Exploiting Symmetry Properties in the Evaluation of Inductive Learning Algorithms: An Empirical Domain-Independent Comparative Study

Hussein Almuallim
Department of Computer Science
Oregon State University
Corvallis, OR 97331-3202

91-30-09

Exploiting Symmetry Properties in the Evaluation of Inductive Learning Algorithms: An Empirical Domain-Independent Comparative Study [1]

Hussein Almuallim

303 Dearborn Hall
Department of Computer Science
Oregon State University
Corvallis, OR 97331-3202
Phone: (503) 737-5568
Email: almualh@cs.orst.edu

Abstract

Although numerous Boolean concept learning algorithms have been introduced in the literature, little is known about what categories of concepts are actually learned satisfactorily by most of these algorithms. Conventional comparison studies, which test various algorithms in some chosen domain, do not provide such information, since their conclusions are limited to the domain considered. A more general way to evaluate a learning algorithm is to test it on all the possible concepts defined on a given number of Boolean features. However, this immediately leads to unaffordable computational costs, since we need to consider as many as 2^{2^n} concepts, when the number of features is n. In [D89], experiments of this type were reported for the case of three features, while the cases of four or more features were concluded to be infeasible.

This paper directly builds on the work of [D89]. We introduce two techniques that significantly cut the computational costs of the desired experiments and enable us to perform experiments over the space of concepts defined on up to five variables. The first technique is to exploit the fact that inductive learning algorithms are generally insensitive to permuting and/or complementing the features of the domain. We give a method for eliminating redundancy in the experiments by computing a set of *representative concepts* that suffices to characterize the behavior of a given algorithm over the space of all concepts. The second technique is to resort to statistical approximation to avoid running algorithms on all the possible samples of a concept. We show that testing a feasibly small number of samples suffices to obtain results with a high level of confidence.

Applying these techniques, we report experimental results analogous to those of [D89] on some decision tree building algorithms over five Boolean features. The results we present are rather surprising and demonstrate that there is still much to be learned about the algorithms we tested.

The paper also discusses the possibility of enhancing the above techniques to work for the cases of six or more Boolean features.

[1] Thanks to Tom Dietterich for his many helpful comments on this work, to Bella Bose and Paul Cull for suggesting related references, to Robert Rowley for insightful discussions on permutation groups and to Prasad Tadepalli and Ghulum Bakiri for valuable comments on earlier version of this paper. This work was supported in part by NSF grant number IRI-86-57316 (Presidential Young Investigator Award).

1 Introduction

The conventional way of evaluating and comparing inductive learning algorithms has been to empirically examine the generalization performance of these algorithms in some selected domains. Studies of this type necessarily lead to conclusions that apply only to the domains in which the algorithms are tested. A more desirable direction, which we follow in this paper, is to perform wider-scope evaluation studies that are aimed at the characterization of the type of concepts on which a given learning algorithm does or does not work well. Not enough attention has been paid to this topic, and the behavior of many well-known algorithms that lack formal analysis (such as ID3 [Q86], FRINGE [PH89], Backpropagation [RHW86], etc.), is still not well understood.

In his paper [D89], Dietterich proposed to take the number of concepts *reliably* learned by a given learning algorithm as the assessment figure for the algorithm. In this work, we call this quantity the "coverage" of the algorithm, and follow Dietterich's notion of Frequently Approximately Correct (FAC) learning as the definition of reliable learning of a concept. Dietterich proved an upper bound on the maximum coverage that no learning algorithm can exceed when given a fixed number of examples. He also measured the coverage of some learning algorithms in the space of concepts defined over 3 Boolean features. The number of concepts in this case is only 256, and thus, it was computationally possible to measure the coverage by running exhaustive experiments on all these concepts. However, for 4 or more Boolean features, running such experiments was concluded to be infeasible due to the explosive number of concepts and of samples per concept needed to be tested.

In this work, we introduce some techniques that lead to substantial cuts in the computational costs of coverage measurement. The first technique is to exploit the fact that most of the known learning algorithms are symmetric with respect to permuting and/or negating the domain features. Cost reduction is realized as a result of the elimination of redundant runs caused by these symmetry properties. The second technique is to resort to statistical approximation to avoid running the algorithms on a prohibitively large number of samples per concept. Both of these techniques are algorithm-independent in the sense that they are applicable to almost any Boolean concept learning algorithm. The details of this approach are given in Sections 3 and 4.

