. This component induces the constraints in the premises of DENDRAL rules through a general to specific search. It begins with a totally

general description of a cleavage: X1∗X2. This pattern means that a cleavage, indicated by the asterisk, can occur between any two atoms. It specializes the pattern by:

adding atoms: X1∗X2 → X3 − X1∗X2

where the "−" operator indicates a chemical bond, or

instantiating atoms or attributes of atoms: X1∗X2 → C∗X2

- Meta-DENDRAL learns from positive examples only and performs a hill-climbing search of the concept space. It prevents overgeneralization by limiting candidate rules to cover only about half of the training instances. Subsequent components of the program evaluate and refine these rules, looking for redundant rules or modifying rules that may be overly general or specific.
- The strength of meta-DENDRAL is in its use of domain knowledge to change raw data into a more usable form. This gives the program noise resistance, through the use of its theory to eliminate extraneous or potentially erroneous data, and the ability to learn from relatively few training instances. The insight that training data must be so interpreted to be fully useful is the basis of explanation-based learning.

✓ Explanation-Based Learning

Explanation-based learning uses an explicitly represented domain theory to construct an explanation of a training example, usually a proof that the example logically follows from the theory. By generalizing from the explanation of the instance, rather than from the instance itself, explanation-based learning filters noise, selects relevant aspects of experience, and organizes training data into a systematic and coherent structure.

1. A target concepts. The learner's task is to determine an effective definition of this concept. Depending upon the specific application, the target concept may be a classification, a theorem to be proven, a plan for achieving a goal, or a heuristic for a problem solver.
2. A training example, an instance of the target.
3. A domain theory, a set of rules and facts that are used to explain how the training example is an instance of the goal concept.
4. Operationality criteria, some means of describing the form that concept definitions may take.

Explanation-based learning offers a number of benefits:

1. Training examples often contain irrelevant information, such as the color of the cup in the preceding example. The domain theory allows the learner to select the relevant aspects of the training instance.

2. A given example may allow numerous possible generalizations, most of which are either useless, meaningless, or wrong. EBL forms generalizations that are known to be relevant to specific goals and that are guaranteed to be logically consistent with the domain theory.

3. By using domain knowledge EBL allows learning from a single training instance.

4. Construction of an explanation allows the learner to hypothesize unstated relationships between its goals and its experience, such as deducing a definition of a cup based on its structural properties.

✓ EBL and Knowledge-Level Learning

- Although it is an elegant formulation of the role of knowledge in learning, EBL raises a number of important questions. One of the more obvious ones concerns the issue of what an explanation-based learner actually learns. Pure EBL can only learn rules that are within the deductive closure of its existing theory.
- This means the learned rules could have been inferred from the knowledge base without using the training instance at all. The sole function of the training instance is to focus the theorem prover on relevant aspects of the problem domain
- A further use for explanation-based learning is to integrate it with similarity-based approaches to learning. Again, a number of basic schemes suggest themselves, such as using EBL to refine training data where the theory applies and then passing this partially generalized data on to a similarity-based learner for further generalization. Alternatively, we could use failed explanations as a means of targeting deficiencies in a theory, thereby guiding data collection for a similarity-based learner.

✓ Analogical Reasoning

- The standard computational model of analogy defines the source of an analogy to be a problem solution, example, or theory that is relatively well understood. The target is not completely understood. Analogy constructs a mapping between corresponding elements of the target and source.
- Analogical inferences extend this mapping to new elements of the target domain. Continuing with the "electricity is like water" analogy, if we know that this analogy maps switches onto valves, amperage onto quantity of flow, and voltage onto water pressure, we may reasonably infer that there should be some analogy to the capacity (i.e., the cross-sectional area) of a water pipe; this could lead to an understanding of electrical resistance.
- A typical framework consists of the following stages:

  1. Retrieval. Given a target problem, it is necessary to select a potential source analog. Problems in analogical retrieval include selecting those features of the target and source that increase the likelihood of retrieving a useful source analog and indexing knowledge according to those features. Generally, retrieval establishes the initial elements of an analogical mapping.

