Compression, Significance and Accuracy

Stephen Muggleton,
Ashwin Srinivasan,
Michael Bain

The Turing Institute,
36 North Hanover Street,
Glasgow G1 2AD,
UK.

Abstract

Inductive Logic Programming (ILP) involves learning relational concepts from examples and background knowledge. To date all ILP learning systems make use of tests inherited from propositional and decision tree learning for evaluating the significance of hypotheses. None of these significance tests take account of the relevance or utility of the background knowledge. In this paper we describe a method, called HP-compression, of evaluating the significance of a hypothesis based on the degree to which it allows compression of the observed data with respect to the background knowledge. This can be measured by comparing the lengths of the input and output tapes of a reference Turing machine which will generate the examples from the hypothesis and a set of derivational proofs. The model extends an earlier approach of Muggleton by allowing for noise. The truth values of noisy instances are switched by making use of correction codes. The utility of compression as a significance measure is evaluated empirically in three independent domains. In particular, the results show that the existence of positive compression distinguishes a larger number of significant clauses than other significance tests The method is also shown to reliably distinguish artificially introduced noise as incompressible data.

1 INTRODUCTION

1.1 COMPRESSION AND BACKGROUND KNOWLEDGE

Although background knowledge is assumed important for learning, it is not entirely obvious how to quantify its relevance. The following example illustrates an information-theoretic approach to this problem. A learner is presented with the points in Figure 1 as examples of a 2-dimensional concept.

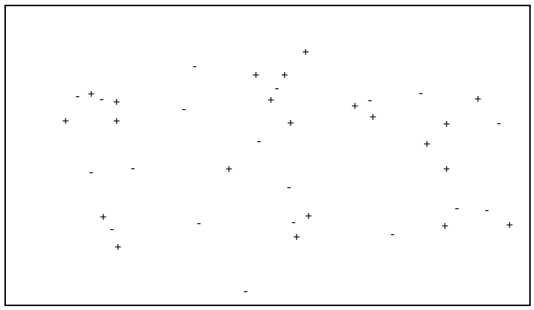

Figure 1: Positive and Negative Examples of a 2 dimensional concept

In the absence of background knowledge, the concept appears somewhat arbitrary. If there are 1000 possible X-coordinate values and 500 Y-coordinate values, describing each example by XY pairs requires $log_2 1000 + log_2 500$ bits. Figure 2 shows that the missing relevant background for this concept (which we might call *port(X,Y)*) was the global coastline. Given this information, if the learner's hypothesis is

port(X,Y) $\leftarrow$ coastal_town(X,Y)

each positive example can be rederived from the set of coastal towns.

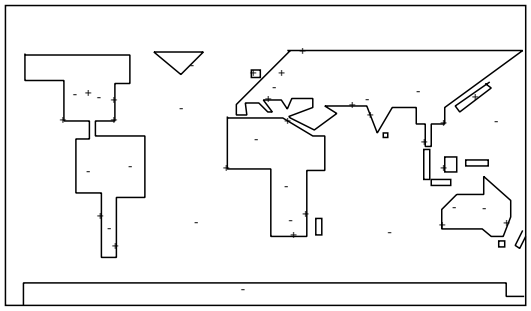

Figure 2: Learning the concept with background knowledge

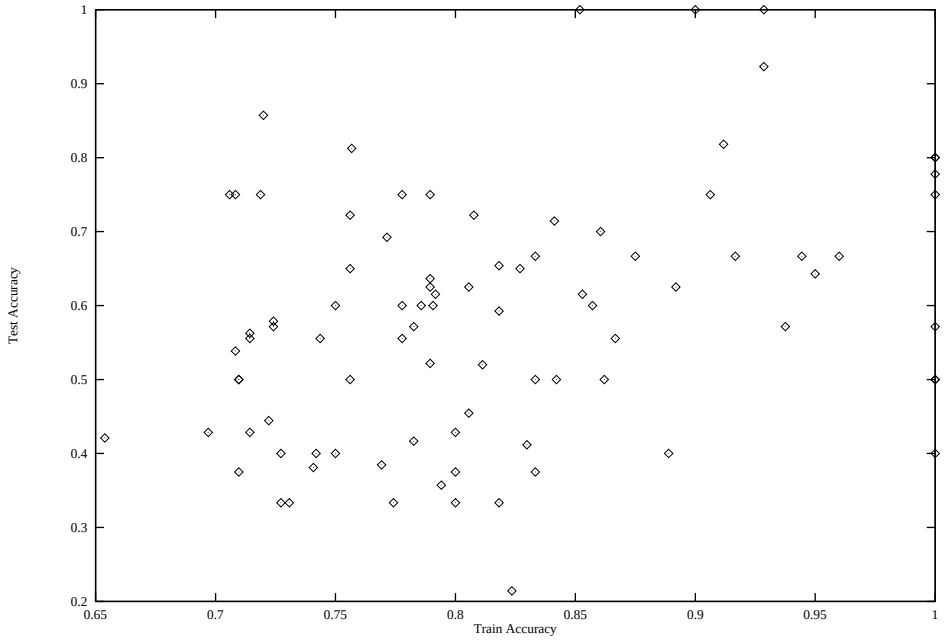

Figure 3: Training and Test set accuracies of clauses predicting protein structure

