Induction is a central issue in the philosophy of science. It has practical import for the field of machine learning, which is concerned with discovering good descriptions of concepts. Examples are presented to a machine which must, using background knowledge and rules of inductive inference, induce a theory that explains the data. Induction is an extremely powerful inference technique. It is, however, highly under-constrained, and evaluation of inductive inference systems and methods can be very elusive.

The aim of this paper is to describe and illustrate the principle of complexity-based induction. Informally, this principle judges theories by the amount of information they need to reproduce a given set of positive examples of a concept. The measure strikes a delicate balance between the complexity and generality of a theory.

The paper is structured as follows. In section 1 we give a formal description of induction and generalization. Induction embraces concept learning [16], connectionist learning [14], incremental learning, sequence identification [9], explanation-based generalization [20] and unsupervised classification [17]. Here we restrict attention to concept learning—the results extend naturally to other forms. Section 2 develops complexity-based induction in detail, and gives measures which can be applied directly to logic programs. Two applications of the technique are presented in section 3. Section 4 discusses complexity-based inductive inference. Section 5 outlines the principles of probabilistic induction, and establishes the relationship between pure logical and probabilistic induction.

1 Formalizing induction

Deductive inference generates theorems; inductive inference generates hypotheses. We briefly review the logical foundation of induction, as presented in [11, chapter 7] and [22]. This exposition of induction uses logic programming terms [15]. Let

L_O be the language of observations,

L_B be the language of background knowledge,

L_H be the language of hypotheses.

The background knowledge and hypotheses are often expressed in the same form, and this is assumed henceforth. We shall refer to this language as the *hypothesis space*, and denote it by the symbol $\Omega \equiv L_B \equiv L_H$. We require

that $L_O \subseteq \Omega$, and that statements in L_O are ground. The hypothesis space Ω and the language of observations L_O may be finite or infinite.

Suppose $B \subseteq \Omega$ is background knowledge, and $E \subseteq L_O$ is the set (without duplicates) of observations seen so far. Inductive inference generates a hypothesis $H \subseteq \Omega$ subject to the following requirements:

necessity $B \not\vdash E$,

consistency $B \cup E \not\vdash \neg H$,

explanatory power $B \cup H \vdash E$.

The necessity condition ensures that the background knowledge does not already entail the examples — if it did, there would be no need for an inductive hypothesis. The consistency condition asserts that the negation of the hypothesis is not entailed by the background knowledge and examples — if it were, the hypothesis would be inconsistent with what is already known. The final condition dictates that the background knowledge and hypothesis, taken together, entail the observations.

Given a specification of the three languages described above, and a set of observations $E \subseteq L_O$, there will generally be an indefinite number of inductive hypotheses H that meet the above requirements. The requirements say nothing about which one to prefer. This is the role of a *preference criterion*, an extra-logical rule that prefers one hypothesis to another.

1.1 Concept learning

We now describe concept learning, a basic inductive inference task. Suppose that there is a concept named c. In logic, a *instance* of the concept is represented by the conjunction of literals

$$c(e_1, \ldots, e_k) \wedge B_1 \wedge \ldots \wedge B_n, \tag{1}$$

where the e_i are constants. The B_i are the predicates which are "operational" or "observable". In concept learning, it is implicitly assumed that other predicates "cause" c. For the purposes of concept learning, the instance is therefore translated into the clause

$$c(e_1, \ldots, e_k) \leftarrow B_1 \wedge \ldots \wedge B_n. \tag{2}$$

A concept C is a set of instances — it may be infinite. The $B_1, \ldots, B_n$ are *sufficient* conditions for concept membership [20]. By reversing the implication in (2) they become *necessary* conditions.

A *positive example* is an element of C. A *negative example* or *non-instance* of a concept is a clause

$$\neg c(e_1, \ldots, e_k) \leftarrow B_1 \wedge \ldots \wedge B_n. \tag{3}$$

A non-instance is an element of the set $L_O - C$. Negative examples do not naturally occur: they arise from querying an *oracle*, which can answer the question "is $x \in C$?" for any possible observation x. The ultimate goal of learning is to construct a compact, intensional, sufficient definition of the set C, in finite time, from examples and/or queries of the oracle. Restriction to a finite amount of time and data necessitates using inductive inference to achieve or approximate this goal. An approximation to a concept C is called a *theory*.

Formally, a theory of a concept is a Horn-clause theory [15], called a *logic program*, such that c is the head of at least one clause, that is, there is at least one clause of the form

$$c(X_1, \ldots, X_k) \leftarrow \ldots, \tag{4}$$

where the variables X_i are understood to be universally quantified. The disjunction of the bodies of all such clauses is called a *recognition function* or *classification rule* for the concept. Note that logic programs cannot deduce negative literals, and we assume a mechanism such as PROLOG's negation as failure rule to infer negative information. Induction is very important in all concept learning areas where background knowledge is incomplete or non-existent. An induced theory can augment a machine's background knowledge, and contribute to improved concept recognition.

1.2 Generality and simplicity

Inductive inference is closely related to the notion of generalization. Here we give an informal discussion of theory generality; see [4] for a more detailed description.

A logic program T_1 is *more general than* or *subsumes* a program T_2 if $T_1 \vdash T_2$ and $T_2 \nvdash T_1$. That is, the ground literals entailed by T_1 are a superset of those entailed by T_2. This definition imposes a lattice on logic programs, with the empty clause as the top element, and the empty program as the bottom element. An upper bound of a set of programs is called a *generalization* of that set.

A theory $T \equiv B \cup H$ *explains* a set of examples E if $T \vdash E$. In Michalski's [16] classification, we require that the theory "strongly," not "weakly,"

entail the examples — that is, every example is a strict logical consequence of the theory. Note that the examples are points in the lattice, and any generalization of them is an explanatory theory. In particular, the maximally simple theory

$$\top \equiv c(X_1, \ldots, X_k) \leftarrow \tag{5}$$

and the maximally complex theory

$$\bot \equiv E \tag{6}$$

both explain any set E of positive examples.