  2. Elaboration. Once the source has been retrieved, it is often necessary to derive additional features and relations of the source. For example, it may be necessary to develop a specific problem-solving trace (or explanation) in the source domain as a basis for analogy with the target.

  3. Mapping and inference. This stage involves developing the mapping of source attributes into the target domain. This involves both known similarities and analogical inferences.

  4. Justification. Here we determine that the mapping is indeed valid. This stage may require modification of the mapping.

  5. Learning. In this stage the acquired knowledge is stored in a form that will be useful in the future.

➢ Conceptual Clustering

  - The clustering problem begins with a collection of unclassified objects and a means for measuring the similarity of objects. The goal is to organize the objects into classes that meet some standard of quality, such as maximizing the similarity of objects in the same class.
  - Numeric taxonomy is one of the oldest approaches to the clustering problem. Numeric methods rely upon the representation of objects as a collection of features, each of which may have some numeric

value. A reasonable similarity metric treats each object (a vector of n feature values) as a point in n-dimensional space. The similarity of two objects is the euclidean distance between them in this space.

- Using this similarity metric, a common clustering algorithm builds clusters in a bottom-up fashion. This approach, often called an agglomerative clustering strategy, forms categories by:

  1. Examining all pairs of objects, selecting the pair with the highest degree of similarity, and making that pair a cluster.

  2. Defining the features of the cluster as some function, such as average, of the features of the component members and then replacing the component objects with this cluster definition.

  3. Repeating this process on the collection of objects until all objects have been reduced to a single cluster.

  4. Many unsupervised learning algorithms can be viewed as performing maximum liklihood density estimations, which means finding a distribution from which the data is most likely to have been drawn. An example is the interpretation of a set of phonemes in a natural language application

  Conceptual clustering addresses these problems by using machine learning techniques to produce general concept definitions and applying background knowledge to the formation of categories. CLUSTER/2 (Michalski and Stepp 1983) is a good example of this approach. It uses background knowledge in the form of biases on the language used to represent categories.

  CLUSTER/2 forms k categories by constructing individuals around k seed objects. k is a parameter that may be adjusted by the user. CLUSTER/2 evaluates the resulting clusters, selecting new seeds and repeating the process until its quality criteria are met.

✓ The algorithm is defined:

  1. Select k seeds from the set of observed objects. This may be done randomly or according to some selection function.

  2. For each seed, using that seed as a positive instance and all other seeds as negative instances, produce a maximally general definition that covers all of the positive and none of the negative instances. Note that this may lead to multiple classifications of other, nonseed, objects.

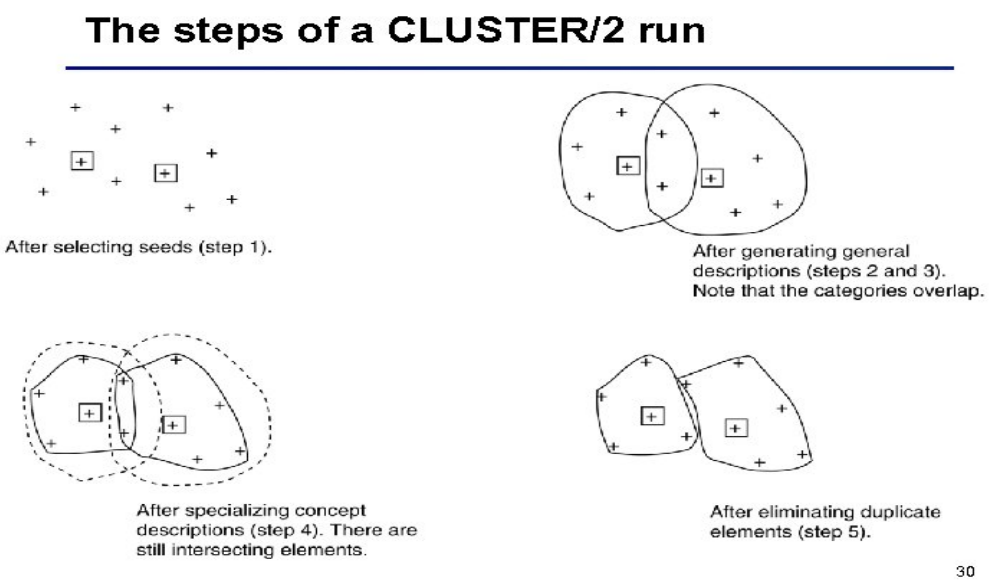
  3. Classify all objects in the sample according to these descriptions. Replace each maximally general description with a maximally specific description that covers all objects in the category. This decreases likelihood that classes overlap on unseen objects.