If only 100 such coastal towns were recorded, then the rederivation is done from a relatively sparse set which requires only $log_2 100$ bits to describe. Appropriate background knowledge and hypotheses can thus produce a considerable compression of the data. Note that the learner's hypothesis though allowing compression, will not be completely correct (not all coastal towns are ports). Yet, there appears to be something significant about this hypothesis. In this paper we describe a model that exploits this approach: the significance of a hypothesis is estimated from the data compression it produces.

1.2 SIGNIFICANCE AND ACCURACY

Research on classification using induced decision trees or rule-sets has found that performance on training data is unlikely to be replicated on unseen (test) data (for example, see [Clark and Niblett, 1989]). Consider the difficult problem of predicting protein secondary structure from amino acid sequence data [Muggleton et al., 1992]. Figure 3 shows the training and test set accuracies of clauses predicting the position of α-helices. Each point in this figure represents a first-order clause constructed by the learning program Golem [Muggleton and Feng, 1990], with its training and test accuracies plotted on X and Y axes respectively. [1] The lack of any correlation (rank correlation = 0.3) between the training and test set

[1] Note that in this domain the accuracy of a rule which predicted all positions to be part of an α-helix is 0.5.

accuracies raises the question: how significant is the training set accuracy of a hypothesis?

Significance measures have been used for some time [Clark and Niblett, 1989] by zero-order induction algorithms with arbitrary settings for the confidence parameter. Current first-order algorithms either assume noise-free data (as in [Hume and Sammut, 1991], [Rouveirol, 1991] or [Bain, 1991]), require arbitrary parameters to be set [Muggleton and Feng, 1990, Quinlan, 1990] or use the significance measures used by zero-order algorithms [Dzeroski, 1991]. The first two approaches are clearly inadequate, while the third offers no direction on how the level of confidence is to be decided. Further, a key difference between first and zero order algorithms is the use of background knowledge. A measure based on the latter does not take into account the extent to which such knowledge is used.

In this paper we propose a uniform, parameter-free compression measure that can be used to answer these questions adequately within the framework of learning first-order theories. It is an extension of a Turing machine compression model described by [Muggleton, 1988] with roots in the formal theory of algorithmic complexity. The utility of the measure is illustrated with three very different problems: protein secondary structure prediction [Muggleton et al., 1992], the drug design problem of modelling structure-activity relationships [King et al., 1992] and detecting illegality in positions of the KRK chess endgame [Muggleton et al., 1989].

2 HYPOTHESIS–PROOF (HP) COMPRESSION

The compression model used in this paper is related to a theory of algorithmic information developed independently by [Solomonoff, 1964], [Kolmogorov, 1965] and [Chaitin, 1987]. The basic result of the theory is that the complexity of any string s is taken to be the length (in bits) of the shortest Universal Turing machine program required to generate s. The length of this program is termed the Kolmogorov complexity of s. Solomonoff applied this approach to the problem of theory confirmation. In his model observational data appears on the output tape of a Universal Turing machine. Theories (programs) explaining this data are placed on the corresponding input tape (Figure 4). Inductive inference is then a

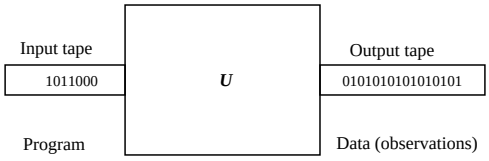

Figure 4: Universal Turing machine model of theory and data

matter of finding the theory which generates the observational output using the shortest input tape. This model has two desirable features. Firstly, it provides a

clear, representation-independent formulation of the Occam's razor principle. It is always possible to count the number of bits in a particular Universal Turing machine program. Compare this to the situation where one is trying to decide the relative simplicity of grammars. The grammars could be expressed as production rules or state transition diagrams. In either case the relative simplicity is not obvious.

Muggleton [Muggleton, 1988] was the first to describe a Turing machine compression model for the problem of learning Horn clause programs. Unlike Figure 4, he uses a reference Turing machine. The input tape of this machine has two distinct parts: a Horn clause theory and a proof specification. The latter specifies how the examples on the output tape are to be derived using the theory (Figure 5).

