85-32

INDUCTION, OF AND BY PROBABILITY*

Larry Rendell

Department of Computer Science,
University of Illinois at Urbana–Champaign,
1304 West Springfield Avenue, Urbana, Illinois 61801

This paper examines some methods and ideas underlying the author's successful *probabilistic learning systems (PLS)*, which have proven uniquely effective and efficient in generalization learning or *induction*. While the emerging principles are generally applicable, this paper illustrates them in heuristic search, which demands noise management and incremental learning. In our approach, both task performance and learning are guided by probability. Probabilities are incrementally normalized and revised, and their errors are located and corrected.

1. INTRODUCTION

Recent work on uncertainty in mechanized inference has shown that various methods for updating beliefs are more stongly related than commonly realized [Ho 86, Wi 86]. While it is not obvious which techniques will become standard in AI, established probability theory may be the best starting point for both analysis and implementation [Ch 85, Go 83]. For reasons of economy, however, learning systems must compress a whole probability distribution into some concise form. One condensation is just a pair of numbers: the belief (primary probability) and some measure of its confidence (second order probability) [Re 86]. In terms of objective probability, such abstractions are unequivocally defined, but in a practical system, beliefs and confidences are partly subjective, and there may be several ways to determine them.

1.1. Belief and Uncertainty in Induction

Recent research has considered the propagation of beliefs in deductive inference, but uncertainty is at least as important in induction. Induction is the compression of data into some concise and meaningful form [Wa 69]. More precisely, induction is the creation of abstractions or sets called *categories* or *classes;* classes are produced from data called *events, patterns,* or *objects.* An object might be a visual grid, the state of a checker board, or countless other items of interest, usually involving performance of some task. Objects within a class are similar for purposes of the current task or goal. Because a class description or *concept* embodies not only observed objects, but also similar objects yet to be encountered, induction is predictive. Prediction is one of the main characteristics of intelligence.

For several reasons, induction is computationally difficult. First, the number of potential *hypotheses* (candidate concepts) is very large, compared with the number of explicit descriptions that can be explored in reasonable time. Hypotheses are hard to distinguish, since data are usu-

* This work was supported in part by an operating grant from the Natural Sciences and Engineering Research Council of Canada.

ally sparse. Further, the data are often noisy or inadequately described. Finally, not all the data may be available at once, so incremental learning is required; this suggests that multiple hypotheses should be continually developed, refined, assessed, and compared.

An hypothesis is assessed as an expression of the correct concept, based on evidence (data), and also on biases (subjective models or extra–evidential criteria). The combined measure of our befief in an hypothesis is its *credibility*. Because there are multiple sources of knowledge which influence credibility, and because it should be updated as new evidence becomes available, uncertainty in induction is a complex problem.

Early work in pattern recognition accounted for probability and included subtleties such as techniques for combining several sources of belief, both evidential and extra–evidential [Wa 69]. In contrast, much of the machine learning research in artificial intelligence, while permitting symbolic representation, has avoided uncertainty [Di 82, Mi 83], although there are some exceptions such as [Ha 86, Qu 83, Re 83a]. In an attempt to unify some of the AI and pattern recognition work, [Re 86] gives a general framework for probabilistic induction and examines some emerging principles.

1.2. Incremental, Probabilistic Induction, and an Illustrative Domain

The present paper addresses the problem of efficient probabilistic and incremental induction. As shown in [Re 86], induction can be considered as the problem of learning a function. Because induction normally has some purpose or goal, we call the associated function a *utility* function. Later we shall define the utility as the probability of goal attainment.

The exact form of the utility function u depends on the particular task and approach. In simpler cases, u is expressed in terms of variables which describe high level attributes or *features* of objects. When u is well behaved, having few maxima, objects neighboring in feature space have similar u values, and this permits the relatively straightforward techniques of *selective induction* such as discriminant analysis and hyperrectangle formation [Re 86]. For example, "single concept" learning usually discovers a simple Boolean function having few disjunctions [Di 82]. More elaborately, management of uncertainty requires a multiple–valued or continuous utility function u; u may be probabilistic if it has values in $[0, 1]$. The problem of new terms (constructive induction or variable transformation) is very difficult; it corresponds to a multi–modal utility function. In the case of either selective or constructive induction, incremental learning is easier if the standard representation of u is probabilistic (updating may then be gradual).

In this paper, we examine some fairly general methods for incremental and uncertain induction, illustrating them mainly in the case where induction is selective and the utility function u is an evaluation function H for heuristic search. Here there are really two kinds of search which alternate. In the primary task, states chosen by H are transformed into others by applying operators (e.g. moves in a game), and the purpose is to find a goal state (e.g. a win). In the learning phase, search takes place to find a useful function H which will lead to a solution in the task domain. (If this induction is to be selective, H must be expressed in terms of high level features such as piece advantage and center control.) We shall discuss efficient and effective methods for learning such a function.

Designers of early AI programs for learning heuristics often employed statistical methods [Ne 65, Sa 63, Sa 67, Sl 68]. These methods utilize probability or a related measure in two ways. First, the probability p that a state contributes to success in the task domain becomes the basis for heuristic search there—the evaluation function H estimates p. Secondly, information about the variation of p with the features of a state helps to determine the parameters and form of H—i.e. p also plays a part in the inductive search for H. These roles of p alternate in an iterative, two stage algorithm: interpolated p values guide task search, then observed p values guide inductive search.

1.3. Some Basic PLS Methods and Capabilities

Based on probabilistic techniques, partly established and partly novel, this paper describes some advances implemented in the author's *probabilistic learning systems* (PLS). The original system *PLS1* has achieved unique results, efficiently converging to optimal search heuristics when the user supplies features [Re 83a, Re 83b, Re 85a]. A newer, radically different system *PLS0* attempts the difficult problem of constructing features for heuristic functions, given only elementary state descriptions [Re 85b].