There may be more than one generalization of the examples in the lattice, and a preference criterion is needed to choose between them. One pervasive preference criterion is Occam's principle of parsimony. We accept as an axiom:

Occam's principle. *Entia non sunt multiplicanda praeter necessitatum,*

literally, "entities should not be multiplied without necessity". The simplest, least ornate explanatory theory is to be preferred.

Although theory simplicity is an ancient and intuitive notion, Occam's principle must be interpreted carefully to yield a useful preference criterion for machine learning, as the next section will demonstrate.

1.3 Overgeneralization

Suppose that only positive examples are available. Inductive inference machines employing Occam's principle under standard interpretation will not succeed, as they are liable to select the most general theory $\top$. Clearly, the theory is explanatory and maximally simple, but is probably meaningless because it indiscriminately accepts any new observation. We construct some examples of this phenomenon, drawn from well-known concept learning systems.

Mitchell [19] characterizes concept learning as searching for generalizations (propositional theories T) that explain all positive and no negative examples. The *candidate elimination* algorithm retains two sets, G and S, that are, respectively, the most general and most specific theories that explain the examples. To satisfy Occam's principle, select the simplest theory from the set G of maximally general theories. The theory G is initialized to $\top$. If no negative examples are given, $\top$ will be the simplest explanatory theory.

Quinlan's ID3 [26] inductive learning algorithm learns decision trees (propositional theories) from examples. The ID3 algorithm recursively finds an attribute to test, and for every possible outcome of the test, prunes examples from E and creates a new branch in the decision tree. The recursion terminates when all examples are in the same class. If given only positive examples, this termination criterion will immediately succeed with the theory $\top$.

Winston [36] describes a general–purpose procedure for learning descriptions of structured scenes from examples. The theory T is initialized to the first positive example. Subsequent positive and negative examples ("near misses") provoke theory generalization and specialization, respectively. If no negative examples are given, the machine may steadily march towards the generalization $\top$.

Shapiro's Model Inference System [32] learns a concept by initializing T to the most general theory $\top$, and searching through a specialization hierarchy of logic programs, directed by negative examples. Only negative examples can remove $\top$ from its perch.

The MARVIN concept learning system [30] uses background knowledge B to guide search through a generalization hierarchy of first–order theories. In contrast to Shapiro's system, the theory T is initialized to the most specific theory $E \cup B$. Generalization operators are incrementally applied to the theory. Although MARVIN does not use negative examples, at each stage of generalization an oracle must be consulted to confirm that overgeneralization has not occurred.

The concept learning systems described above do not behave gracefully when negative examples are not present. This is because $\top$ is a generalization of any set of positive examples. The insistence on negative examples is, in many domains, quite artificial. They require careful preparation, and learning machines lose a large degree of autonomy. The next section describes and applies an interpretation of Occam's principle that works in learning settings that include neither an oracle nor negative examples. Negative examples may be present; the interpretation has no problem with this.

The abolition of negative examples, however, comes with a price. Fundamental theorems of Gold [13] and Angluin [2] show that most hypothesis spaces are not identifiable in the limit from positive examples. That is, an inductive inference machine may never converge to an intended or "target" theory.

Identification in the limit from positive examples is a very strong property of a hypothesis space. It is, nevertheless, somewhat unrealistic in most

real–world domains where only a small number of examples are available. Valiant's [35] "probably approximately correct" learning framework grants hypotheses a margin of error. A hypothesis space is PAC–learnable if an inductive inference procedure exists which will, given a bounded number of examples, produce a hypothesis that will never accept a non–instance, and will accept *most* instances of the concept. Valiant [35] demonstrates the PAC–learnability of some hypothesis spaces. For example, bounded k–CNF propositional expressions (where each clause is the disjunction of at most k literals) are PAC–learnable from positive examples. Monotone DNF–expressions (where no propositional variable is negated) are PAC–learnable using an oracle. Note the severe restrictions on the hypothesis space in both cases.

Both identification in the limit and PAC–learnability of a hypothesis space Ω state that if there is a theory $T \in \Omega$ that explains all positive instances (most instances in PAC learnability) and no non–instances of the concept, then there exists an inductive inference machine that will, given enough examples, eventually find T. In practice, however, there may not exist such a theory in Ω. In this situation, we are interested in finding the best possible theory in Ω using the available examples. We attribute no intensional descriptions to nature, and are interested in selecting the theory that provides the nearest possible approximation to reality, whatever that may be.

2 Complexity–based induction

Complexity–based induction theory [33] motivates a fresh interpretation of Occam's principle, based on the idea of data compression. The metaphor is one of *communication*: the set or sequence E of (positive) examples must be transmitted across a noiseless channel. Any similarity that is detected among observations can be exploited by giving them a more compact coding. Instances of the concept should have short codes, and non–instances should have no code at all. If examples of a problem or concept cannot be appreciably compressed, it is called *random* [1].

According to complexity–based induction, the best theory for a concept is defined to be the one that minimizes the number of bits required to communicate the examples. This is the essence of the *minimum description length principle* [28, 29]. Complexity–based induction allows the induction of a theory from positive data without separate hypothesis testing.

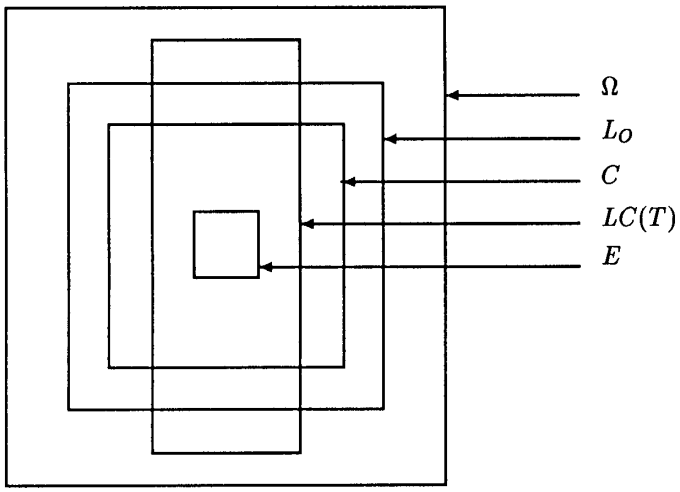

Figure 1: The languages of induction.