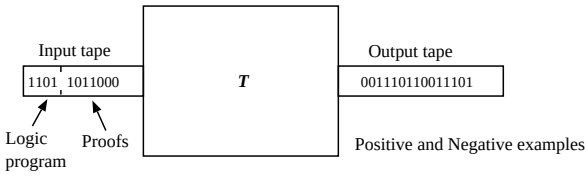

Figure 5: A Turing machine model for learning logic programs

The theory is compressive if the length of the input tape (in bits) is shorter than that of the output tape. The use of a reference machine (as opposed to a universal one) is motivated by demonstrating that the probability of obtaining a compressive hypothesis by chance decreases exponentially with the amount of compression for any machine. We reproduce the proof of this result here

Theorem 1 *Let Σ_n be the set of all binary strings of length n, T_r be an arbitrarily chosen reference Turing machine and the k-bit-compressible strings of length n, $K_{n,k}$, be defined as $\{y : y \in \Sigma_n, x \in \Sigma_{n-k}, T_r(x) = y\}$. The set $K_{n,k}$ has at most 2^{n-k}elements.*

Proof Since Turing machines are deterministic T_r either induces a partial one-to-one or many-to-one mapping from the elements of Σ_{n-k} to the elements of $K_{n,k}$. Thus $|K_{n,k}| \leq |\Sigma_{n-k}| = 2^{n-k}$. $\square$

Corollary 2 *The probability of a binary string generated by tossing an unbiased coin being compressible by k bits using any Turing machine T_r as a decoding mechanism is at most 2^{-k}.*

The approach can be seen as incorporating the MDL principle [Rissanen, 1978, Gao and M.Li, 1989, Quinlan and Rivest, 1989] by identifying the encoding of the data relative to a theory with the notion of proof. By encoding proofs, the model incorporates aspects of time-complexity in the same units (bits) as

the program description. This provides a natural bias towards learning efficient programs.

In this paper we view the reference Turing machine model as providing a uniform framework for answering the questions posed earlier. However, in its original formulation Muggleton leaves issues of efficient coding and noisy data unexplored. To this end, we describe a general coding scheme for hypotheses and proofs intended as the input tape for a machine capable of interpreting logic programs. Theories on the input tape may be incorrect, allowing for noise in the data. Details of the coding scheme and its implementation for this "HP–compression" model are described in Appendix A. The following sections elaborate on the consequences of using this model. Although the model is not dependent on any particular learning algorithm, the results reported here are a result of incorporating it within Golem [Muggleton and Feng, 1990].

3 COMPRESSION AND PROOF ENCODING

A fundamental requirement of the model is that the input tape contains not only the hypothesis, but also information on how to derive the examples that it covers. The latter is achieved by encoding the proof of the examples using the hypothesis. Thus "simplicity" refers not just to the syntactic description of a hypothesis, but also to how well it explains the examples covered. In fact, as the coverage of the hypothesis increases, the proof encoding dominates the input tape (the syntactic description is a once-off cost) and to a good approximation compression per example decreases as proof encoding per example increases.

For each example covered by the hypothesis on the input tape, let us define its *proof complexity* to be the sum of the log of the choice-points involved in deriving the example using SLDNF resolution. Thus, for an example with proof complexity P, the input tape contains a P bit encoding of the choice-points in the derivation. The proof complexity of the hypothesis is defined here as the average proof complexity of the examples it covers. We believe the emphasis on encoding proofs to be a unique feature of this model and one that gives it two distinct advantages when used within the framework of a first-order learning system

1. The definition of proof complexity is procedural: hypotheses that are more efficient to execute have lower proof complexity. Given two similar hypotheses (in terms of length, coverage and accuracy), the one with lower proof complexity produces more compression. This provides a bias towards learning efficient clauses by incorporating notions of time complexity.

2. The proof encoding quantifies the relevance of the background knowledge. A lower average proof complexity implies fewer arbitrary choices were made from the background knowledge in order to derive the examples on the output tape. Compression is higher for such hypotheses.

We now qualify the relationship between compression and proof encoding for the implementation described in Appendix A. For each example covered by a hypothesis, there are two parts of the proof encoding that are of interest: the choice-point encoding (representing the proof complexity of the example) and a proof tag (which indicates if the example is true or false positive) The complete length of the proof encoding on the input tape is approximately

$$L_{proof} = n \times P_{av} + t_p \times log\frac{n}{t_p} + f_p \times log\frac{n}{f_p}$$

where n is the number of examples covered, t_p of which are true positives and f_p are false positives. P_{av} represents the average proof complexity of the examples (by definition, the proof complexity of the hypothesis). Defining hypothesis accuracy A to be $t_p/(t_p + f_p)$, it is a matter of simple manipulation to show