  4. Classes may still overlap on given objects. CLUSTER/2 includes an algorithm for adjusting overlapping definitions.

  5. Using a distance metric, select an element closest to the center of each class. The distance metric could be similar to the similarity metric discussed above.

  6. Using these central elements as new seeds, repeat steps 1–5. Stop when clusters are satisfactory. A typical quality metric is the complexity of the general descriptions of classes. For instance, a variation of Occam's Razor might prefer clusters that yield syntactically simple definitions, such as those with a small number of conjuncts.

7. If clusters are unsatisfactory and no improvement occurs over several iterations, select the new seeds closest to the edge of the cluster, rather than those at the center.



- Reinforcement Learning
  - A moment of thought, however, reminds us that feedback from our actions in the world is not always immediate and straightforward.
- ✓ The Components of Reinforcement Learning
  - In reinforcement learning, we design computational algorithms for transforming world situations into actions in a manner that maximizes a reward measure. Our agent is not told directly what to do or which action to take; rather, the agent discovers through exploration which actions offer the most reward.
  - Reinforcement learning is not defined by particular learning methods, but by actions within and responses from an environment. Any learning method creating this interaction is an acceptable reinforcement learning method. Reinforcement learning is also not supervised, as seen throughout our chapters on machine learning. In supervised learning, a "teacher" uses examples to directly instruct or train the learner. In reinforcement learning, the learning agent itself, through trial, error, and feedback, learns an optimal policy for accomplishing goals within its environment.

Many of the problem-solving algorithms presented earlier in this book, including planners, decision makers, and search algorithms can be viewed in the context of reinforcement learning. For example, we can create a plan with a teleo-reactive controller (Section 7.4) and then evaluate its success with a reinforcement learning algorithm. In fact, the DYNA-Q reinforcement algorithm (Sutton 1990, 1991) integrates model learning with planning and acting. Thus, reinforcement learning offers a method for evaluating both plans and models and their utility for accomplishing tasks in complex environments.

We now introduce some terminology for reinforcement learning:

t is a discrete time step in the problem-solving process

st is the problem state at t, dependent on st -1 and at -1

at is the action at t, dependent on st

rt is the reward at t, dependent on st -1 and at -1

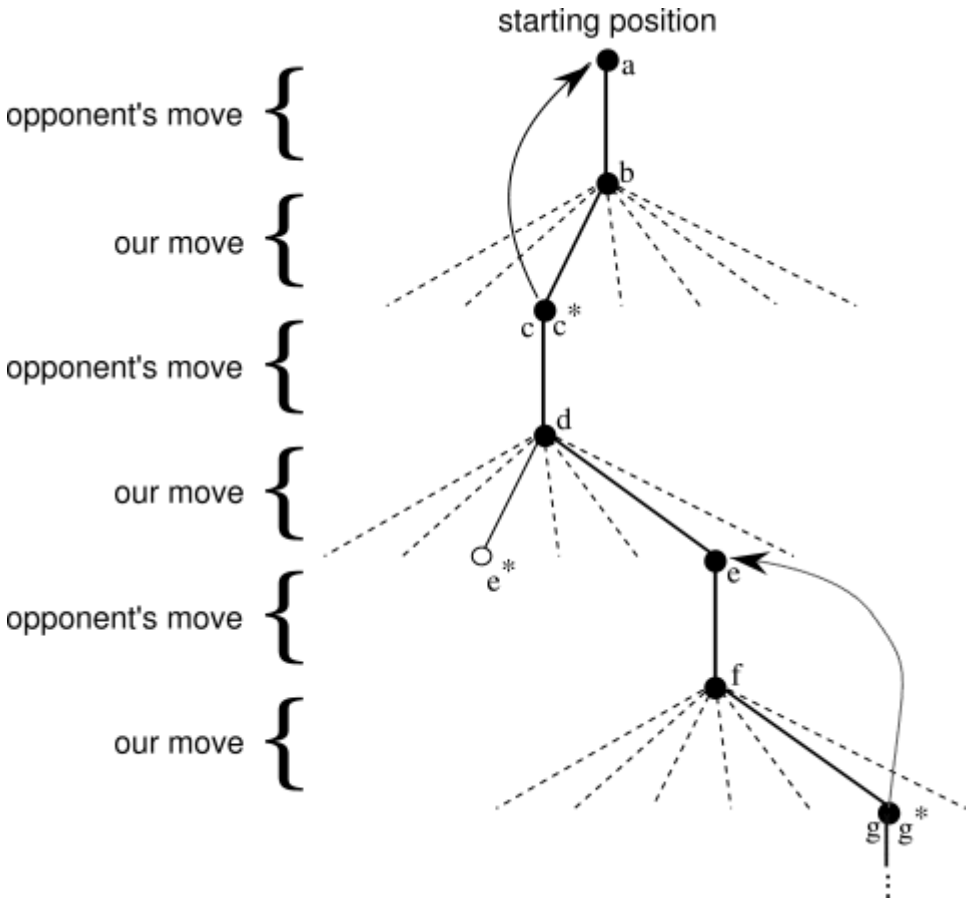
π is a policy for taking an action in a state. Thus, π is a mapping from states to actions

π* is the optimal policy

V maps a state to its value. Thus, Vπ(s) is the value of state s under policy π

An Example: Tic-Tac-Toe Revisited

- We next demonstrate a reinforcement learning algorithm for tic-tac-toe, a problem we have already considered It is important to compare and contrast the reinforcement learning approach with other solution methods, for example, mini-max.



- Dashed arrows indicate possible move choices, down solid arrows indicate selected moves, up solid arrows indicate reward, when reward function changes state s value.
  winning from that state. This will support a policy for a strictly winning strategy, i.e., either a win by our opponent or a drawn game will be considered a loss for us. This drawas-loss approach allows us to set up a policy focused on winning, and is different from our perfect information win-lose-draw model of Section 4.3. This is an important difference, actually; we are attempting to capture the skill of an actual opponent and not the perfect information of some idealized opponent. Thus, we will initialize our value table with a 1 for each

win position for us, a 0 for each loss or drawn board, and a 0.5 everywhere else, reflecting the initial guess that we have a 50 per cent chance of winning from those states.

- ✓ Inference Algorithms and Applications for Reinforcement Learning

- According to Sutton and Barto (1998) there are three different families of reinforcement learning inference algorithms: temporal difference learning, dynamic programming, and Monte Carlo methods. These three form a basis for virtually all current approaches to reinforcement learning. Temporal difference methods learn from sampled trajectories and back up values from state to state.
- Dynamic programming methods compute value functions by backing up values from successor states to predecessor states. Dynamic programming methods systematically update one state after another, based on a model of the next state distribution. The dynamic programming approach is built on the fact that for any policy π and any state s the following recursive consistency equation holds

$V\pi(s) = \Sigma\pi(a \mid s) * \Sigma\pi(s \rightarrow s \mid a) * (Ra(s \rightarrow s) + \gamma(V\pi(s)) \; a \; s \; \pi(a \mid s)$

$\pi(a \mid s)$ is the probability of action a given state s under stochastic policy π. $\pi(s \rightarrow s \mid a)$ is the probability of s going to s under action a. This is the Bellman (1957) equation for Vπ.