The ground logical consequences $LC(T)$ of a theory T are defined as $\{x : T \vdash x, x \text{ is ground}\}$. In logic programming terms, this is the least Herbrand model of the logic program T. The set of logical consequences of a theory can be very large, and for this discussion we are interested in only a subset of them, namely those that are also possible observational statements:

$$Q(T) = L_O \cap LC(T). \tag{7}$$

In other words, $Q(T)$ is the set of observational statements entailed by T. Popper [24] calls $Q(T)$ the *empirical content* of a theory.

Figure 1 shows the relationships between the various languages of induction. The goal of concept learning is to make the set $Q(T)$ identical to the concept set C. The hypothesis space Ω contains all possible theories and all their theorems, including all that can be expressed in the observation language L_O. The set $\Omega - L_O$ contains all well-formed formulae of the hypothesis space which are not observational statements. The set $L_O - C$ contains all non-instances of the concept C; as we have seen, existing inductive inference systems rely heavily on a presentation of this set in the form of oracle queries or explicit negative examples. Note that the logical

consequences $LC(T)$ of a theory T may contain non–observational statements, and the provably true observational statements $Q(T)$ may contain non–instances.

Inductive inference hypothesizes a theory T, and this theory explains some observational statements $Q(T)$. There are three possible relations between the set of examples E and $Q(T)$:

1. $E = Q(T)$. The theory explains all the examples and no more.
2. $E \not\subset Q(T)$. There are examples not explained by the theory.
3. $E \subset Q(T)$. The theory explains all the examples, and more.

In case 1, only the theory T need be transmitted, since the receiver can reconstruct E exactly by deductive inference. As E grows, however, it becomes increasingly unlikely that a simple theory in a hypothesis space explains E and only E. If case 2 holds, we must add the set $E - Q(T)$ — the exceptions to the theory — explicitly to T, thus ensuring that the theory is explanatory. This situation is not depicted in figure 1 because it is assumed that T has already been so augmented. In case 3 the theory is explanatory, but more general than E — that is fine, to an extent. It indicates that induction has taken place. However, further information must be transmitted to convey the set E.

2.1 Coding examples

In case 3, the common situation, to transmit E the explanatory theory T must be sent along with some extra coding information that specializes it to explain E and only E. To transmit E, first transmit T, encode E with respect to T, then transmit that encoding. Denote the code for T by $D(T)$, and the encoding of the examples by $D(E|T)$. If the codes for theories and examples form a *prefix set* (no code is a prefix of any other), upon receiving the catenation $D(T) \cdot D(E|T)$ a receiver can exactly reconstruct E.

The complexity of T is the length (in bits) of the string $D(T)$; denote this by $L(T)$. The fit of T to E is the length (in bits) of $D(E|T)$; denote this by $L(E|T)$. If T does not explain E, $L(E|T)$ is defined to be infinite. Complexity–based induction dictates that the best theory T for the data E minimizes the description length

$$L(T|E) = L(T) + L(E|T). \tag{8}$$

If $L(T|E)$ is greater than or equal to $L(E)$, the code length of the raw examples, useful induction has not taken place. Otherwise, valid generalizations have been made, redundancy has been extracted from E, and the data has been compressed.

The quantity $L(T)$ measures the textual complexity of a theory T, and will be discussed in section 2.2. The quantity $L(E|T)$ measures the complexity of the examples when coded using T. There is some question about what this means in a logical setting. As mentioned above, the string $D(E|T)$ must specialize T to explain E and only E. There are various ways to do this:

1. Identify n observational statements in $Q(T)$ using $\log_2 \left(\begin{array}{c} |Q(T)| \\ n \end{array} \right)$ bits. This can (almost) be achieved using an efficient coding method such as *arithmetic coding* [37].

2. For every example, send information identifying its position in an enumeration of the observational theorems of T.

The first measure is only useful when L_O is a finite set, because it requires the computation of $Q(T)$ (see expression 7). If L_O is countably infinite, measure 2 must be used.

Let us look at the consequences of adopting these interpretations. In the first, the generality of the theory is counterbalanced by the number of extra observational statements — other than the examples — it explains or covers. The theory $\top$ has maximal extra coverage; in fact, $Q(\top) = L_O$, and $L(E|\top) = \log_2 \left(\begin{array}{c} |L_O| \\ n \end{array} \right) = L(E)$. At the other extreme, consider the theory $\bot$. This theory has zero extra coverage, $L(E|\bot) = \log_2 \left(\begin{array}{c} n \\ n \end{array} \right) = 0$, and $Q(\bot) = E$. Neither theory is able to compress the examples. The first cannot because (as discussed in section 2.2) $L(E|\top)$ is just as large as $L(E)$; the second cannot because $L(\bot)$ is just as large as $L(E)$.

The first measure discriminates against overly-general hypotheses, as they will require too much information to specify examples from E. In the second interpretation, one devises a scheme for enumerating $Q(T)$. It is natural to have statements with short proofs appear early in the enumeration. If this is the case, the measure discriminates against hypotheses that require long proofs to explain E. Muggleton [21] gives an application of this method, using the structure of the resolution universe as an enumeration

technique. Feldman [10] also uses a proof-theoretic measure of complexity, called *derivational complexity*, for evaluating inductive inference of context-free grammars. Note the similarity with explanation-based learning, where there is a space/time tradeoff; adding redundant generalizations to a theory may speed up performance for similar future examples, but also may degrade overall performance. This is known as the *utility problem* [18]. Complexity-based induction can offer a principled objective measure for choosing when to save a generalization.

In conclusion, when the examples are coded relative to the theory, the options we identify are (1) to code the identity of the appropriate statements in $Q(T)$, and (2) to code the proofs of the statements in $Q(T)$. The first is a syntactic measure which encodes certain sentences (i.e., the examples) relative to the language in which they are couched (i.e., the language generated by the theory). The latter is more semantic: it encodes how the sentences are generated by the theory in terms of the "proof" which exhibits that they belong to the language.