$$L_{proof} = n \times (P_{av} + log\frac{1}{A^A \times (1-A)^{(1-A)}})$$

The proof complexity will dominate the logarithmic term in the previous equation (which decreases from 1 to 0 as accuracy of the hypothesis increases from 0.5 to 1). In general, the proof and syntactic complexity of a hypothesis are balanced against its accuracy and coverage. For example, consider hypotheses with similar proof complexities and accuracies. The proof cost and therefore, compression per example is approximately constant. Consequently, hypotheses with greater coverage produce more compression (that is, for very similar theories, confidence is higher in theories that cover more). Finally, it is worth noting that although the accuracy of a hypothesis affects the proof encoding and hence its compression, it does not necessarily follow that hypotheses with large compression will be highly accurate. This is consistent with compression being a measure of significance and not a predictor of accuracy. Accuracy only plays a role when comparing hypotheses of similar length, proof complexity and coverage. In such situations, the hypothesis with the higher accuracy is taken to be more significant (that is, has higher compression).

4 COMPRESSION AND SIGNIFICANCE

We illustrate the utility of using compression as a measure of confidence in clauses learned for three different problems

1. Prediction of protein secondary structure. The prediction of protein secondary structure from primary sequence is an important unsolved problem in molecular biology. Recently it has been shown that the use of relational learning algorithms (see [Muggleton et al., 1992] and [King and Sternberg, 1990]) can lead to improved performance.

2. Modelling drug structure-activity relationships. The design of a pharmaceutical drug often requires an understanding of the relationship between its structure and chemical activity. Rules learned to model this relationship have been recently been shown to perform better than existing numerical methods [King et al., 1992].

3. Learning rules of illegality for the KRK chess end-game. Despite its simplicity, the KRK problem remains the test-bed for ILP techniques. We evaluate the compression measure with different levels of noise (introduced artificially into the class values).

Although there is usually a difference between training and test accuracy for clauses, the utility of a confidence measure can be evaluated on the basis of the fluctuation of this difference for clauses reported to be significant by the measure. With this notion of "stability" of clause-set, a compression-based choice is compared against one based on

1. Training set accuracy.

2. Training set coverage.

3. Likelihood ratio. This is a standard test of significance that has been used by zero-order learning algorithms [Clark and Niblett, 1989] and more recently within the first-order framework [Dzeroski, 1991]. If a clause covers n examples, t_p of which are true positives then the value of the statistic is calculated as follows. Let p^+ and p^- be the prior probabilities of the positive and negative examples (usually estimated from their relative frequencies in the training set), and $q^+ = t_p/n$, $q^- = 1 - q^+$. Then

$$LikelihoodRatio = 2n(q^+ log(\frac{q^+}{p^+}) + q^- log(\frac{q^-}{p^-}))$$

   This is distributed approximately as χ^2 with 1 degree of freedom.

For each domain, the standard deviation of the difference in training and test accuracies (s) is normalised to the one obtained for compressive clauses. Each pair of numbers in Figure 6 represents the value at which this standard deviation is achieved for the different measures and the number of clauses detected as significant respectively. The values of s obtained for compressive clauses is shown in Figure 7.

Figure 6 highlights some key points

1. Compression usually distinguishes a larger number of significant clauses than the other measures.

Domain	Number of clauses	Compression	Accuracy	Cover	Likelihood Ratio
Proteins	89	>0; 16	_	>2.2; 15	>20.8; 12
Drugs	107	>0; 103	>0.9; 97	>2.1; 79	>9.7; 103
KRK(5%)	19	>0; 19	>0.5; 19	>0; 19	>0; 19
KRK(10%)	36	>0; 31	_	>0.9; 30	>9; 24
KRK(20%)	40	>0; 33	>0.7; 17	>1.1; 31	>6.0; 30

Figure 6: A comparison of significance measures

Domain	SD of (Train - Test) accuracy
Proteins	0.09
Drugs	0.05
KRK(5%)	0.08
KRK(10%)	0.13
KRK(20%)	0.15

Figure 7: Standard deviation of compressive clauses

2. A compression-based choice is easy: all compressive clauses are thought to be significant. On the other hand, the threshold for the other measures is not obvious. For example, recall that the value of the Likelihood ratio is distributed approximately as χ^2 with 1 degree of freedom. Normal practice is to set the threshold at about 6.6 (99% confidence).

3. In most cases, training set coverage appears to be as good a measure as the more sophisticated likelihood ratio.

4. Training set accuracy appears to be the worst measure of significance: in some cases (indicated by "_"), none of the clauses thought significant by this measure were as reliable as those detected by compression.

Finally, it is worth noting that the compression obtained using the model has a definite meaning. According to Corollary 2 the probability of obtaining k bits compression by chance is at most 2^{-k}. Clearly, this statement only makes sense when k is positive (hence the choice of compressive clauses only). As k increases, we become increasingly confident that the hypothesis has not detected a chance regularity. Given two hypotheses of similar accuracy, the one that produced higher compression is deemed more reliable.