Some readers may find certain aspects of PLS1 familiar. Classes are represented as prototypes or centroids, and (simultaneously) as hyperrectangles or logical conjunctions of feature ranges. Disjunctions of rectangles are allowed. Classes are optimally discriminated according to an inductive criterion which is essentially information–theoretic. On the other hand, some PLS methods used to represent and induce a class description are unusual. For reasons given later, concepts are both prototypes and hyperrectangles. To accommodate uncertainty and noise, the inductive criterion incorporates both probability and its error. To facilitate incremental learning, classes of probability are clustered, updated, and refined. Errors and biases are located and corrected.

The next section of this paper details the problem of uncertain induction, and suggests some knowledge structures for probabilistic incremental learning. These include double representation (class extent and class prototype), and its dual meaning (probability partition and utility function). The third section outlines some PLS methods, including normalization of biased data, location of errors, and refinement of the probability function. The fourth section briefly considers probability as a factor in effective and efficient induction.

2. REPRESENTATIONS FOR INDUCTIVE LEARNING

When induction converts individual patterns or objects into classes or concepts, both the objects and the concepts must be expressed using some language, such as feature space representations, logic, grammars, etc. The language may limit expressible concepts, for example the feature space model is a restriction of logic which cannot express many concepts [Bi 77]. The concepts not covered are those for which structure must be learned, or for which a large number of disjunctions are required. Using a feature space representation alone, the only feasible induction is through aggregation of neighboring points as shown in Fig. 1 [An 73]. Here concepts are local neighborhoods of objects; this selective induction is the simplest form of "implicit disjunction"

The reason for any language restriction is pragmatic. To induce concepts in practice (at least with current understanding), very general representation languages such as the predicate logic cannot be used in full, because of the extreme computational complexity. However, there are indications that a controlled introduction of successively more powerful means of concept formation are practical. This involves a variable language constraint or bias, and multiple and interrelated representations [Re 85b]. In most of this paper, though, we shall consider just a feature space restriction of logic. Even so, we need flexible representation: the form of a concept should accommodate degrees of precision, which should be modifiable during incremental learning.

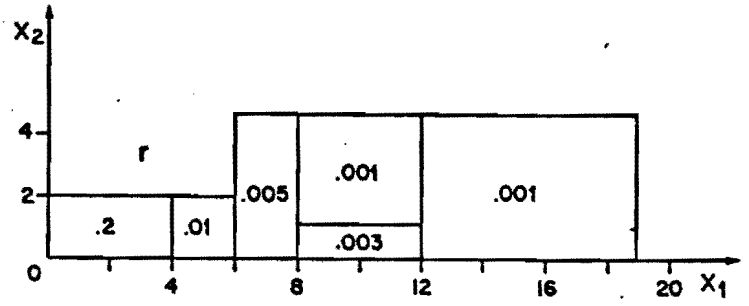

Figure 1. Deterministic versus probabilistic induction. Ideally, classes can be neatly differentiated into positive and negative instances (a). But more commonly, exceptions occur (b). In less extreme cases, these are just anomalies whose effects can be recorded using proportions as shown here.

2.1. Deterministic versus Probabilistic Concept Expression

In a feature space representation, an object is a vector $x = (x_1, x_2, ..., x_n)$, where n is the number of features x_i. In vision for example, these features might be the light intensity (gray level) for a total of n squares (pixels) of a grid representing an image. If images contain some symbol of interest, each vector observed becomes a positive or negative instance of the concept "contains the symbol". In a game such as checkers, the ultimate concept is "winning positions". Objects could be expressed as vectors of high level features such as piece advantage, center control, etc.; or, instead, objects might be represented more basically as states, i.e. as vectors describing the contents of individual squares. However, for the techniques of selective induction to be adequate, the data objects must be expressed in terms of high level features.

In supervised learning, the data provide positive and negative training examples of a concept or class. Fig. 1(a) shows a simple example, where the class is quite regular, and missing instances are predicted by selective induction (e.g. insertion of decision boundaries, or clustering into hyperrectangles [An 73, Du 73, Re 86, To 74]).

Instead of just two categories (positive and negative), the presence of uncertainty requires gradation, expressible using multiple classes as in Fig. 1(b).[1] These classes might represent a narrow range of probabilities p. Associated with each p is some set S_p of feature values; p can be considered the name of class S_p. S_p often has some concise description, as in Fig. 2, where the leftmost cell is the hyperrectangle $r = S_p$, compactly expressible as $(0 \leq x_1 \leq 4) \cap (0 \leq x_2 \leq 2)$. Here r represents the class having probability $p(r) = 0.2$ of occurrence of some class C, i.e.

p(r) is the conditional probability $Pr(C|r)$ of C, given that object $x \in r$. (An example is shown in Fig. 3.)

From one point of view, the number of classes is the number of p values, and the induction is partially unsupervised (this is the clustering problem applied to probability classes). From another point of view, we have returned to the situation in which there are only two classes (success or not). In the latter view, we have a discrete function p(r) indicating the probability of membership in the "success" class. The function p(r) can also be considered as the degree of class membership [Re 86].