2.2 Coding theories

We have discussed two methods for coding the consequences of a logical theory, and now turn to the problem of coding the theories themselves. For simplicity, function-free logic programs are assumed.

The easiest way to code a logic program is to transmit it directly in, say, ASCII format. This, however, is extremely inefficient. Like examples, theories contain redundancy and can be compressed; there is redundancy inherent in the grammar of well-formed theories, and there is redundancy introduced by a particular use of the grammar.

Just as it is impossible to find the best theory for a set of examples, it is impossible, in general, to find the best "theory for theories" (see section 4.1 below). However, the efficiency of the theory coding scheme can have an effect on how many examples a complexity-based inductive inference machine requires. The best coding scheme for theories will have the property that

$$L(\bot|E) = L(\top|E) \tag{9}$$

or, equivalently, that $L(\bot) = L(E)$ and $L(\top) = 0$ (recall that $L(E|\bot) = 0$ and $L(E|\top) = L(E)$). In practice, $L(\bot)$ will be at least as large as $L(E)$, due to Shannon's second theorem [31]. A good theory coding scheme reduces this differential.

program	⇒ clause ◇ program	(0.5)
program	⇒ Λ	(0.5)
clause	⇒ head ← body	(1.0)
head	⇒ atom	(1.0)
body	⇒ literal , body	(0.5)
body	⇒ Λ	(0.5)
literal	⇒ atom	(0.5)
literal	⇒ ¬ atom	(0.5)
atom	⇒ (see text)	

Figure 2: A probabilistic grammar for logic programs.

There are numerous valid ways to code logic programs. We use the following simple, but reasonably efficient, scheme. Well–formed logic programs have an unambiguous context–free grammar, shown in figure 2. The numbers on the right are production probabilities. The probability $P(T)$ of a particular theory T with respect to the grammar is the product of the production probabilities used in its derivation. The code length $L(T)$ of the theory is $-\log_2 P(T)$ bits. All that remains for a particular hypothesis space is to specify what atoms can occur. This can be done using the theory's lexicon — the variables, constants, and predicate symbols that appear in the program — along with their arities. If there are v variables and c constants, there are $\frac{(v+c)!}{(v+c-a)!}$ ways to arrange them as arguments for a predicate of arity a. If there are p predicate symbols in the lexicon, an individual predicate symbol can be identified using $\log_2 p$ bits. In general, if there are c constants, p predicate symbols and v variables in the lexicon, an atom formed using a predicate of arity a can be coded in

$$\log_2 \frac{(v+c)!}{(v+c-a)!} + \log_2 p \text{ bits.} \tag{10}$$

Whereas the ground atoms formed from the constants and predicate symbols (the Herbrand universe) are fixed in advance, different theories may need a different number of symbols for variables. To accommodate this, we preface each theory with a code of length $\log_2 v$ bits, identifying the number of variables v. In the case that the theory has no variables, one is added so that the logarithm is defined. The next section shows how this measure is used.

We do not conjecture that our coding method for logic programs is near optimal. One immediate idea for improvement is to base the probability of a literal on its relative frequency of occurrence in the theory. Exploring efficient methods for coding logic programs is a worthwhile area for future research.

3 Examples of complexity–based induction

This section gives a brief description of the coding scheme described above, in two simple concept learning settings. In both examples, L_O is finite, and measure 1 of section 2.1 is used to compute $L(E|T)$. The hypothesis space Ω is finite in the first example, and infinite in the second.

3.1 Ravens

The hypothesis space Ω is the set of logic programs with lexicon

$$\{raven/1, black/1, e_1/0, \ldots, e_n/0\}, \tag{11}$$

where the e_i are constants. In the three theories considered below, there is only one variable X. Therefore, by expression 10, each atom can be coded in $1 + \log_2(n+1)$ bits.

The concept under consideration here is that of "raven–ness". We describe ravens by one property: "black–ness". The observation language L_O is therefore all statements of the form

$$\begin{aligned} &raven(e) \leftarrow black(e), \text{ or} \\ &raven(e) \leftarrow \neg black(e), \end{aligned} \tag{12}$$

where e is a constant.

Consider an example set of n black ravens:

$$E = \{raven(e_1) \leftarrow black(e_1) \diamond \ldots \diamond raven(e_n) \leftarrow black(e_n)\}, \tag{13}$$

(the symbol $\diamond$ is a clause separator). The complexity $L(E)$ of n examples is $\log_2 \left(\begin{array}{c} 2n \\ n \end{array} \right)$. We now consider the complexity–based evaluation of three different explanatory theories for E.

Theory 1

Consider the maximally complex theory $T \equiv E \equiv \bot$. The code length of this theory, according to the grammar of figure 2, is

$$1 + 2n + 2n + 2n(1 + \log_2(n+1)) = 1 + n(6 + 2\log_2(n+1)) \text{ bits.} \quad (14)$$

It is clear that T needs no specialization, since $Q(T) = E$. The total description length is therefore

$$L(T|E) = L(T) + L(E|T) = L(\bot) + 0 = 1 + n(6 + 2\log_2(n+1)) \text{ bits.} \quad (15)$$

Theory 2

Now consider the simplest possible theory $T \equiv \{raven(X) \leftarrow\} \equiv \top$. The complexity $L(T)$ of this theory is $4 + \log_2(n+1)$. It is clear that $Q(T) = L_O$, that is,

$$\begin{aligned} Q(T) \quad = \{ \quad & raven(e_1) \leftarrow black(e_1) \diamond raven(e_1) \leftarrow \neg black(e_1) \diamond \ldots \diamond \\ & raven(e_n) \leftarrow black(e_n) \diamond raven(e_n) \leftarrow \neg black(e_n)\}. \end{aligned} \quad (16)$$