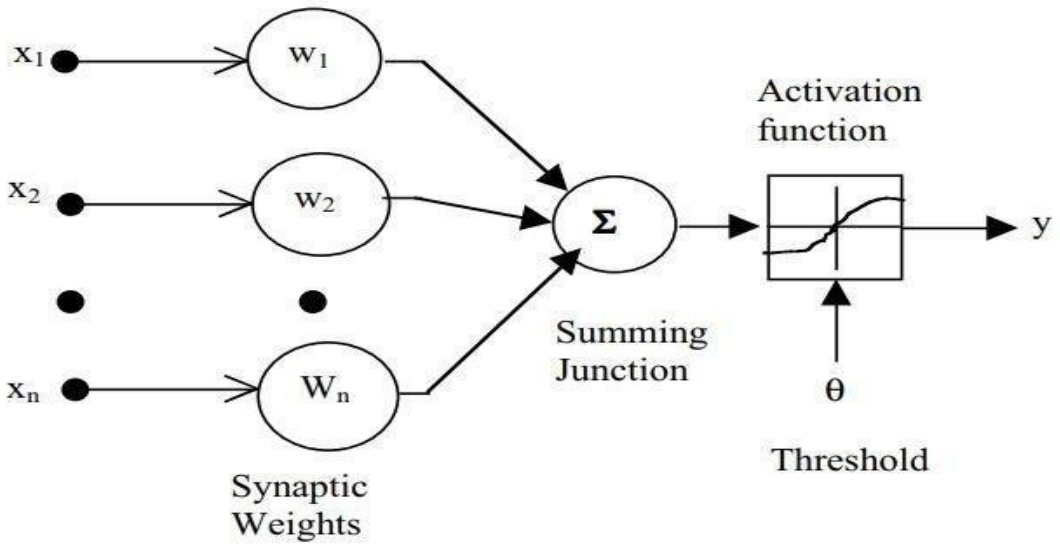
- ➢ **MACHINE LEARNING-CONNECTIONIST:**

- ✓ Foundation for Connectionist Networks

Connectionist architectures are often thought of as a recent development, however we can trace their origins to early work in computer science, psychology, and philosophy. John von Neumann, for example, was fascinated by both cellular automata and neurally inspired approaches to computation. Early work in neural learning was influenced by psychological theories of animal learning, especially that of Hebb (1949). In this section, we outline the basic components of neural network learning, and present historically important early work in the field

The basis of neural networks is the artificial neuron, as in Figure An artificial neuron consists of:

Input signals, xi. These data may come from the environment, or the activation of other neurons. Different models vary in the allowable range of the input values; typically, inputs are discrete, from the set {0, 1} or {−1, 1}, or real numbers.



An artificial neuron, input vector xi, weights on each input line, and a thresholding function f that determines the neuron s output value. Compare this figure with the actual neuron.

A set of real valued weights, wi. The weights describe connection strengths.

An activation level Σwi xi. The neuron's activation level is determined by the cumulative strength of its input signals where each input signal is scaled by the connection weight wi along that input line. The activation level is thus computed by taking the sum of the weighted inputs, that is, Σwi xi .

A threshold function, f. This function computes the neuron's final or output state by determining how far the neuron's activation level is below or above some threshold value. The threshold function is intended to produce the on/off state of actual neurons.

In addition to these properties of individual neurons, a neural network is also characterized by global properties such as:

The network topology. The topology of the network is the pattern of connections between the individual neurons. This topology is a primary source of the net's inductive bias.

The learning algorithm used. The encoding schemes. This includes the interpretation placed on the data to the network and the results of its processing.

The earliest example of neural computing is the McCulloch−Pitts neuron (McCulloch and Pitts 1943). The inputs to a McCulloch−Pitts neuron are either excitatory (+1) or inhibitory (−1). The activation function multiplies each input by its corresponding weight and sums the results; if the sum is greater than or equal to zero, the neuron returns 1, otherwise, −1. McCulloch and Pitts showed how these neurons could be constructed to compute any logical function, demonstrating that systems of these neurons provide a complete computational model.