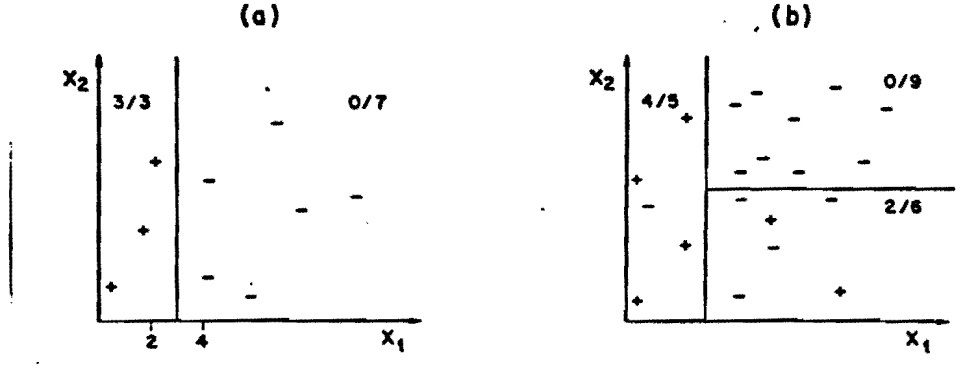

Figure 2. Probability classes, a step function, or a set of sub-hypotheses. Feature space can be partitioned (clustered) into rectangular cells r, each having a roughly similar probability p of implying some concept C. The p values shown inside each r represent $Pr(C|r)$. The set $\{(r,p)\}$ can be considered a step function of feature vector x whose values are given by p. This set of probability classes is also a disjoint and exhaustive set of sub-hypotheses for C.

2.2. Economy through Step Functions or Sub-Hypotheses

The problem of inducing a concept C is the problem of learning a utility function u(x) (where x is a vector of features and u is the probability of class membership). In deterministic learning, u is Boolean; in probabilistic learning, u has real values in $[0, 1]$. An arbitrary function is very hard to learn, although our current assumption is that u is not too irregular (i.e. that u varies gradually with x, so techniques of selective induction may be used).

In Fig. 2, we approximate u(x) by using a step function p(r); in learning u, a program may improve its estimate of u by refining the feature space partition $\{r\}$, and by updating p(r) for each r (as in Fig. 6, discussed later). Like finding linear discriminant functions, learning hyperrectangles is efficient [Bi 77]; moreover their storage requirements are small [Re 77]. As we shall see in §3, the entire partition $\{r\}$ may be optimized so that boundaries maximize differences in p. Hence the entire function representation is economic (cf. [WB 68]).

Representing the utility function u as a step function has still another important advantage. Each discrete step or cell of the partition is also a sub-hypothesis of the complete hypothesis or candidate function u. If a learning algorithm is available which can generate and test these individual components, the task of learning u becomes simpler and faster. This divide and conquer approach is part of PLS.

2.3. Structures for Probabilistic Concepts

Instead of recording a rectangle r for a probability class, we could store some other information to express the concept. One possibility is a representative of r, its *prototype* or *centroid* c [Du 73]. Additional information may also be useful: to express our confidence in p, we could include its error e. Let us call an association of elements such as (r, p, e, c) a *region* if it contains r and p. Sometimes we shall abbreviate regions as triples (r, p, e) or as pairs (r, p) if the extra elements are unimportant in the current context.

One of the unusual aspects of our systems is that their regions include both cells *and* centroids. PLS uses cells for various reasons. Incremental learning demands progressively refined knowledge, and cells facilitate refinement. Moreover, rectangles r are easy to express, and convey meaning well: conjunctions of feature ranges define classes concisely (Fig. 2). Further, the size of r in a region R = (r, p, e, c) gives an impression of the gradient ∇p (i.e. the rate of change of probability) near c.

But prototypes may also be important. PLS uses centroids c for several reasons. This vector c, along with the p value of a region R, define one datum in a regression to determine a smoothed estimate of the utility function u(x). (In our current application to heuristic search, u is an evaluation function for interpolated state evaluation.) In one mode of system operation, c allows sophisticated measurement of the proximity of a state in feature space: the proximity measure weights features according to their "importance" near the prototype c [Re 83c].

PLS1 performs several operations on regions, including reclassification (of probability category p), differentiation (refinement or splitting of cells r), generalization (merging cells r), and reorganization (of {r}) [Re 85a]. Because PLS1 is complex, we shall consider just part of its structure and just two of its operators: cell splitting and probability reclassification, although some other aspects will be mentioned briefly.

3. PROBABILITY AS PRODUCT OF LEARNING

The probabilistic learning system PLS can be used for "single concept" learning, like the systems described in [Di 82], but most testing has been in the more complex domain of heuristic search. Here noise arises because of inadequate features, search anomalies, and changing environments.

According to §2, PLS1 expresses and induces uncertain utility (probability of success) as a function of features. In search, success means leading to discovery of a good solution or win. If feature space rectangle r contains a vector x which represents a state in a problem or a game, then the function H(x) = p(r) is the probability that the state x$\in$r will be useful (its utility). H may be used as a discrete *evaluation function (heuristic function)* to assess state x (see Figs. 2, 3).

Instead of the discrete function H(x) = p(r), an alternative form for the heuristic is the linear combination H(x) = b.x, where b is the *coefficient vector.* Here regions ($r, \hat{p}, e, c$) are used as data in weighted stepwise regression to determine b. (Providing that each r is small enough, a discrete function p(r) is fully general, whereas the linear combination b.x of course is not.) PLS1 uses both forms of H [Re 83a, Re 83b, Re 85a], and also piecewise linear functions [Re83c].

Results using PLS1 include efficient convergence to optimal evaluation functions H. Novel aspects of PLS include incremental revision (reclassification) of probability estimates $\hat{p}(r)$ during task performance, appropriate refinement of feature space cells r, and normalization of heavily biased samples. These and other aspects are examined below.

3.1. Definition and Use of Conditional Probability

In the supervised learning of a single class, the definition of success probability is clear: a training sample is in the class or not, and therefore contributes 1 or 0 in a success count for its corresponding feature space cell r (Fig. 1). In problem solving, however, there are complications.