By employing the above two techniques, we were able to measure the coverage of some decision-tree building algorithms on concepts defined on up to 5 features. The experimental results we obtained are reported in Section 5. Running the same experiments for 6 or more features was not feasible using the above techniques alone. To extend this work, we propose in Section 6 some ideas that we believe may help in performing experiments with larger number of features.

2 Notation

Consider the space of Boolean concepts (or equivalently, functions) defined on n Boolean features (variables). A truth assignment, $\vec{x} \in \{0,1\}^n$, to these n variables is called an *instance* and, for a concept c, a pair $\langle \vec{x}, c(\vec{x}) \rangle$ is called an *example* of c. When $c(\vec{x}) = 1$ (=0) the example is called *positive* (*negative*), respectively.

We represent a concept as a 2^n-bit vector, which is just the right-most (output) column of the truth table of the concept, and call this the *bit-vector representation* of the concept. That is, for $0 \leq i < 2^n$, the i-th bit is assigned the value $c(\vec{x})$ where $\vec{x}$ is the i-th instance (i.e., the instance obtained by using the binary representation of i to assign the values to the n variables, in the obvious way). For convenience, we use the the hexadecimal representation to express the bit-vector representation of concepts. For example, `aaaa00ff` denotes the function $\neg x_1 \neg x_5 \vee x_1 x_2$.

The integer interpretation of the bit-vector representation of a concept is called the *integer value* of the concept. The *weight* of a concept is the number of positive examples in the concept (or, equivalently, the number of 1's in the bit-vector representation of the concept).

We assume the uniform distribution on the space of instances and say that a concept c_1 is ϵ-close to a concept c_2, if the hamming distance between the bit-vector representation of the two concepts divided by 2^n (the total length of the bit-vectors) is at most ϵ.

In this work, examples of the target concept are drawn without replacement. An m-sample of a concept c is a set of m distinct examples of c. m is called the size of the sample. Clearly, there are $\binom{2^n}{m}$ different m-samples for any concept. A learning algorithm is an algorithm that maps samples to concepts. Following

[D89], we say that an algorithm *Frequently Approximately Correctly (FAC) learns* a concept c, with respect to m, ϵ and δ, if a fraction of at least $1 - \delta$ of the m-samples of c are mapped by the algorithm to concepts that are ϵ-close to c.

For a given learning algorithm L, testing the FAC learnability of a concept c given m, ϵ and δ, means determining whether or not L FAC learns c with respect to these parameters. Finally, the *coverage* of a learning algorithm, is the number of concepts that are FAC learned by the algorithm, for given values of m, ϵ and δ.

3 Exploiting Symmetry Properties

One source of the prohibitive computational cost in the empirical measurement of coverage is that the space of concepts to be considered grows very rapidly as n becomes larger. When $n = 5$, for example, the number of concepts in the space is 2^{2^5} which is more than 4 billion concepts. Measuring the coverage by directly testing the FAC learnability of all these concepts is clearly not practical.

In this section, we argue that it is not necessary to test algorithms against all possible concepts in the space. We will first introduce two kinds of symmetry properties generally satisfied by algorithms that learn Boolean concepts, and then describe how these properties can be exploited to reduce the cost of coverage measurement.

3.1 Symmetry Properties

Most of the known inductive learning algorithms (e.g. ID3 [Q86], FRINGE [PH89], Backpropagation [RHW86]) are insensitive to permuting and/or negating the features of the domain. For given values of ϵ and δ and a given sample size, if we know that an algorithm FAC learns a concept represented by a Boolean function say $f(x_1, x_2, \cdots, x_i, \cdots, x_j, \cdots, x_n)$ then this implies that the same algorithm also learns the concepts represented by $f(x_1, x_2, \cdots, x_j, \cdots, x_i, \cdots, x_n)$, $f(x_1, x_2, \cdots, \neg x_i, \cdots, x_j, \cdots, x_n)$ and so on for all the functions obtained by permuting and/or negating the features in f.