5 COMPRESSION AND NOISE

The original motivation of algorithmic information theory was to provide a definition of randomness. Following this approach, *noise* in our model is defined as any data that cannot be compressed (any regularity detected amongst them is taken to happen by chance). This provides a method of "seeing through" the noise in the domain. This feature is used in [Srinivasan et al., 1992] to guide the progressive correction of first-order theories within a non-monotonic framework termed Closed-World-Specialisation (CWS: see [Bain and Muggleton, 1991, Bain, 1991]). In encoding terms, each correction performed by the CWS algorithm increases the theory encoding on the input tape and decreases the proof encoding. In the model in Figure 5, a net decrease in the length of the input tape occurs when the correction succeeds in identifying some pattern in the errors (that is, the errors are not noise).

Figure 8 (from [Srinivasan et al., 1992]) tabulates the features of the most compressive theory to learn the concept *legal*/6 in the KRK end game. Specialisation commences with the top-level over-general clause *legal(A,B,C,D,E,F)*. The results are for different levels of noise introduced to the class value in a training set of 10,000 examples. A simple noise model introduced in [Angluin and Laird, 1988] is used (for example, 10% noise implies 10% of the examples were selected at random and their class values flipped). In the figure, the error on noisy data refers to the error on the training set. This represents the data left uncompressed by the specialisation process. The next column represents the accuracy of the theory on error-free test data of the same size, which shows that the fraction of the data left uncompressed at the training stage was mostly noise. It is worth noting that the cases in which the White King is in between the White Rook and Black King accounts for less than 0.4% of the examples. While this is picked up by the compression model in the noise-free data, it is lost at higher noise levels.

% Noise introduced	Compression (bits)	% Error on noisy data	Accuracy on error-free data
0	8547.65	0	100.00
5	5917.39	5.39	99.61
10	4390.40	10.31	99.61
20	2072.20	20.86	99.61

Figure 8: Performance of theories with maximum compression on KRK end-game

Within the non-monotonic framework adopted, all theories are complete. The proof encoding scheme described in Appendix A ensures that all variables are instantiated before a new negated predicate is introduced as part of the specialisation. Except for the noise-free case, the theories selected are the same (the noise-free case has the additional clauses accounting for the exceptions to the

illegal concept). All theories tabulated therefore have the same proof-complexity, coverage and length (the noise-free theory is slightly longer). As mentioned at the end of Section 3, these are the circumstances in which increased accuracy results in greater compression.

Besides being an interesting quantity in itself, the amount of noise plays a role in determining the accuracy of a hypothesis on unseen data. In a noisy domain, it is unlikely that this accuracy would be as high as that from which the hypothesis was derived (the training set). Consider a training set classified by an oracle (that is, a noise-free data set). Let p be the accuracy of the hypothesis on this data set (this is the "real accuracy"). Let q be the probability that the observed classification of examples in the training set agrees with that of the oracle. For noisy domains $q \leq 1$. Then, on unseen data from the same source the accuracy of the hypothesis can at best be $pq + (1-p)(1-q) = 1 - p - q + 2pq$.

The problem with this analysis is that in general, both p and q are unknown. One way out is to use compression as a "noise meter". Under the assumption that clauses with very high compression are performing as well as they possibly can in the domain (that is, they exactly avoid fitting any noise), p can at best be equal to 1. On the other hand, it should always be possible to achieve a real accuracy of q. Estimating $1 - q$ using the noise meter technique, it is possible to obtain a rough estimate of the test accuracy of a theory chosen on the basis on the basis of high compression. For example, for the case with 20% errors in Figure 8, q is estimated at 0.79. The accuracy of the theory on unseen data from the same source can be estimated as lying between 0.67 (for $p = q$) and 0.79 ($p = 1$). On the other hand, on noise-free data, its accuracy is in the interval 0.79 ($p = q$) and 1 ($p = 1$).

6 CONCLUSIONS

In this paper we have developed a general encoding scheme for deciding the significance of first-order hypotheses by refining the approach found in [Muggleton, 1988]. The requirement to encode both hypotheses and proofs results in some unique advantages

1. The resulting compression measure appears to be the first significance measure that accounts for the relevance and utility of background knowledge. This issue has been avoided to date by relational learning systems.

2. The measure appears to reliably distinguish noisy data by finding them to be incompressible with the background knowledge.

3. The encoding incorporates aspects of time complexity in the same units (bits) as the program description, thus addressing some of the issues concerning the time-space tradeoff for encoded knowledge (this is discussed in detail by Michie in [Michie, 1977]).