One PLS method for learning H begins as follows. Given an (initially trivial) H and a set S of sample problems, attempt their solution, to compute a corresponding set {T} of search trees. (There must be at least one solution in {T} if any useful information is to be extracted.) For feature space cell r, the probability of success p(r) is the number of states x within r found on solution paths in {T}, divided by the total number of states x within r. (A similar probability is defined for games in [Re 83b].) These conditional probabilities $p(r) = Pr(\text{success}|r)$ are illustrated in Fig. 3.

An ideal measure of state quality would be p(r) for *undirected (breadth-first)* search, *very large, random* S, and *very small* r. Unfortunately, resource constraints preclude all three ideals. One alternative is to obtain probabilities using informed and progressively improving heuristics H, and then to normalize to standard, breadth-first values. This involves probability estimates $\hat{p}(r)$, initially obtained for easy S and larger r, and then for progressively harder S's and gradually refined r's. By making use of improving $\hat{p}(r)$ values, H may be bootstrapped. This whole approach is complex, however.

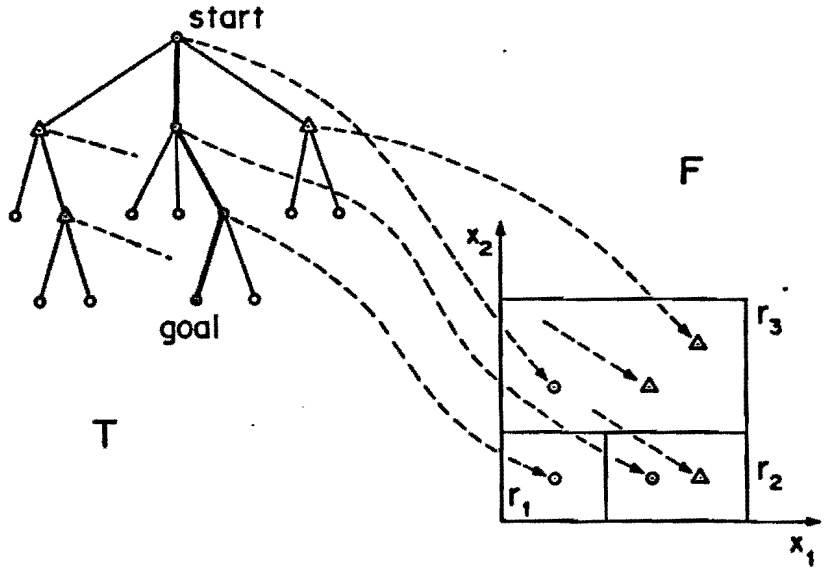

Figure 3. Probabilities computed from a search tree T. Nodes x, developed in T are mapped into feature space F. Here the (arbitrary) rectangular partition gives three values: $p(r_1) = 1/1 = 1.0$, $p(r_2) = 1/2 = 0.5$, and $p(r_3) = 1/3 = 0.3$. These pairs or *regions* (r_i, p_i) may be used to define a discrete heuristic function, or they may become data to fit a regression model, to represent the probability of a node's being on a solution path.

3.2. Good Use of Scarce Information To Estimate Probabilities

One purpose of induction is to minimize knowledge acquisition costs, so training samples x are usually sparse. The data alone are not sufficient to constrain hypotheses, so a *model* is used. Two models suitable for heuristic search have already been mentioned. One is the linear combination $H(x) = Pr(success|x) = b.x$. A more general model is the discrete function $H(r) = \hat{p}(r)$ of feature space cells r. For either model to be useful, $\hat{p}$ must vary smoothly with x. Then sizable volumes of feature space are meaningful as units, and neighboring coordinates x may be clustered into cells r.[2] PLS1 clusters similar probabilities.

Clustering criteria often measure object similarity only as a function of features, but this can cause problems. Criteria based on something other than features are *external* criteria [An 73, p. 194]. Several years ago the author suggested ***utility similarity*** as a suitable external criterion when clustering relates to performance of some task [Re 77, Re 83a] (cf. [GK 78]). Utility similarity involves the whole task environment, not just features of an individual object. Here we measure utility as probability of success (Fig. 3).

The system employs a splitting algorithm which repeatedly dichotomizes rectangles r using a *dissimilarity* measure d. If p_1 and p_2 are the two probabilities for a tentative dichotomy, and e_1 and e_2 their error *factors*, then $d = |\log p_1 - \log p_2| - \log(e_1 e_2)$. This measure is computed for a number of boundary insertions parallel to feature space axes. If the largest d is positive, the corresponding split is retained. The process is repeated until additional refinement is unwarranted by the data (until $d \leq 0$). Larger d means more *assured* dissimilarity. A formal derivation of our criterion d is based on a statistical analysis [Re 81]. Notice that we do not need to know the distributions of probability values p (although the probability p and its error e comprise a compressed form of this information).

Our probability-based dissimilarity d is related to Quinlan's information-theoretic measure [Qu 83]. The information criterion for a dichotomy is $d' = q_1 \log q_1 + q_2 \log q_2 + m_1 \log m_1 + m_2 \log m_2$, where q_i is the probability of a positive example in subrectangle r_i, and m_i is the probability of a negative example in r_i. This information-theoretic criterion d' is similar to the statistical criterion d without its error term (see [Re 86]). Intuitively we can relate d and d' by saying that splitting governed by probability dissimilarity d lessens our ignorance of success (i.e. knowledge of the discrimination decreases our surprise or increases the information).

Clustering similar probabilities improves their accuracy since it effectively produces larger samples. Since data determine splits, the sizes and shapes of feature space cells or clusters tend to match characteristics of the domain, and splitting occurs only when warranted (see Fig. 2).

PLS may be compared with Samuel's statistical method of adjusting parameters, and with his signature tables [Sa 63, Sa 67]. All these schemes use available information well, allowing every state encountered to influence H stochastically — each useful state increments a "success" count (see Fig. 3). Consequently, PLS and Samuel's checker player are both effective and efficient. PLS is more stable, general, and mechanized [Re 85a].