More formally, let OP_n be the set of all operators that permute and/or negate the set $\{x_1, x_2, \ldots, x_n\}$ in all possible ways. For example, $op_1 \in OP_5$ might be the operator that maps $x_1x_2x_3x_4x_5$ to $x_2x_1x_3x_4\neg x_5$ (i.e., exchanges x_1 and x_2 and negates x_5). Clearly, operators in OP_n can also be applied to Boolean concepts by carrying out the transformations of the operator on the variables of the corresponding Boolean function. In our example, op_1(`aaaa00ff`) gives `550055ff`. We can then argue that for any symmetric algorithm A, the fact that A FAC learns a concept c implies that A also learns all the concepts $op(c)$, $op \in OP_n$.

This says that the operators in OP_n partition the space of concepts into equivalence classes in terms of FAC learnability. For every pair of concepts, c_1 and c_2, if there exist some $op_i, op_j \in OP_n$ such that $op_i(c_1) = c_2$ and $op_j(c_2) = c_1$, then c_1 and c_2 are in the same equivalence class, and thus c_1 is FAC learned if and only if c_2 is. On the other hand, if there exist no operators that map c_1 to c_2 (or the opposite), then c_1 and c_2 are in two different classes.

For algorithms that satisfy the above symmetry properties, testing an algorithm against one concept in an equivalence class suffices to determine the FAC learnability of all the concepts in the class. Therefore, to obtain a list of the concepts that are learned by a given algorithm in a given setting, we need to test the algorithm only on a list of *representative* concepts, one from each equivalence class. The next subsection gives one method of computing a set of such representative concepts.

3.2 Finding a Set of Representative Concepts

The procedure we used to obtain a set of representative concepts is shown in Figure 1. The choice of which concept is to represent an equivalence class can, of course, be done arbitrarily. In this work, we let the concept with the maximum integer value (over all the concepts in the same class), be the representative of the class. Therefore, to check whether a given concept c is the representative of its class (step 4.1), we simply apply all the operators in OP_n and check whether there exists no operator op such that $op(c)$ gives a concept with a higher integer value than c. [2]

[2]This is, definitely, impractical for large values of n, since there are as many as $n!2^n$ operators in OP_n. An alternative implementation of the test in step 3.1, is the the use of *invariants* of a concept [H65]. These are a set

Algorithm: Find-Representatives
Input: n

1. Let REP be empty
2. Let Q contain the concept $\overbrace{000\ldots00}^{2^n}$
3. Let c be the concept at the head of Q
4. While $weight(c) \leq 2^{n-1}$, repeat:
   4.1. If for all $op \in OP$,
        the integer value of $c >$ the integer value of $op(c)$ then:
      4.1.1. Add c to REP
      4.1.2. Add $Successors(c)$ to the tail of Q
   4.2. Remove c from Q
   4.3. Let c be the concept at the head of Q
5. Return REP

Figure 1: Algorithm for finding a set of representative concepts

Step 4.1.2 calls the function *Successors(c)*. This function computes and returns the successors of the concept c as follows: Let j be the position of the right-most bit in c with the value 1 (letting $j = -1$ if $c = 000\ldots00$). Then the successors of c are all the concepts generated by setting the k-th bit of c to 1, for $k = j+1, j+2, \ldots, 2^n - 1$. For example, the successors of `abcdf0f8` are `abcdf0fc`, `abcdf0fa` and `abcdf0f9`.

The procedure given in Figure 1 maintains a queue Q which initially contains only the concept $00\ldots00$. In each iteration, the concept at the head of Q is removed form the queue. The concept is added to the set REP and its successors are attached to the end of Q, only if the concept was found to be a representative. When the algorithm terminates, the set REP will contain all the representatives of the equivalence classes of weights 0 to 2^{n-1}. The rest of the equivalence classes (of weights $2^{n-1}+1$ to 2^n) can be easily obtained by complementing the representatives of weight $2^{n-1}-1$ down to 0, as found by the given procedure.

The proof of correctness for this procedure is omitted here due to the limited space. The number of equivalence classes and the number of concepts per equivalence class can be computed using Pólya's theory of counting which is discussed in detail in [H65].