While the empirical results in this paper show compression to be a reliable significance measure and noise meter, the simplicity of contending hypotheses does not give any direct indication of how well they will do on unseen data. This question was dealt with first by Gold [Gold, 1967] and more recently within the PAC (Probably-Approximately-Correct) framework [Valiant, 1984]. The Gold and PAC frameworks describe the conditions under which a class of concepts can be said to be learnable. In the PAC framework it is explicitly assumed that the distribution of examples in the training and test sets are the same. [Blumer et al., 1986], [Board and Pitt, 1989] and [Li and Vitanyi, 1989] have in various ways shown that a class of concepts is PAC-learnable if and only if it can be guaranteed that a learning algorithm is able to find a hypothesis which is smaller than the data. It remains to be shown that our concept of hypothesis size (that is, hypothesis and proofs) is equivalent to that adopted in these theoretical results.

Acknowledgements

The authors would like to thank Donald Michie and the ILP group at the Turing Institute for their helpful discussions and advice. This work was carried out at the Turing Institute and was supported by the Esprit Basic Research Action project ECOLES, the IED's Temporal Databases and Planning project and the SERC Rule-Base Systems Project. Stephen Muggleton is supported by an SERC post-doctoral fellowship.

A HP–COMPRESSION: A GENERAL CODING SCHEME FOR LOGICAL HYPOTHESES AND PROOFS

A.1 SETTING

We follow the logical setting for inductive learning used by current Inductive Logic Programming systems [Muggleton, 1991]

$$B \wedge H \vdash E^+$$

where B is background knowledge, H is an hypothesis consisting of one or more clauses and E^+ is a set of positive examples not entailed by B. In addition, if E^- is a set of negated formulae representing counter-examples then we can guard against over-generalisation by ensuring that $B \wedge E^- \wedge H$ is *satisfiable* i.e. self-consistent.

In the Turing machine model adopted here, a machine T has the following input-output behaviour

$$T(\mathrm{I}(B \wedge H, \mathrm{Proof}(E^+, E^-))) = \mathrm{O}(B \wedge E^+ \wedge E^-)$$

where I, O and $Proof$ are input, output and proof encodings for T. The k-bit compression achieved by the input tape is then

$$k = |\mathrm{O}(B \wedge E^+ \wedge E^-)| - |\mathrm{I}(B \wedge H, \mathrm{Proof}(E^+, E^-))|$$

A.2 INPUT TAPE ENCODING

The basis of the compression model is that efficient (ideally optimal) encodings are found for input and output tapes of a machine. The components of the input tape are shown in Figure 9.

Size-of-B	No-of-clauses	Symbol-frequencies	B	Hypothesis	Proofs

Figure 9: Sections of the input tape

The reference machine interprets the input tape as follows:

- The size of the background knowledge (number of atoms and/or clauses in B) allows the machine to distinguish between it and the hypothesis constructed. The number of clauses is used for two purposes. It states how many clauses to expect on the input tape and it is also used to construct a special clause separator symbol. Although the background knowledge can consist of clauses, it is common practice with current ILP systems to represent it by a ground model [Muggleton and Feng, 1990, Quinlan, 1990]. The need to specify symbol frequencies is elaborated shortly.
- Each example on the output tape is generated by its proof encoding. The machine acts as a logic program interpreter. For each example, the proof encoding specifies the clauses in the hypothesis and background knowledge that are used to derive the example.
- The machine outputs the atoms and/or clauses in B without interpreting them onto the output tape.

A.2.1 THEORY ENCODING

A logical theory can be viewed as a sequence of symbols. A near optimal choice for encoding these symbols involves the use of prefix codes. We assume a vocabulary S of symbols where each symbol $s \in S$ appears with relative frequency p_s. A prefix code is a function

$$Prefix : S \rightarrow \{0,1\}^*$$

which has the property that no code is a prefix of any other code. This property ensures that codes are self-delimiting. Information Theory [Shannon and Weaver, 1963] tells us that the optimal code length for symbol s is $-log_2 p_s$ bits. Huffman coding

[Gallager, 1968] is a prefix coding which achieves approximately this code length for each symbol.

In order for the machine to "understand" the encoding, it is necessary to place a header on the input tape which defines the frequencies of the different symbols used. This can then be used to construct a code-book for the message on the input tape. The components of this header are shown in Figure 10.

PSym-count	Zero	FSym-count	Zero	Var-count	Zero	PSym-arity	FSym-arity

Figure 10: Sections of symbol description header

Predicate, function and variable symbols have different codes. A prefix table, such as that of the predicate symbols, consists of the individual symbol counts in order of their appearance. This sequence of natural numbers is sufficient for a unique reconstruction of the codes used in the theory. The clause separator symbol (constructed using the clause count) is treated as though it were a predicate symbol. The arities for predicate and function symbols are also number sequences whose orders correspond to those in the prefix tables. Clearly the clause separator "predicate" symbol has no arity. "Zero" is defined to be the encoding of the natural number 0 and acts as a separator for different sections of the header. Separators are not necessary to delimit the arities since their number is determined by the predicate and function symbol counts. In order to avoid infinite regress we must find a universal coding for the natural numbers that appear in the header. Natural numbers can be encoded using prefix codes given an appropriate prior distribution. Rissanen [Rissanen, 1982] shows that an optimal distribution can be defined for which the code length $L(n)$ is bounded as follows

$$log_2 n < L(n) < log_2 n + r(n)$$

where $r(n)/log_2 n \to 0$ and $r(n) \to \infty$ as $n \to \infty$.