3.3. Incremental Learning of Probability Estimates

Initially, PLS may possess no heuristic information. To start, easy problems can be presented which are solvable within resource constraints (Fig 4). (In a different mode of operation, the system can be initialized by presenting harder problems, as long as some minimal "start–up" heuristic can be provided [Re 83b].) Problem instances are selected either by the user or by machine. As shown in Fig. 4, training problems are not training examples, rather the system generates its own examples.

The data so obtained are assumed to represent all problems, and the resulting probabilities are used predictively, to form heuristic $H(x) = b.x$ (continuous) or $H(r) = \hat{p}(r)$ (discrete), for a new round of problem solving or game playing. Because H is now improved, harder problems can be solved, and consequently, new probability measurements p(x) become available (called *elementary* probabilities). These are used both to improve (reclassify) $\hat{p}(r)$, and also to refine (split) current cells r (Fig. 6). These two operations are detailed below, but first we shall deal with a phenomenon arising from the incremental nature of PLS.

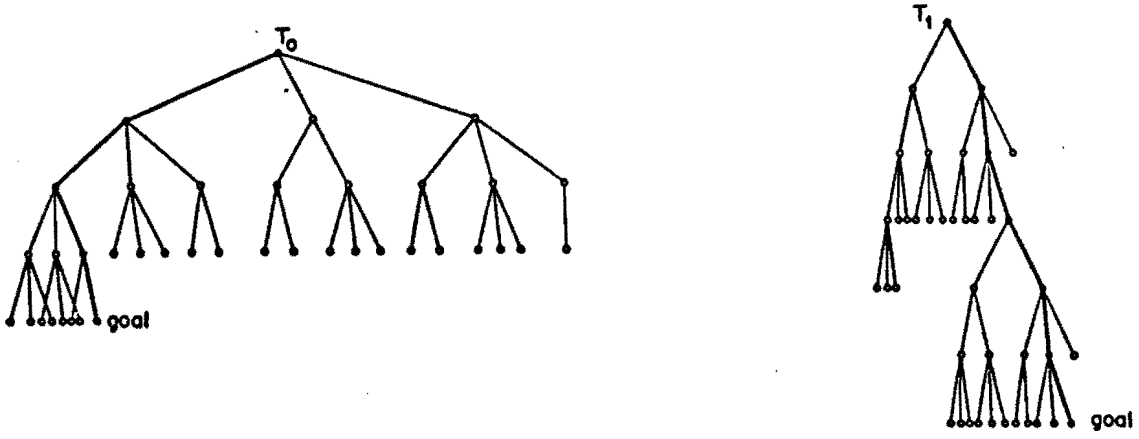

Figure 4. Effects of improving heuristic functions. As knowledge increases over successive iterations of the system, search becomes narrower. T_1 develops as many nodes as T_0 (each 15), but has 50% more "success" nodes. In terms of information available for learning, this means that "success" counts are higher (a greater proportion of the nodes appear on the solution path or game win), which aids learning. However, the same phenomenon biases the probabilities unpredictably.

3.4. Normalization of Inherently Biased Probabilities

When the system computes probabilities p(x), it generally uses a non–trivial heuristic H to sample solutions to difficult problems. These p(x) are heavily biased, they overestimate results relative to unguided (breadth–first) search. Positive bias occurs because fewer useless states are developed when H guides search, and a higher proportion participate in solutions (see Fig 4). The elementary values p(x) must be corrected if commensurate state evaluation and meaningful probability learning are to be possible. Since the basis for unbiasing must be partly extra–evidential, this correction introduces an element of subjectivity into the final probabilities.

Unfortunately, while the bias is known to be positive, its magnitude is unpredictable. Nevertheless, elementary probabilities may be normalized, coarsely in any one system iteration, but effectively overall. Incremental feedback over repeated iterations of the learning system serves to correct errors in rough treatments. One method (a simplification of [Re 83a]) is dep-

icted in Fig. 5. Methods of error detection and correction will be discussed shortly.