To specialize this theory, we must identify the statements

$$\{raven(e_1) \leftarrow black(e_1) \diamond \ldots \diamond raven(e_n) \leftarrow black(e_n)\} \quad (17)$$

which requires $\left(\begin{array}{c} |L_O| \\ n \end{array} \right) = L(E)$ bits. The description length is therefore

$$L(T|E) = L(\top) + L(E|T) = 4 + \log_2(n+1) + L(E) \text{ bits.} \quad (18)$$

Theory 3

Finally, consider the theory $T \equiv \{raven(X) \leftarrow black(X)\}$. The complexity $L(T)$ of the theory is $7 + 2\log_2(n+1)$. It is clear that $Q(T) = E$, and therefore $L(E|T) = 0$. Thus the description length is

$$L(T|E) = L(T) + L(E|T) = 7 + 2\log_2(n+1) \text{ bits.} \quad (19)$$

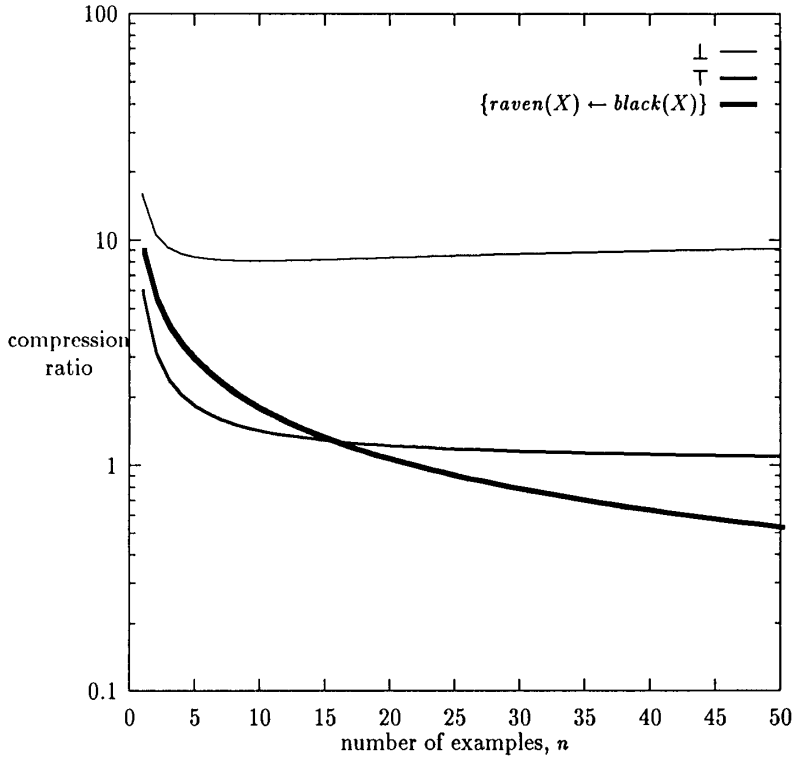

Figure 3: The evaluation of three explanatory theories.

Discussion

The graph of figure 3 plots the compression ratio $L(T|E)/L(E)$ against the number of examples n. Until about 15 black ravens are seen, the machine prefers the theory that admits the possibility of a non-black raven. Note that the theories $\bot$ and $\top$ never achieve any compression of the examples. Given 50 examples, the theory $T \equiv \{raven(X) \leftarrow black(X)\}$ will compress them to about half their original size.

3.2 Networks

The next application of complexity-based induction is learning a general network relation from examples, originally discussed by Quinlan [27]. The concept under consideration is "reachability" in a directed graph. The hypothesis space is the set of logic programs with lexicon

$$\{reach/2, linked/2, 0/0, \ldots, 8/0\}, \tag{20}$$

where $0, \ldots, 8$ are constants.

The background knowledge B contains an extensional definition of the predicate *linked*:

$$\begin{aligned}\{\ & \langle 0,1\rangle \diamond \langle 0,3\rangle \diamond \langle 1,2\rangle \diamond \langle 3,2\rangle \diamond \langle 3,4\rangle \diamond \\ & \langle 4,5\rangle \diamond \langle 4,6\rangle \diamond \langle 6,8\rangle \diamond \langle 7,6\rangle \diamond \langle 7,8\rangle\}.\end{aligned} \tag{21}$$

The notation $\langle x, y\rangle$ is shorthand for the clause $linked(x, y) \leftarrow$, meaning that there is a directed edge between vertices x and y in the network. Below we ignore the complexity $L(B)$ of this background knowledge, as it is constant for all proposed theories.

The observation language L_O is the set of all statements of the form

$$reach(x, y) \leftarrow \tag{22}$$

where x, y are constants, meaning that there is a path between vertices x and y. Thus $|L_O| = 81$.

The example set E is a complete specification of the *reach* relation for a particular network:

$$\begin{aligned}\{\ & \langle 0,1\rangle \diamond \langle 0,2\rangle \diamond \langle 0,3\rangle \diamond \langle 0,4\rangle \diamond \langle 0,5\rangle \diamond \langle 0,6\rangle \diamond \langle 0,8\rangle \diamond \\ & \langle 1,2\rangle \diamond \langle 3,2\rangle \diamond \langle 3,4\rangle \diamond \langle 3,5\rangle \diamond \langle 3,6\rangle \diamond \langle 3,8\rangle \diamond \langle 4,5\rangle \diamond \\ & \langle 4,6\rangle \diamond \langle 4,8\rangle \diamond \langle 6,8\rangle \diamond \langle 7,6\rangle \diamond \langle 7,8\rangle\}.\end{aligned} \tag{23}$$

Theory	T	$L(T)$	$L(E\|T)$	$L(T\|E)$
$T_1 \equiv \top$	$reach(X,Y) \leftarrow$	11.8	60.4	72.2
$T_2 \equiv \bot$	$reach(0,1) \leftarrow \diamond\, reach(0,2) \leftarrow \diamond \ldots \diamond$ $reach(7,6) \leftarrow \diamond\, reach(7,8) \leftarrow$	175.8	0	175.8
T_3	$reach(X,Y) \leftarrow linked(X,Y)$	109.8	0	109.8
T_4	$reach(X,Y) \leftarrow linked(X,Y) \diamond$ $reach(X,Y) \leftarrow linked(X,Z)$	42.6	47.4	90.0
T_5	$reach(X,Y) \leftarrow linked(X,Y) \diamond$ $reach(X,Y) \leftarrow linked(X,Z), linked(Z,Y)$	92.6	0	92.6
T_6	$reach(X,Y) \leftarrow linked(X,Y) \diamond$ $reach(X,Y) \leftarrow linked(X,Z), reach(Z,Y)$	52.6	0	52.6

Table 1: Theories for networks

Thus $L(E) = \log_2 \binom{81}{19} = 60.4$ bits.