We assume that logical theories are expressed as a set of Prolog clauses. The following grammar gives the syntax of our encoding of theories.

Theory ::= { Clause }$^{\text{No of clauses}}$
Clause ::= Atom Clause | Stop
Atom ::= PredSym [Negated] {Term}$^{\text{Arity(PredSym)}}$
Term ::= ["0"] FuncSym {Term}$^{\text{Arity(FuncSym)}}$ |
["1"] VarSym
Negated::= "0" | "1"

Although for longer theories we would expect that the header information would be considerably shorter than the statement of the theory, prefix coding may not be very efficient for small theories. Clearly, each symbol type (predicate, function, variable) can be coded differently with bits at the front of the input tape indicating the type of coding adopted for each symbol. This will change the contents of the header. Within

our implementation, we can select the most efficient amongst three different coding schemes for a symbol: universal natural number code, a fixed-length code or a prefix code (listed in order of increasing header information). For each scheme, we use the non-integral code length as an optimal estimate. The assumption here is that this value can be reached when sufficiently long messages are encoded.

A.2.2 PROOF ENCODING

The reference machine T takes the theory and a proof encoding and generates the examples. Derivational proofs are represented as sequences of choices to be taken by a Prolog interpreter. For example, consider deriving normal(1581) using the clauses

```
normal(Year) :- year(Year), not(leap4(Year)).
leap4(Year) :- mod(Year,4,0).
```

Assume a set of 1581 ground atoms for year/1 and corresponding ones for mod/3 are part of the background knowledge (of course, these would have to be encoded as described earlier). In deriving the example, the interpreter has to first choose which clause of that predicate to execute. In our case there is only one such clause, and thus no choice. The first atom in the body of the chosen clause, year(Year), can be matched against any one of the set of unit clauses running from year(1) to year(1581). Specifying the choice for the example requires $\lceil log_2 1581 \rceil$ bits on the input tape. This choice completely determines the rest of the proof and thus no more information need be provided on the tape. Thus the complete derivational proof for normal(1581) can be represented in 11 bits. However we want to be able to encode a sequence of proofs; one for each example on the output tape. This can be achieved by preceding the series of proofs by an encoding of the number of examples.

This encoding of proofs is sufficient for examples which are derivable from range-restricted (generative) theories. However, it has to be extended to accomodate for the following

1. For non-generative clauses, substitutions have to be provided for variables that do not occur in the body of the clause (since these will never be bound by any choice specification). The function codes for any substitutions needed appear after the choice specifications.

2. Incorrect theories [Shapiro, 1983] can still be used for compressing data to a certain degree. The theory in the example earlier is an example of a useful, though incorrect, theory.

To address the second issue we distinguish two categories of results obtained from the theory

1. True Positives. In this case the truth-value of the derived fact agrees with the intended interpretation.

2. False Positives. The truth-value of the derived fact is the opposite of the intended interpretation. These are errors of commission made by the theory.

Each choice-point encoding is preceded by a prefix code indicating its category (this acts as a correction flag). The prefix codes for the categories are constructed using a pair of numbers indicating the counts in each category. These numbers are coded using the universal coding scheme and precede the proof encoding on the input tape. Clearly, the total number of examples no longer have to be specified.

A.3 OUTPUT TAPE ENCODING

The output tape encoding is almost the same as that of a logical theory (see Section A.2). The difference is that examples are simply encoded as atoms rather than clauses. Examples appear in the output tape in the order of observation (that is, no reordering is permitted).

References

[Angluin and Laird, 1988] Angluin, D. and Laird, P. (1988). Learning from noisy examples. *Machine Learning*, 2(4):343–370.

[Bain, 1991] Bain, M. (1991). Experiments in non-monotonic learning. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 380–384, San Mateo, CA. Morgan Kaufmann.

[Bain and Muggleton, 1991] Bain, M. and Muggleton, S. (1991). Non-monotonic learning. In Michie, D., editor, *Machine Intelligence 12*. Oxford University Press.

[Blumer et al., 1986] Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. (1986). Classifying learnable geometric concepts with the Vapnik-Chervonenkis dimension. In *Proceedings of the 18th ACM Symposium on Theory of Computing*, pages 273–282.