3.3 Usefulness and Limitations of the Technique

How much reduction in the computational cost is obtained by considering only the representative concepts? For n features, there are $n!$ permutations and 2^n ways of negating. Therefore, this will result in an asymptotic reduction factor of $O(n!2^n)$. The following table shows specific numbers for n up to 6.

n	2^{2^n}	# of equivalence classes
3	256	22
4	65,536	402
5	$> 4.2 \times 10^9$	1,228,158
6	$> 1.8 \times 10^{19}$	400,507,806,843,728

These numbers show that the technique we give contributes to a significant cut in the number of concepts to be considered. Indeed, without employing this technique, experiments for $n = 5$ would not be feasible.

Unfortunately, it is clear that performing experiments for $n \geq 6$ is far from feasible, unless further cost reduction techniques are also incorporated. Some ideas that may help for larger number of features are discussed in Section 6 of the paper.

of quantities whose values are common for all concepts in a given equivalence class, but different from all other classes; i.e. a set of quantities that identify the equivalence class to which a given concept belongs. However, in the experiments carried out in this work, n is up to 5, and thus, the simple method of applying all the operators in OP_n was found adequate to perform the experiments.

4 Statistical Determination of FAC Learnability of a Concept

Another source of computational expense in the empirical measurement of coverage is the fact that for each concept, there exists a very large number of samples on which algorithms must be executed. For example, when $n = 5$, there are $\binom{32}{16}$, i.e. more than 600 million distinct samples of size 16, per concept.

To reduce the cost for testing the FAC learnability of a concept, we resort to statistical approximation and estimate the coverage by giving upper and lower bound values such that the true value of coverage lies, with high confidence, between these two bounds.

To determine the FAC learnability of a concept by a given learning algorithm, we run the algorithm on a sufficiently large number of randomly-chosen samples of the desired size. We then compute the ratio $\tilde{\delta}$ of those samples on which the algorithm does not return an ϵ-close hypothesis. If $\tilde{\delta}$ is less than δ by a certain threshold, say θ, then we conclude that the concept is FAC learned. Conversely, if $\tilde{\delta}$ exceeds δ by θ or more, then we conclude that the concept is not FAC learned. The lower bound on coverage is computed by counting all those concepts for which the first case applies, and the upper bound by counting those for which the second case does not apply.

The level of confidence of the above estimate depends, of course, on the value of θ and the number of samples tested per concept. For example, in the experiments of the next section, setting $\delta = 0.10$, we have chosen to test 10,000 samples per concept, and let θ be 0.007 . Three possible outcomes were considered: (i) $\tilde{\delta} < 0.093$, (ii) $0.093 \leq \tilde{\delta} \leq 0.107$, and (iii) $\tilde{\delta} > 0.107$. The concept was counted FAC learned in the first case, and not FAC learned in the last case. Using the standard normal approximation of the binomial distribution, we can show that, for a given concept, the probability of making a wrong conclusion in this setting (i.e. counting a FAC learned concept as not FAC learned, and vice versa) is at most 0.01. Thus, this test gives a 99% level of confidence.

5 Experiments

The techniques introduced in the previous sections were directly applied in the evaluation of the following four learning algorithms:

1. **ID3** of Quinlan [Q86].
2. **FRINGE** as given in [PH89], with the maximum number of iterations set to 10.
3. **MDT(Minimum Decision Tree):** The algorithm that exhaustively searches for a decision tree consistent with the sample with a minimum number of nodes (ties broken randomly).
4. **RSC(Random Selection Criterion):** Same control structure as ID3 but choosing the root of the tree arbitrarily in each recursive call.

The last two algorithms are tested to check how the feature selection criteria implemented in ID3 and FRINGE are compared to the optimal (MDT) and arbitrary (RSC) behavior. Particularly, RSC was reported to achieve a level of performance surprisingly comparable to that of ID3 in some real domains [M89].

We also give the coverage of BALLS, the simple algorithm that classifies all the unseen examples as positive (negative) if the majority of the examples in the sample are positive (negative), respectively, breaking ties randomly. The exact coverage of this algorithm is obtained analytically using methods given in [AD90].

With the exception of FRINGE, all the algorithms given here are also symmetric with respect to complementing the concept itself. That is, if ID3 FAC learns the concept `abcd0000`, for example, then this implies that ID3 also FAC learns the complement concept `5432ffff`. Therefore, we actually ran our experiments for these algorithms only on the representative concepts of weight 0 to 2^{n-1}.