Table 1 presents six theories, with their code lengths, the code lengths of the examples, and the final complexity-based evaluation $L(T|E)$. The Appendix gives a detailed derivation of the table.

The intuitively most satisfying theory T_6 is preferred by our complexity-based induction measure. Its closest competitor is the theory $\top$. It is very interesting to note that the measure will continue to prefer T_6 even as the number of available positive examples is decreased from 19 to 15, where $L(T_1|E) = 11.8 + \log_2 \binom{81}{15} = 64.7$ bits and $L(T_6|E) = 52.6 + \log_2 \binom{19}{15} = 64.5$ bits. It is unclear how Quinlan's FOIL system [27] would perform if given 15 randomly-chosen positive examples, since it assumes that every possible observation that is not a positive example is a non-instance.

The theories in table 1 are presented in the order in which FOIL constructs them (with the exception of theory 1, which is not considered). The search space has some interesting properties. It has a sub-optimal minimum at $\top$, and a smaller one at T_4. An inductive inference machine employing this complexity-based measure must be on guard against such minima.

Finally, we strongly suspect that, for this concept learning example, the theory T_6 minimizes the description length, although we are not sure how to give a non-constructive proof of this.

```
T ← ⊤
H ← first(Ω)
while L(H) < L(E)
        if E ⊆ Q(H) and L(H|E) < L(T|E) then T ← H
        Ω ← rest(Ω)
        H ← first(Ω)
return T
```

Figure 4: Induction by enumeration.

4 Inductive inference

Inductive inference can be viewed as a search over a hypothesis space for explanatory theories, evaluated by a preference criterion. The enumeration algorithm shown in figure 4 finds the best theory in a decidable, ordered hypothesis space Ω, according to a complexity–based measure.

The algorithm works as follows. T keeps track of the best theory so far; initially it is the most general theory $\top$. The `while` block retrieves theories from the ordered set Ω. The `if` statement saves the retrieved theory as the best so far if it is explanatory and gives a description length less than the current best theory. The enumeration continues until the complexity of the retrieved theory exceeds the complexity of the examples; there is no point in considering any theory more complex than the examples themselves.

It should be pointed out that the search space may have some structure. The hypothesis space may have a subsumption relation that can be exploited, as is the case with logic programs; if a theory does not explain E, no more specific theory will either. That is, if $T \not\vdash E$ and $T \vdash T'$, then T' need not be considered as a candidate. In addition, if the first complexity measure of section 2.1 is used, any theory that has a simpler equivalent theory in the hypothesis space need not be considered, due to the ordering imposed on Ω. That is, if $Q(T) = Q(T')$ and $L(T') > L(T)$, then T' need not be considered as a candidate.

If uninterrupted, the algorithm terminates with the best theory in Ω, or $\top$ if no theory in Ω explains E. It will terminate so long as Ω is decidable, since there cannot be an infinite number of theories with complexity less than $L(E)$.

Note that the enumeration algorithm is distinctly non–incremental. If

the set E changes, the routine must be performed again. An incremental learning operator maps a current theory and a single example to a new theory, and does not require that E be retained. Some examples of incremental inductive inference methods are the candidate elimination algorithm [19], incremental concept formation [12], the perceptron learning procedure [14], and types of decision tree induction [34]. Incremental inductive inference machines use hill–climbing search, and are unable to minimize a complexity–based measure. However, the complexity–based induction principle can be used to evaluate two or more proposed theories, provided that some of E has been retained.

While the enumeration algorithm cannot be feasibly applied to most hypothesis spaces, it is certainly feasible to use a complexity–based evaluation measure to compare a handful of proposed theories. Our view is that candidate theory generation (inductive inference) is a separate process guided by heuristics, and evaluated by complexity–based measures. Since a complexity–based measure is purely objective, induction can be viewed as a combinatorial minimization problem. As such, optimization techniques such as simulated annealing [25] may be applicable.

4.1 Randomness and complexity

An attempt to minimize the description length $L(T|E)$ requires the specification of a parameterized hypothesis space Ω, ordered by increasing theory complexity. For example, Ω may be the set of logic programs over a Herbrand universe, DNF formulae over k propositional variables, context–free grammars over a specified set of terminals and non–terminals, and so on. Minimization requires that Ω be decidable: there must exist an algorithm for deciding, given an arbitrary $T \in \Omega$ and observation x, if $T \vdash x$. Due to the halting problem, if Ω permits one to define the notion of a Turing machine, it is not decidable.

Because the set Ω of all partial recursive functions is not decidable, the proposition "T minimizes the description length" is only partially decidable for Ω. If it is false, it can be proven so by finding a theory $T' \in \Omega$ such that $L(T'|E) < L(T|E)$. It cannot, however, be proven true.

If Ω is the set of all partial recursive functions, and T minimizes the description length, then $D(T) \cdot D(E|T)$ is a random string [5], and the quantity $L(T|E)$ is called the *algorithmic entropy* [6] of the set E. The algorithmic entropy of E is uncomputable, and a string cannot be proven random. It is impossible, in general, to know whether the best theory for a concept has

been obtained. Computational complexity of theory enumeration aside, this shows that one must place restrictions on Ω, and hope that some theory therein is an adequate representation of the concept. There are no "universal" inductive inference machines that can guarantee inference of the best theory, no matter how much time and data they are given!

5 Probabilistic induction

An expression of the form (8) is known as the *minimum description length principle* [28, 29]. While work in that area has been confined to a form of probabilistic propositional logic, we have in mind a form of probabilistic logic program [11]. This section establishes the relationship between pure logical induction and probabilistic induction. Many examples of (propositional) probabilistic induction can be found in the literature [23].

5.1 Coding probabilistic theories and examples

In section 1 no restriction was placed on how facts were presented to an inductive inference system. In this section, we consider probabilistic concepts; the concept C has a probability distribution. Whereas logical induction could assume that E was a set, for probabilistic induction it is essential that repetition be preserved and reproduced; E is a set with duplicates.