PERCEPTION LEARNING:

- ✓ The Perceptron Training Algorithm
    - Frank Rosenblatt (1958, 1962) devised a learning algorithm for a type of single layer network called a perceptron. In its signal propagation, the perceptron was similar to the McCulloch−Pitts neuron. The input values and activation levels of the perceptron are either −1 or 1; weights are really valued. The activation level of the perceptron is given by summing the weighted input values, Σxi wi. Perceptrons use a simple hard-limiting threshold function, where an activation above a threshold result in an output value of 1, and −1 otherwise. Given input values xi, weights wi, and a threshold, t, the perceptron computes its output value as:

1 if Σxi wi t −1

if Σxi wi < t

- The perceptron uses a simple form of supervised learning. After attempting to solve a problem instance, a teacher gives it the correct result. The perceptron then changes its weights in order to reduce the error
- The following rule is used. Let c be a constant whose size determines the learning rate and d be the desired output value. The adjustment for the weight on the ith component of the input vector, Δwi, is given by

$\Delta wi = c\ (d - sign\ (\Sigma xi\ wi))\ xi$

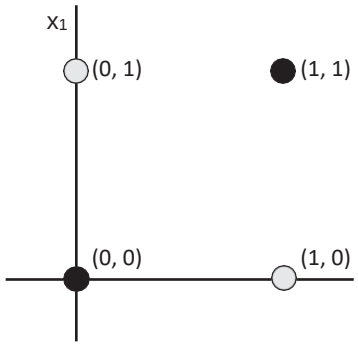
- The sign ($\Sigma xi\ wi$) is the perceptron output value. It is +1 or −1. The difference between the desired output and the actual output values will thus be 0, 2, or −2. Therefore, for each component of the input vector:

  If the desired output and actual output values are equal, do nothing.

  If the actual output value is −1 and should be 1, increment the weights on the ith line by 2cxi

  If the actual output value is 1 and should be −1, decrement the weights on the ith line by 2cxi.

- This procedure has the effect of producing a set of weights which are intended to minimize the average error over the entire training set. If there exists a set of weights which give the correct output for every member of the training set, the perceptron learning procedure will learn it (Minsky and Papert 1969).



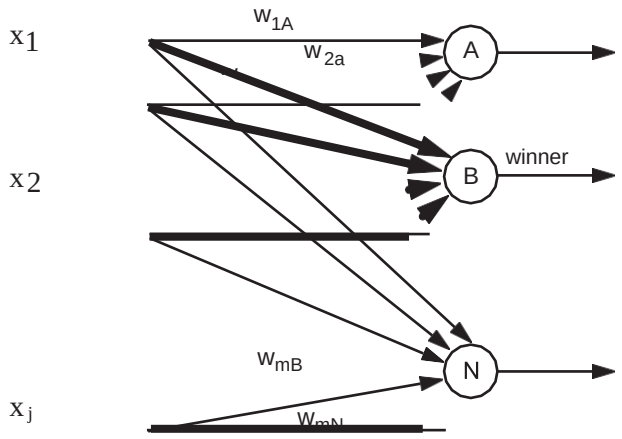
The exclusive-or problem. No straight line in two-dimensions can separate the (0, 1) and (1, 0) data points from (0, 0) and (1, 1).

- **COMPETITIVE LEARNING:**

  - The winner-take-all algorithm (Kohonen 1984, Hecht-Nielsen 1987) works with the single node in a layer of nodes that responds most strongly to the input pattern. Winner-take-all may be viewed as a competition among a set of network nodes, as in Figure 11.13. In this figure we have a vector of input values, $X= (x_1, x_2, ..., x_m)$, passed into a layer of network nodes, A, B,..., N. The diagram shows node B the winner of the competition, with an out- put signal of 1.
  - Learning for winner-take-all is unsupervised in that the winner is determined by a "maximum activation" test. The weight vector of the winner is then rewarded by bringing its components closer to those of the input vector. For the weights, W, of the winning node and components X of the input vector, the increment is:

$\Delta W^{t} = c\ (X^{t-1} \square W^{t-1})$,

where c is a small positive learning constant that usually decreases as the learning pro- ceeds. The winning weight vector is then adjusted by adding ΔW^t.