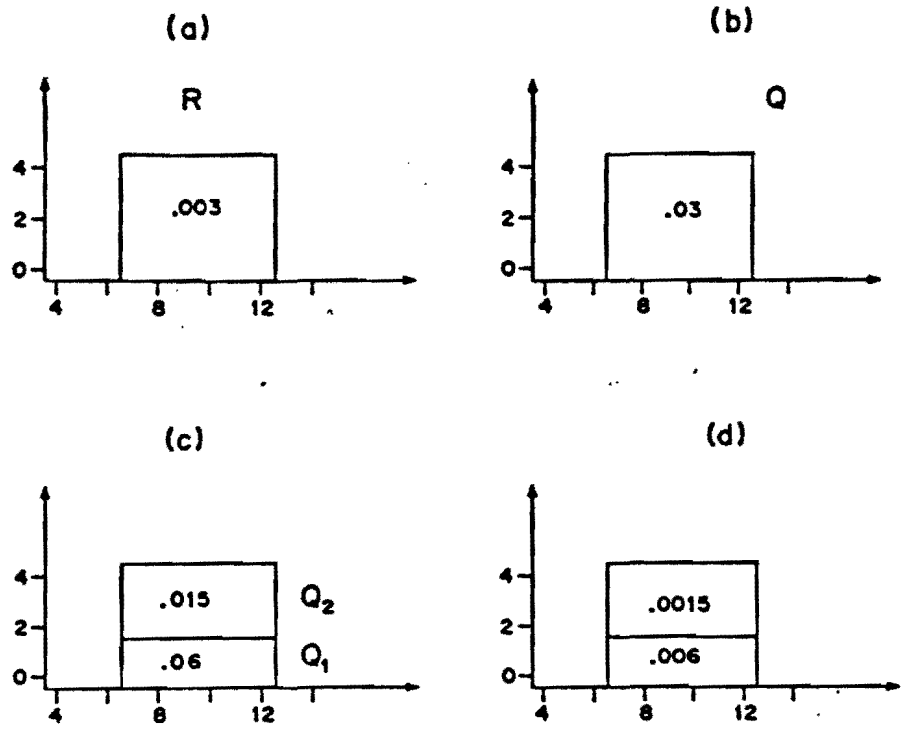

Figure 5. A simplified method of normalising probability estimates. The *cumulative* region R in (a) estimates "true" (unbiased) success probability $\hat{p}_R$. The *elementary* region Q in (b) obtained using informed search has a rectangle matching that of R, and Q has a biased probability estimate p_Q. The ratio of the two probabilities $\hat{p}_R / p_Q = .03/.003 = 10$. A hypothetical outcome of splitting Q is diagrammed in (c). To normalise Q_1 and Q_2, the elementary probabilities can simply be divided by 10, to give (d). This illustrates a simplification of one of the normalization procedures of [Re83a].

3.5. Refinement of Feature Space with Improving Probability Resolution

As more training problems are encountered in successive PLS iterations, domain knowledge accumulates in a *region set*, a set of tuples $(r, \hat{p}, e, c)$, where e is the error in $\hat{p}$, and c is the prototype or centroid. Region set refinement uses the clustering algorithm previously described. Each *established* or *cumulative* region becomes the starting point for further splitting (Fig. 6).

This learning is not just linear accretion of information, but rather an accelerating process guided by experience. Cells r have size and shape determined by the particular domain. These cells are refined (split by the clustering algorithm) whenever significant differences emerge in success probability. Further, as the heuristic H improves, it guides search toward more successful states, and this results in more useful information. More "success" states in turn permit greater contrast in probability (see Figs. 4, 5, 6). The iterative process can be compared to search by a microscope: at first a large area might be scanned, and when something interesting appears, magnification is increased, now omitting surrounding incidentals, and allowing greater resolution of the phenomenon of interest. In our case, the general phenomenon is variation of $\hat{p}$ with its determinants the features, and the phenomenon of particular interest is variation of $\hat{p}$ in volumes of feature space having fast–changing $\hat{p}$ values.

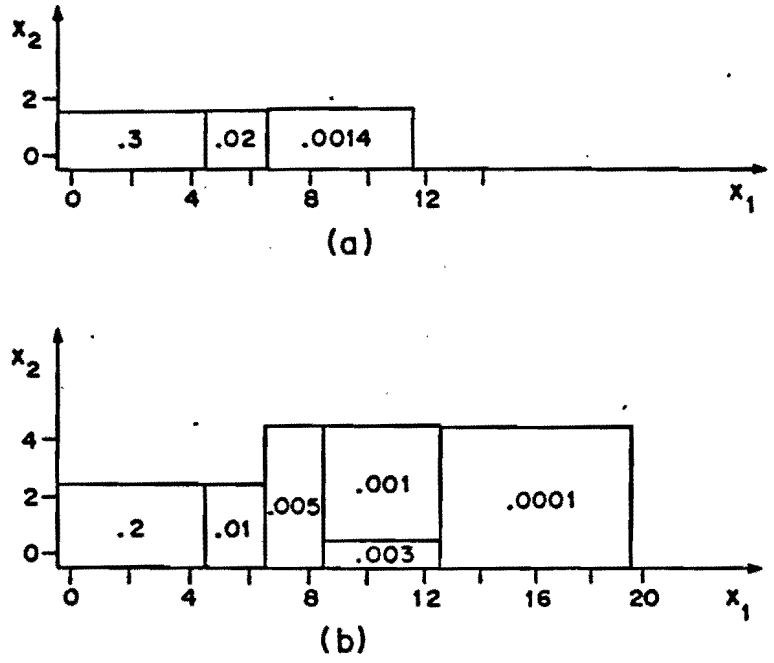

Figure 6. Incremental growth of feature space cells. As information accumulates with continuing task performance, the partition becomes more refined. The volume of feature space covered gradually extends to enclose all data encountered over repeated iterations of the system.

3.6. Improvement of Probability Estimates and Location of Errors

In PLS1, the region set $\{(r, \hat{p}, e, c)\}$ is the fundamental knowledge structure, wherein information accumulates. PLS has several means of detecting errors e and improving probability estimates $\hat{p}$. The simplest method of dealing with errors is to moderate their influence by statistical regression—fitting a linear model $H(\mathbf{x}) = \mathbf{b}.\mathbf{x}$, using the regions $\{(r, \hat{p}, e, c)\}$ as data to find $\mathbf{b}$. This may be repeated anew at every iteration, with the region set the only reservoir of accumulating heuristic knowledge. Regression has the effect of smoothing errors in probabilities $\hat{p}$, which is very important, since one high estimate can vitiate a discrete heuristic function. (A secondary effect of this regression is the increased heuristic power resulting from the interpolation $H(\mathbf{x}) = \mathbf{b}.\mathbf{x}$.)

Another means of improving probability estimates involves the incremental creation of fresh data $\{(\mathbf{x}, p', e', c')\}$ in every new iteration. As shown in Fig. 5, these data are used to generate a (normalized) region set $\{(r, \hat{p}', \hat{e}', \hat{c}')\}$ corresponding to the already available, established set $\{(r, \hat{p}, \hat{e}, c)\}$. For each established rectangle r, the two probability estimates $\hat{p}$ and $\hat{p}'$ are averaged and the error estimate resulting from combining $\hat{e}$ and $\hat{e}'$ is decreased. (This averaging uses error-weighting; see [Re 83a, Re 83b].)