[Board and Pitt, 1989] Board, R. and Pitt, L. (1989). On the necessity of occam algorithms. Uiucdcs-r-89-1544, University of Illinois at Urbana-Champaign.

[Chaitin, 1987] Chaitin, G. (1987). *Information, Randomness and Incompleteness - Papers on Algorithmic Information Theory*. World Scientific Press, Singapore.

[Clark and Niblett, 1989] Clark, P. and Niblett, T. (1989). The CN2 algorithm. *Machine Learning*, 3(4):261–283.

[Dzeroski, 1991] Dzeroski, S. (1991). *Handling Noise in Inductive Logic Programming*. University of Ljubljana, (M.Sc. Thesis), Ljubljana.

[Gallager, 1968] Gallager, R. G. (1968). *Information theory and Reliable Communication*. Wiley, New York.

[Gao and M.Li, 1989] Gao, Q. and M.Li (1989). An application of minimum description length principle to online recognition of handprinted numerals. In *IJCAI-89*, Detroit, MI. Kaufmann.

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

[Hume and Sammut, 1991] Hume, D. and Sammut, C. (1991). Using inverse resolution to learn relations from experiments. In Birnbaum, L. and Collins, G., editors, *Proceedings of the Eighth International Workshop on Machine Learning*, pages 412–416, San Mateo. Morgan Kaufmann.

[King et al., 1992] King, R., Muggleton, S., and Sternberg, M. (1992). Drug design by machine learning: The use of inductive logic programming to model the structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proc. of the National Academy of Sciences*, 89(23):11322–11326.

[King and Sternberg, 1990] King, R. and Sternberg, M. (1990). A machine learning approach for the prediction of protein secondary structure. *Journal of Molecular Biology*, 216:441–457.

[Kolmogorov, 1965] Kolmogorov, A. (1965). Three approaches to the quantitative definition of information. *Prob. Inf. Trans.*, 1:1–7.

[Li and Vitanyi, 1989] Li, M. and Vitanyi, P. (1989). Inductive reasoning and Kolmogorov complexity. In *Proceedings of the Fourth Annual IEEE Structure in Complexity Theory Conference*, pages 165–185.

[Michie, 1977] Michie, D. (1977). A theory of advice. In Elcock, E. and Michie, D., editors, *Machine Intelligence 8*, pages 151–168. Horwood.

[Muggleton, 1988] Muggleton, S. (1988). A strategy for constructing new predicates in first order logic. In *Proceedings of the Third European Working Session on Learning*, pages 123–130. Pitman.

[Muggleton, 1991] Muggleton, S. (1991). Inductive logic programming. *New Generation Computing*, 8(4):295–318.

[Muggleton et al., 1989] Muggleton, S., Bain, M., Hayes-Michie, J., and Michie, D. (1989). An experimental comparison of human and machine learning formalisms. In *Proceedings of the Sixth International Workshop on Machine Learning*. Kaufmann.

[Muggleton and Feng, 1990] Muggleton, S. and Feng, C. (1990). Efficient induction of logic programs. In *Proceedings of the First Conference on Algorithmic Learning Theory*, Tokyo. Ohmsha.

[Muggleton et al., 1992] Muggleton, S., King, R., and Sternberg, M. (1992). Predicting protein secondary structure using inductive logic programming. *Protein Engineering*, 5:647–657.

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

[Quinlan and Rivest, 1989] Quinlan, J. and Rivest, R. (1989). Inferring decision trees using the Minimum Description Length principle. *Information and Computation*, 80:227–248.

[Rissanen, 1978] Rissanen, J. (1978). Modeling by Shortest Data Description. *Automatica*, 14:465–471.

[Rissanen, 1982] Rissanen, J. (1982). A universal prior for integers and estimation by Minimum Description Length. *Annals of Statistics*, 11:416–431.

[Rouveirol, 1991] Rouveirol, C. (1991). Itou: Induction of first-order theories. In *First International Workshop on Inductive Logic Programming*, Porto, Portugal.

[Shannon and Weaver, 1963] Shannon, C. and Weaver, W. (1963). *The Mathematical Theory of Communication*. University of Illinois Press, Urbana.

[Shapiro, 1983] Shapiro, E. (1983). *Algorithmic program debugging*. MIT Press.

[Solomonoff, 1964] Solomonoff, R. (1964). A formal theory of inductive inference. *Information and Control*, 7:376–388.

[Srinivasan et al., 1992] Srinivasan, A., Muggleton, S., and Bain, M. (1992). Distinguishing noise from exceptions in non-monotonic learning. In Muggleton, S. and Furukawa, K., editors, *Second International Inductive Logic Programming Workshop*. Institute for New Generation Computer Technology.

[Valiant, 1984] Valiant, L. (1984). A theory of the learnable. *Communications of the ACM*, 27:1134–1142.