A layer of nodes for application of a winner-take-all algorithm. The old input vectors support the winning node

- This reward increments or decrements each component of the winner's weight vector by a fraction of the $x_i - w_i$ difference. The effect is, of course, to make the winning node match more closely the input vector. The winner-take-all algorithm does not need to directly compute activation levels to find the node with the strongest response. The activa- tion level of a node is directly related to the closeness of its weight vector to the input vec- tor. For a node i with a normalized weight vector W_i, the activation level, W_iX, is a function of the Euclidean distance between W_i and the input pattern X. This can be seen by calculating the Euclidean distance, with normalized W_i:

$$\| X - W_i \| = \sqrt{(X - W_i)^2} = \sqrt{X^2 - 2XW_i + W_i^2}$$

From this equation it can be seen that for a set of normalized weight vectors, the weight vector with the smallest Euclidean distance, $\| X - W \|$, will be the weight vector with the maximum activation value, WX. In many cases it is more efficient to determine the winner by calculating Euclidean distances rather than comparing activation levels on normalized weight vectors.

- We consider the "winner-take-all" Kohonen learning rule for several reasons. First, we consider it as a classification method and compare it to perceptron classification. Sec- ond, it may be combined with other network architectures to offer more sophisticated models of learning. We look at the combination of Kohonen prototype learning with an outstar, supervised learning network. This hybrid, first proposed by Robert Hecht-Nielsen (1987, 1990), is called a *counterpropagation* network.

✓ Outstar Networks and Counterpropagation

To this point we considered the unsupervised clustering of input data. Learning here requires little *a priori* knowledge of a problem domain. Gradually detected characteristics of the data, as well as the training history, lead to the identification of classes and the dis- covery of boundaries between them. Once data points are clustered

according to similari- ties in their vector representations, a teacher can assist in calibrating or giving names to data classes. This is done by a form of supervised training, where we take the output nodes of a "winner-take-all" network layer and use them as input to a second network layer. We will then explicitly reinforce decisions at this output layer.

we know exactly what values we want each winning node of the Kohonen net to map to on the output layer of the Grossberg net, we could directly set those values. To demonstrate outstar learning, however, we will train the net using the formula just given.

If we make the (arbitrary) decision that node S on the output layer should signal safe situ- ations and node D dangerous, then the outstar weights for node A on the output layer of the Kohonen net should be [1, 0] and the outstar weights for B should be [0, 1]. Because of the symmetry of the situation, we show the training of the outstar for node A only.

The Kohonen net must have stabilized before the Grossberg net can be trained. We demonstrated the Kohonen convergence of this same net The input vec-tors for training the A outstar node are of the form [x_1, x_2, 1, 0]. x_1 and x_2 are values from Table that are clustered at Kohonen output node A and the last two components indi- cate that when A is the Kohonen winner, safe is "true" and dangerous is "false," as

We initialize the outstar weights of A to [0, 0] and use .2 as the learning constant:

W^1 = [0, 0] + .2[[1, 0] □ [0, 0]] = [0, 0] + [.2, 0] = [.2, 0]

W^2 = [.2, 0] + .2[[1, 0] □ [.2, 0]] = [.2, 0] + [.16, 0] = [.36, 0]

W^3 = [.36, 0] + .2[[1, 0] □ [.36, 0]] = [.36, 0] + [.13, 0] = [.49, 0]

W^4 = [.49, 0] + .2[[1, 0] □ [.49, 0]] = [.49, 0] + [.10, 0] = [.59, 0]

W^5 = [.59, 0] + .2[[1, 0] □ [.59, 0]] = [.59, 0] + [.08, 0] = [.67, 0].

with training these weights are moving toward [1, 0]. Of course, since in this case elements of the cluster associated with A always map into [1,0], we could have used the simple assignment algorithm rather than used the averaging algorithm for training.