The learning parameters were as follows: $n = 5$, $m = 8, 10, 12, 14$ and 16, $\epsilon = 0.10$ and $\delta = 0.10$. The number of representatives tested was 1,228,158 for FRINGE, and 698,635 for the other algorithms. Determining the FAC learnability of these concepts was done in two passes. In the first pass, only 100 randomly-chosen samples per concepts were tested. Concepts for which the number of samples that resulted in ϵ-close hypothesis is less than 60 out of 100 (i.e. $\tilde{\delta} > .4$), were considered not FAC learned and thus excluded. In the second pass, the rest of the concepts were tested using 10,000 randomly-chosen samples, as explained in Section 4.

The specific concepts FAC learned by the five algorithms are shown in the appendix. The coverage figures of these algorithms are given in the following table. The last row in the table gives the maximum coverage that can not be exceeded by any algorithm, using the upper bound result given in [D89].

	Sample size				
Algorithm	8	10	12	14	16
ID3	12±0	332±0	396±0	1,756±0	4,954±640
FRINGE	12±0	332±0	396±0	1,756±0	5,284±970
MDT	12±0	12±0	116±40	496±0	3,694±0
RSC	66±0	66±0	226±0	226±0	1,698±0
BALLS	10,978	10,978	10,978	82,898	82,898
Upper Bound	661,333	2,041,173	6,148,551	17,985,991	50,753,991

Examining the above table and the results listed in the appendix, we found the behavior of the five tested algorithms to be largely unexpected, and sometimes counter-intuitive. We give here a few interesting points based on these data:

1. It is rather surprising that MDT always has less coverage than ID3 and FRINGE. MDT covers, however, some *interesting* concepts that both ID3 and FRINGE miss. An example of such a concept is `ff0000ff`, or $x_1 \oplus x_2$, for $m = 14$. On the other hand, the concepts `ff800000` and its complement (which have rather complex trees) are covered by ID3 and FRINGE but not MDT, for the same sample size. There is a total of 1280 concepts in the equivalence classes of `ff800000` and its complement, compared to only 20 in that of `ff0000ff`.
2. ID3 and FRINGE cover the same concepts for $m \leq 14$. When $m = 16$, FRINGE starts covering concepts such as `ff0000ff` that are not covered by ID3.
3. Unexpectedly, RSC has higher coverage than ID3 and FRINGE for $m \leq 12$. This is reversed, however, for larger sample sizes. We noticed that RSC works much better than ID3 or FRINGE for concepts like `80000000` ($x_1x_2x_3x_4x_5$) and `c0000000` ($x_1x_2x_3x_4$), but miserably misses very simple concepts such as `ffff0000` (x_1).
4. The coverage of all the above algorithms is far below [D89]'s upper bound.
5. The simple algorithm BALLS has substantially higher coverage than all of the other algorithms. However, the set of concepts that are covered by this algorithm are not interesting in a practical sense. Namely, this algorithm learns all the concepts of weight 0 to 3, and 29 to 32 for $m \leq 12$, and those of weight 0 to 4, and 28 to 32 for $m = 14$ or 16, whereas simple concepts such as `ffff0000`, are not learned by this algorithm unless almost all the examples are included in the sample. Nevertheless, the fact that there exists an algorithm with such a high coverage may possibly mean that the algorithms considered here stand substantial improvements.
6. Careful observation of the patterns shown in the appendix revealed that ID3 and FRINGE work well on concepts that can be either represented or ϵ-approximated by *intuitively simple* decision trees. However, when we tried to restrict the definition of simplicity using (i) the number of nodes (or, similarly, the number of leaves) in the tree [FI90], (ii) the rank of the tree [EH88], (iii) the average depth of the tree, or (iv) the number of distinct features tested in the tree, we found no single measure of these to be adequate to fully explain the behavior of these two algorithms. We expect that a more appropriate measure might be some weighted combination of these.

The above remarks motivate two things. First, the *plain* coverage does not seem suitable as the assessment figure of learning algorithms. This is illustrated by the points (1) and (4) above. Instead, one should consider assigning different weights to the concepts according to some preference measure that reflects a specific bias, and then take the *weighted coverage* as the evaluation measure. Alternatively, we might fix some ordering on the space of concepts and count a concept as covered only if all the preceding concepts are. Such measures would, of course, reflect how well a learning algorithm implements the underlying bias represented by the weights or the ordering over the concept space. Second, the overall results reported here indicate that expanded experiments with larger number of features, combined with formal analysis of the algorithms wherever possible, are needed to resolve the observed ambiguity in the behavior of these algorithms.