Finally, the most sophisticated kind of error correction involves the genetic learning system *PLS2* which controls parallel activations of the basic system PLS1. PLS2 is able to locate errors in individual regions. As we saw in §2.2, a region is a sub-hypothesis of the entire utility function, and our scheme is to learn regions independently of each other, a divide and conquer approach. In recent experiments, PLS2 has proved stable, efficient and effective [Rc 85a].

In summary, PLS1 and PLS2 use clustering iteratively, to refine knowledge gradually. Because of the progressive improvement in the resulting heuristic, all iterations but the first generate biased probabilities, which must be normalized. The errors involved are large and difficult to estimate, but can be improved through iterative feedback. The system is effective and efficient.

4. PROBABILITY AS INDUCTIVE CRITERION

PLS not only *learns* probability, it is also *guided by* probability. Consider a simple kind of induction, that of learning a single class C, given a set S of objects which are examples of C. This can be an immense task. If the number of objects in the universe is only 100, the number of inductions (ways of forming a class C) is $2^{100} \simeq 10^{10}$.

As Watanabe showed in his "theorem of the ugly duckling", no one classification is intrinsically better than any other [Wa 69]. In other words, in order to select an appropriate class, we must rely on some *external* criterion. This criterion is the quality, our belief in the hypothesis C, or its *credibility,* which expresses some ascribed elegance or purpose of a generalization. The credibility may be considered a probability (that we believe C).

4.1. Two Kinds of Credibility

Watanabe discusses and formalizes two kinds of credibility criteria: *evidential,* and *extra–evidential.* These are also called *confirmation* and *creditation.* Evidential and extra–evidential criteria may be combined, to form a single credibility measure. Evidential criteria correspond to the AI notion of data–drivenness, and extra–evidential criteria correspond to model–drivenness.

The simplest evidential criterion in our case involves the *performance* of an induced heuristic H, viz. the number of nodes developed in search for problem solutions. In PLS2, this does not just confirm H, it assigns credit and blame to individual regions [Re 83b, Re 85a]).

A related aspect of PLS1 involves the task–oriented utility or success probability $\hat{p}$. Since we induce for the purpose of successful heuristic search, we want the product of induction (evaluation function H) to differentiate $\hat{p}$. Differences and similarities in $\hat{p}$ are the basis for the clustering algorithm described earlier.

One extra–evidential criterion used in PLS1 is a form of *simplicity:* feature space cells are rectangular, so little information is needed to specify them. Further, the number of rectangular shapes is a small fraction of the total number of possible classes.

4.2. Credibility Criteria and the Complexity of Induction

Both kinds of credibility can aid the extreme computational problem of induction. For example, the constraint that feature space cells be rectangular serves not only to *select* generalizations, but also to *speed* the inductive process. Smoothness in probability–feature relationships permits sizable cells to capture similarities in $\hat{p}$, and the use of rectangles reduces time complexity of algorithms.

Induction is the realization of regularity, partly discovered and partly imposed. Different kinds of credibility criteria and other constraints may be implemented in various forms, all of which provide regularity and improve efficiency: some components become information structures, some are built into algorithms, and some are parameters of algorithms. Means for effective and efficient induction are examined in [Re 86].

4.3. Probabilistic Structural Induction

As explained in Note 2, we have been dealing solely with selective induction, whereas the great challenge is constructive induction requiring substantial transformation of the original primitive descriptions.

The new system *PLS0* is designed to begin with very detailed primitives such as the contents of individual squares of a checker board. The variation of probability p with primitives is highly irregular, and methods like PLS1 are useless since they rely on uniformity of p within clusters. However PLS0 employs a novel *second order* clustering, which groups not just features, or even probabilities, but rather probability *surfaces* in subspaces of primitives. These surfaces represent interrelationships among components of objects. Second–order clustering *creates structure* [Re 83b, Re 85b]. The system effectively forms a group of transformations or induces rules of a grammar.

Extending the methods discussed in this paper, PLS0 imposes various constraints and credibility measures, and breaks down the problem into several levels, each with a reduced time complexity (e.g. polynomial instead of double exponential) [Re 85b].

5. SUMMARY

This paper has outlined some methods used by our probabilistic learning systems PLS. While these systems are quite general, they have been considered here for the induction of heuristic functions in search.

PLS1/2 clusters task utility in feature space. Utility may be a probability, used both to guide heuristic search and also to induce the heuristic itself. A flexible representation (the region) includes both a prototype and a logic description; this permits efficient and effective processing. Similarly beneficial, and included in a region, are both the probability and its error; these abbreviate the underlying probability distributions. Among the PLS operators are normalization of necessarily biased probabilities, and discovery and correction of their errors.

We have also considered some ways of speeding the required induction. Efficiencies result from exploiting various constraints. The key to practical, efficient and effective induction may be found in certain aspects of the probabilistic variation of utility with features, particularly similarity, accuracy, and credibility. These may be incorporated in effective learning schemes which employ reclassification, refinement, reorganization, and layering. Many seemingly diverse learning systems have incorporated ideas discussed in this paper [Re 86].

NOTES

(1) We induce *probability classes*—i.e. classes of discrete probability values. We are NOT directly concerned with probability distributions of the classes. Instead, we associate probabilities with feature space neighborhoods or cells. There are alternative means of probabilistic induction (see [An73, Du73, To74] for feature space methods and [Fu82, Ka72, Re85b] for structured approaches).