- ✓ Support Vector Machines

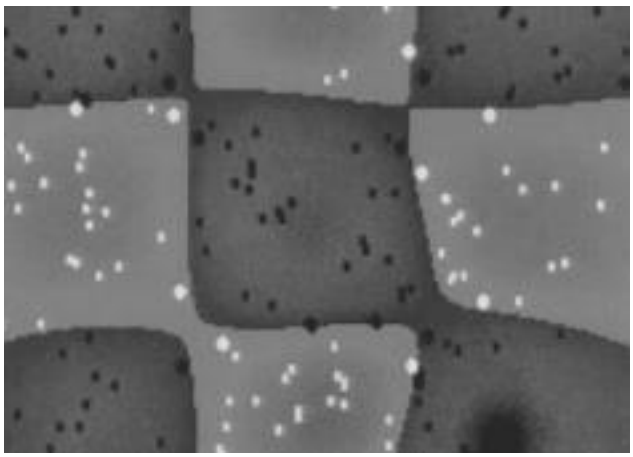
  - Support vector machines (SVM), offer another example of competitive learning. In the support vector approach, statistical measures are used to determine a minimum set of data points (the support vectors) that maximally separate the positive and negative instances of a learned concept.

  - These support vectors, representing selected data points from both the positive and negative instances of the concept, implicitly define a hyperplane separating these two data sets.
  - The support vector machine is a linear classifier where the learning of the support vectors is supervised. The data for SVM learning is assumed to be produced indepen- dently and identically from a fixed, although unknown, distribution of data.
  - The hyper- plane, implicitly defined by the support vectors themselves, divides the positive from the

negative data instances. Data points nearest the hyperplane are in the *decision margin* (Burges 1998). Any addition or removal of a support vector changes the hyperplane boundary. As previously noted, after training is complete, it is possible to reconstruct the hyperplane and classify new data sets from the support vectors alone.

- The support vector machine is a linear classifier where the learning of the support vectors is supervised. The data for SVM learning is assumed to be produced indepen- dently and identically from a fixed, although unknown, distribution of data.
- The hyper- plane, implicitly defined by the support vectors themselves, divides the positive from the negative data instances. Data points nearest the hyperplane are in the *decision margin* (Burges 1998).
- Any addition or removal of a support vector changes the hyperplane boundary. As previously noted, after training is complete, it is possible to reconstruct the hyperplane and classify new data sets from the support vectors alone.

- The SVM algorithm classifies data elements by computing the distance of a data point from the separating hyperplane as an optimization problem. Successfully controlling the increased flexibility of feature spaces, the (often transformed) parameters of the instances to be learned requires a sophisticated theory of generalization. This theory must be able to precisely describe the features that have to be controlled to form a good generalization.



- A SVM learning the boundaries of a chess board from points gen- erated according to the uniform distribution using Gaussian ker- nels. The dots are the data points with the larger dots comprising the set of support vectors, the darker areas indicate the confi- dence in the classification.

- Thus, VC theory provides a distribution free bound on the generalization of the consistent hypothesis (Cristianini and Shawe-Taylor 2000). The SVM algorithm uses this VC theory to compute the hyperplane and controls the margin of error for the generalizations accracy, sometimes called the *capacity* of the

function

- SVMs use a dot product similarity measure to map data from a feature space. Dot product results representing mapped vectors are linearly combined by weights found by solving a quadratic program (Scholkopf et al. 1998).
- A kernel function, such as a polyno- mial, spline, or Gaussian, is used to create the feature vector mapping, where kernel choice is determined by the problem distribution. SVMs compute distances to determine data element classification. These decision rules created by the SVM represent statistical regularities in the data.
- SVMs are best suited to problems with numerical data rather than categorical; as a result, their applicability for many classic categorization prob- lems with qualitative boundaries is limited. Their strength lies in their mathematical foun- dations: minimization of a convex quadratic function under linear inequality constraints.