A probabilistic theory includes a probability function P that attaches probabilities to observational statements, obeying the usual conditions

$$\sum_{x \in Q(T)} P(x) = 1, \text{ and } P(x) \geq 0. \tag{24}$$

Recall expression 8, which defined the best theory T as the one which minimized the sum

$$L(T|E) = L(T) + L(E|T). \tag{25}$$

This expression has a very strong Bayesian flavour. Bayesian inference [7] prescribes maximizing the posterior probability $P(T|E)$ for a theory T given examples E, where

$$P(T|E) = \frac{P(T)P(E|T)}{P(E)}. \tag{26}$$

Since E is held constant, it suffices to maximize the numerator of the expression, or equivalently, to minimize

$$-\log_2(P(T)) - \log_2(P(E|T)). \tag{27}$$

If we let the prior distribution on theories be $P(T) = 2^{-L(T)}$, then (27) is of the form

$$L(T) + L(E|T). \tag{28}$$

In probabilistic induction, $L(T)$ will also include the cost of encoding the probabilities of each clause in the theory. This exposition of probabilistic induction is entirely consistent with the first logical measure outlined earlier, which was tantamount to assuming a uniform distribution over C and $Q(T)$. The machine has little to lose by attempting to reconstruct a probability distribution, real or imagined. On the other hand, standard logical theories view all theorems as equally likely, and cannot make "plausible" inferences. They can fail dramatically when the data include noise [7]. They are inadequate for modelling probabilistic concepts.

5.2 Entropy and complexity

Information theory is concerned with efficient communication of data across a channel. It would thus be surprising if there was not a strong relationship with complexity-based induction. This section briefly describes that relationship.

The entropy $H(X)$ of a random variable X is defined as the minimal expected number of bits required to specify an element $x \in X$, where

$$H(X) = -\sum_{x \in X} P(x) \log_2 P(x). \tag{29}$$

The entropy $H(C)$ of a concept C thus gives a lower bound on the expected complexity of an example. In general, the following inequality will hold

$$H(C) \leq \frac{\mathrm{E}[L(T|E)]}{n}. \tag{30}$$

This is due to Shannon's second theorem; it is, on average, impossible to code the examples in fewer bits than the entropy of the concept.

It can therefore be very useful to obtain an estimate of the entropy $H(C)$ of a concept C. It is a figure that an inductive inference machine can strive to achieve. The entropy of a concept can be estimated by putting a human expert in a guessing or gambling situation [8], thereby eliciting probabilities for instances of the concept.

For example, it is known that the entropy of the English language is about 1.1 bits. (The concept is that of "next letter in a sequence", or

statements of the form $nl(X, Y)$ where X is a sequential context and Y is a letter). However, the best current text compression techniques do not achieve this figure. That is, the right hand side of expression 30 is about 2.2 bits [3]. This indicates that there is much room for improvement of induction techniques in this concept learning domain.

6 Conclusion

Complexity–based induction is an objective approach to evaluating induced theories, and is based on a very natural and intuitive principle. It requires that theory complexity and fit to examples be balanced. It reduces the information storage for an inductive inference machine, but not to the extent that theory performance degenerates. It asks for theories of just the right size.

This paper has demonstrated that complexity–based induction, which has for the most part been applied to propositional probabilistic theories and concepts, can be given a natural interpretation with respect to logical induction.

Both logical and probabilistic induction can benefit from this type of research. Logic programs have a clear semantics, and can represent a much richer space of concepts than propositional logic. Inductive inference of logic programs is clean and precise, relying on a well–defined model of generalization. On the other hand, research in machine learning of logical concepts can benefit from the large body of research on minimal–length encoding and Bayesian learning.

References

[1] Y. S. Abu-Mostafa. Complexity of random problems. In Y.S. Abu-Mostafa, editor, *Complexity in Information Theory*. Springer–Verlag, 1988.

[2] D. Angluin. Inductive inference of formal languages from positive data. *Information and control*, 45:117–135, 1978.

[3] T. C. Bell, J. G. Cleary, and I. H. Witten. *Text Compression*. Prentice Hall, 1990.

[4] W. Buntine. Generalized subsumption and its application to induction and redundancy. *Artificial Intelligence*, 36:149–176, 1988.

[5] G. J. Chaitin. Randomness and mathematical proof. *Scientific American*, 232(5):47–52, 1975.

[6] G. J. Chaitin. Algorithmic entropy of sets. *Computers & Mathematics With Applications*, 2:233–245, 1976.

[7] P. Cheeseman. On finding the most probable model. In J. Shrager and P. Langley, editors, *Computational models of scientific discovery and theory formation*, chapter 3. Morgan Kaufmann, 1990.

[8] T. M. Cover and R. C. King. A convergent gambling estimate of the entropy of english. *IEEE Trans Information Theory*, IT–24(4):413–421, 1978.

[9] T. G. Dietterich and R. S. Michalski. Learning to predict sequences. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume II. Morgan Kaufmann, 1986.

[10] J. Feldman. Some decidability results on grammatical inference and complexity. *Information and Control*, 20:244–262, 1972.

[11] M. R. Genesereth and N. J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, 1988.

[12] J.H. Gennari, P. Langley, and D. Fisher. Models of incremental concept formation. *Artificial Intelligence*, 40:11–61, 1989.

[13] E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[14] G. Hinton. Connectionist learning procedures. In Y. Kodratoff and R. Michalski, editors, *Machine Learning Volume III*, pages 555–610. Morgan Kaufmann, 1990.

[15] J.W. Lloyd. *Foundations of logic programming*. Springer–Verlag, 1987.

[16] R. S. Michalski. A theory and methodology of inductive learning. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning*, pages 83–134. Tioga, 1983.

[17] R. S. Michalski and R.E. Stepp. Learning from observation: conceptual clustering. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning*, pages 331–363. Tioga, 1983.

[18] S. Minton. Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, 42:363–392, 1990.

[19] T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

[20] T.M. Mitchell, R.M. Keller, and S.T. Kedar-Cabelli. Explanation-based generalization: a unifying view. *Machine Learning*, 1:47–80, 1986.