(2) Whenever the task utility (here measured as $\hat{p}$) varies smoothly with features, induction is relatively simple, or *selective* (see [Re86]). Many algorithms rely on smoothness for efficiency. Examples include discriminant functions [Du73, To74], and systems using the restriction of predicate logic VL_1 [Mi83], which corresponds to our rectangular feature space cells. The overriding concern in such algorithms is gradual change of utility (probability) with feature values. For example, in checkers, the quality of a state (probability of its contributing to winning) varies smoothly and monotonically with the abstract features typically used, such as piece advantage, mobility, center control, etc.

Ideally, a learning system would be able to deal with more primitive features which describe objects in detail. In checkers, primitives would be elementary board descriptions giving contents of the 32 squares which can be legally occupied. However, no mechanized learning system has this capability. In an analysis of the difficulty involved, [Re83b] discusses this example as a case requiring *transformation of variables* [To74], also known as the *problem of new terms* [Di82], or *constructive induction* [Mi83]. New descriptions (concepts) must be created which amount to the usual high level features. [Re85b] examines a layered method for reducing the complexity of this problem, and describes some encouraging experiments.

REFERENCES

[An73] Anderberg, M.R., *Cluster Analysis for Applications*, Academic Press, 1973.

[Bi77] Bitner, J.R., Capacity and efficiency of decision functions, *IEEE Transactions on Computers, C-26*, 11 (1977) 1147–1151.

[Ch85] Cheeseman, P., In defense of probability, *Proc. Ninth International Joint Conference on Artificial Intelligence*, 1985, 1003–1009.

[Di82] Dietterich, T.G., London, B., Clarkson, K., and Dromey, G., Learning and inductive inference, STAN-CS-82-913, Stanford University, also Chapter XIV of *The Handbook of Artificial Intelligence*, Cohen, P.R., and Feigenbaum, E.A. (Ed.), Kaufmann, 1982.

[Du73] Duda, R.O., and Hart, P.E., *Pattern Classification and Scene Analysis*, Wiley, 1973.

[Fu82] Fu, K.S., *Syntactic Pattern Recognition and Applications*, Prentice-Hall, 1982.

[Go83] Good, I.J., *Good Thinking: The Foundations of Probability and Its Applications*, University of Minnesota Press, 1983.

[GK78] Gowda, K.C. and Krishna, G., Disaggregative clustering using the concept of mutual nearest neighborhood, *IEEE Transactions on Systems, Man and Cybernetics, SMC-8*, 12 (1978), 888–894.

[Ha86] Hanson, S.J. and Bower, M., Machine learning, clustering, and polymorphy, this volume.

[Ho86] Horvitz, E., Modular belief updates and the inconsistent use of measures of certainty in artificial intelligence research, this volume.

[Ka72] Kanal, L., and Chandrasekaran B., On linguistic, statistical and mixed models for pattern recognition, in Watanabe, S. (Ed.), *Frontiers of Pattern Recognition*, Academic Press, 1972, 163–192.

[Mi83] Michalski, R.S., Carbonell, J.G., and Mitchell, T.M. (Ed.), *Machine Learning: An Artificial Intelligence Approach,* Tioga, 1983.

[Ne65] Newman, C. and Uhr, L., BOGART: A discovery and induction program for games, *ACM Proc. 20th National Conference,* 1965, 176–185.

[Qu83] Quinlan, J.R., Learning efficient classification procedures and their application to chess end games, in [Mi83], 463–482.

[Re77] Rendell, L.A., A locally optimal solution of the fifteen puzzle produced by an automatic evaluation function generator, Dept of Computer Science CS–77–36, University of Waterloo, 1977.

[Re81] Rendell, L.A., An adaptive plan for state–space problems, Dept of Computer Science CS–81–13, (PhD thesis), University of Waterloo, 1981.

[Re83a] Rendell, L.A., A new basis for state–space learning systems and a successful implementation, *Artificial Intelligence 20,* 4 (1983), 369–392.

[Re83b] Rendell, L.A., Conceptual knowledge acquisition in search, in Bolc, L. (ed.), *Computational Models of Learning,* Springer–Verlag (to appear), also Univ. of Guelph Report CIS 83–15, 1983.

[Re83c] Rendell, L.A., A learning system which accommodates feature interactions, *Proc. Eighth IJCAI,* 1983, 469–472.

[Re85a] Rendell, L.A., Genetic plans and the probabilistic learning system: Synthesis and results, *Proc. International Conference on Genetic Algorithms and their Applications,* July, 1985, 60–73.

[Re85b] Rendell, L.A., Substantial constructive induction using layered information compression: Tractable feature formation in search, *Proc. Ninth IJCAI,* August, 1985, 650–658.

[Re86] Rendell, L.A., A general framework for induction and a study of selective induction, *Machine Learning Journal 1,* 2 (1986, in press).

[Sa63] Samuel, A.L., Some studies in machine learning using the game of checkers, in Feigenbaum, E.A. and Feldman, J. (Ed.), *Computers and Thought,* McGraw–Hill, 1963, 71–105.

[Sa67] Samuel, A.L., Some studies in machine learning using the game of checkers II—recent progress, *IBM J. Res. and Develop. 11* (1967) 601–617.

[Sl68] Slagle, J.R., and Bursky, P., Experiments with a multipurpose, theorem–proving heuristic program, *JACM 15* (1968) 85–99.

[To74] Tou, T.T., and Gonzalez, R.C., *Pattern Recognition Principles,* Addison–Wesley, 1974.

[Wa69] Watanabe, S., *Knowing and Guessing: A Formal and Quantitative Study,* Wiley, 1969.

[Wi86] Wise, B.P. and Henrion, M., A framework for comparing uncertain inference systems to probability, this volume.

[WB68] Wallace, C.S., and Boulton, D.M., An information measure for classification, *The Computer Journal 11,* 2 (1968), 185–194.

ACKNOWLEDGEMENTS

I would like to thank Peter Cheeseman, Chris Matheus, and Raj Seshu for their helpful suggestions.