[21] S. Muggleton. A strategy for constructing new predicates in first order logic. In *Proc EWSL 88*, pages 123–130, 1988.

[22] S. Muggleton. Inductive logic programming. In *First Conference on Algorithmic Learning Theory*, 1990.

[23] E. Pednault, editor. *Theory and Application of Minimal-Length Encoding: AAAI Spring Symposium Series.* Stanford University, March 1990.

[24] K.R. Popper. *The Logic of Scientific Discovery.* Hutchinson & Co. Ltd., 1959.

[25] W. H. Press et al. *Numerical Recipes: The art of scientific computing.* Cambridge University Press, 1986.

[26] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[27] J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, August 1990.

[28] J. Rissanen. Modelling by shortest data description. *Automatica*, 14:465–471, 1978.

[29] J. Rissanen. Minimum description length principle. In S. Kotz and N.L. Johnson, editors, *Encyclopedia of Statistical Sciences*, pages 523–527. Wiley, 1985.

[30] C. Sammut and R. B. Banerji. Learning concepts by asking questions. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume II, pages 167–191. Morgan Kaufmann, 1986.

[31] C. E. Shannon and W. Weaver. *The Mathematical Theory of Communication.* University of Illinois Press, 1949.

[32] E.Y. Shapiro. *Algorithmic program debugging.* The MIT Press, 1983.

[33] R. J. Solomonoff. Complexity–based induction systems: Comparisons and convergence theorems. *IEEE Trans. Information Theory*, IT–24(4):422–432, 1978.

[34] P.E. Utgoff. Incremental learning of decision trees. *Machine Learning*, 4(2):161–186, November 1989.

[35] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11), November 1984.

[36] P. H. Winston. Learning structural descriptions from examples. In P.H. Winston, editor, *The Psychology of Computer Vision.* McGraw-Hill, 1975.

[37] I. H. Witten, R. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, June 1987.

A Detailed derivation of network code lengths

Theory 1

$$T_1 = \top = \{reach(X,Y) \leftarrow\} \tag{31}$$

Each atom has a code length of $\log_2 \binom{11}{2} + \log_2 2! + \log_2 2 = 7.8$ bits. $Q(T) = L_O$. $L(T) = \log_2 2 + 1 + 1 + 1 + 7.8 = 11.8$. $L(E|T) = \binom{|L_O|}{19} = L(E) = 60.4$. $L(T|E) = L(T) + L(E|T) = 11.8 + 60.4 = 72.2$.

Theory 2

$$\begin{aligned} T_2 = \bot = E = \ \{ \ & reach(0,1) \leftarrow \diamond\, reach(0,2) \leftarrow \diamond \ldots \diamond \\ & reach(7,6) \leftarrow \diamond\, reach(7,8) \leftarrow\} \end{aligned} \tag{32}$$

There are no variables in this program. Each atom has a code length of $\log_2 \binom{9}{2} + \log_2 2! + \log_2 2 = 7.2$ bits. $Q(T) = E$. $L(T) = 1 + 19 + 19 + 19(7.2) = 175.8$. $L(E|T) = 0$. $L(T|E) = L(T) + L(E|T) = 175.8$.

Theory 3

$$T_3 = \{reach(X,Y) \leftarrow linked(X,Y)\} \tag{33}$$

Note that this theory is not explanatory. To make it so, it must be augmented with the clauses $E - Q(T) =$

$$\begin{aligned} \{ \ & reach(0,2) \leftarrow \diamond\, reach(0,4) \leftarrow \diamond\, reach(0,5) \leftarrow \diamond \\ & reach(0,6) \leftarrow \diamond\, reach(0,8) \leftarrow \diamond\, reach(3,5) \leftarrow \diamond \\ & reach(3,6) \leftarrow \diamond\, reach(3,8) \leftarrow \diamond\, reach(4,8) \leftarrow\} \end{aligned} \tag{34}$$

Each atom has a code length of $\log_2 \binom{11}{2} + \log_2 2! + \log_2 2 = 7.8$ bits. $Q(T) = E$. $L(T) = \log_2 2 + 1 + 10 + 10 + 2 + 11(7.8) = 109.8$ bits. $L(E|T) = 0$. $L(T|E) = L(T) + L(E|T) = 109.8$.

Theory 4

$$T_4 = \{ \quad reach(X,Y) \leftarrow linked(X,Y) \diamond \qquad (35)$$
$$reach(X,Y) \leftarrow linked(X,Z)\}$$

Each atom can be coded in $\log_2 \binom{12}{2} + \log_2 2! + \log_2 2 = 8.0$ bits. $L(T) = \log_2 3{+}1{+}2{+}2{+}2{+}2{+}4(8.0) = 42.6$. $|Q(T)| = 54$. $L(E|T) = \log_2 \binom{54}{19} = 47.4$. $L(T|E) = L(T) + L(E|T) = 90$.

Theory 5

$$T_5 = \{ \quad reach(X,Y) \leftarrow linked(X,Y) \diamond \qquad (36)$$
$$reach(X,Y) \leftarrow linked(X,Z), linked(Z,Y)\}$$

This theory is not quite explanatory. To make it so, it must be augmented with the clauses $E - Q(T) =$

$$\{reach(0,5) \leftarrow \diamond\ reach(0,6) \leftarrow \diamond\ reach(0,8) \leftarrow \diamond\ reach(3,8) \leftarrow\} \qquad (37)$$

$Q(T) = E$. $L(T) = \log_2 3 + 1 + 6 + 6 + 6 + 9(8.0) = 92.6$. $L(T|E) = L(T) + L(E|T) = 92.6$.

Theory 6

Finally, consider

$$T_6 = \{ \quad reach(X,Y) \leftarrow linked(X,Y) \diamond \qquad (38)$$
$$reach(X,Y) \leftarrow linked(X,Z), reach(Z,Y)\}$$

$Q(T) = E$. $L(T) = \log_2 3{+}5{+}6{+}5(8.0) = 52.6$. $L(T|E) = L(T){+}L(E|T) = 52.6$.