6 Conclusion

This paper dealt with reducing the computational cost of the empirical coverage measurement of inductive learning algorithms that learn Boolean concepts from examples. Two algorithm-independent techniques were introduced and applied to compare some of the well-known decision-tree building algorithms over the space of all Boolean concepts defined on five Boolean features. The coverage of the algorithms, and a list of all the concepts reliably learned by these algorithms in this setting were reported and discussed.

Carrying on such experiments would not be possible without the application of the two techniques we introduce. Unfortunately, however, analogous experiments with six or more Boolean features are still not feasible even when these cost reduction techniques are applied. Additional techniques that further help in cutting the cost of the experiments, are needed. We give here one promising direction.

Given the fact that any learning algorithm can learn only a small fraction of the concept space [D89], an encouraging approach is to analytically show, for a given algorithm, that a large portion of the space of concepts is not learned by the algorithm. One then needs to run the algorithm only on representatives of the potentially FAC learnable concepts, rather than testing for the whole concept space. For example, if it is known for some given learning parameters that H is the hypothesis space of a particular algorithm, then we can show that only the concepts in H and those concepts that are ϵ-close to some concept in H, are possibly learned by the algorithm. All other concepts can not be FAC learned by the algorithm and thus can be ignored. Clearly, unlike the techniques introduced in this paper, this method is algorithm-dependent. That is, the success of the method depends on how easy it is to analyze the algorithm in order to restrict H enough to make the experiments feasible. Also, the value of ϵ must be small for this approach to be useful. The validity of this and other approaches is currently under investigation.

Finally, an interesting future direction is to conduct the domain-independent comparative study we followed here on some other learning algorithms, with the possible extension of using noisy data, to test the robustness of the algorithms against the presence of noise.

References

[AD90] H. Almuallim, T. G. Dietterich. Coverage of Inductive Learning Algorithms. *Workshop on Computational Learning Theory and Natural Learning Systems*, Princeton, NJ. Sep 5-6, 1990.

[D89] T. G. Dietterich. Limitations on Inductive Learning. In *Proceedings of the Sixth International Conference on Machine Learning* (pp. 124–128). Ithaca, NY. San Mateo, CA: Morgan Kaufmann, 1989.

[EH88] A. Ehrenfeucht and D. Haussler. Learning Decision Trees From Random Examples. *Proceedings of the First Workshop on Computational Learning Theory*, 182-194, 1988.

[FI90] U. Fayyad and K. Irani. What Should Be Minimized in a Decision Tree? *Proceedings of the Eighth National Conference on Artificial Intelligence* p. 749–754, 1990.

[H65] M. Harrison. Introduction to Switching and Automata Theory. McGraw Hill, 1965.

[M89] J. Mingers. An Empirical Comparision of Selection Measures for Desicion-Tree Induction. *Machine Learning*, 3 (4), 319-342, 1989.

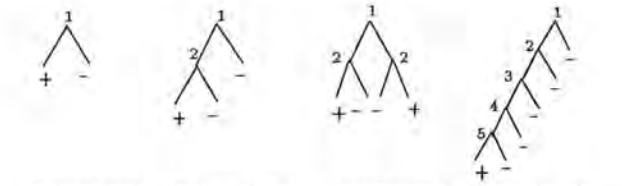
[PH89] G. Pagallo and D. Haussler. Boolean Feature Discovery in Empirical Learning. *Machine Learning*, 5 (1), 71-100, 1990.

[Q86] J. R. Quinlan. Induction of Decision Trees, *Machine Learning*, 1(1):81-106, 1986.

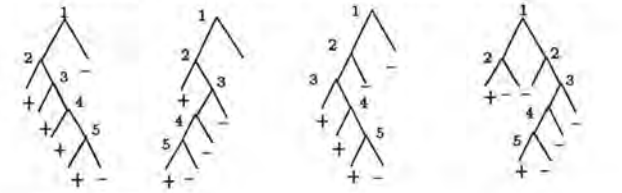
[RHW86] D.E. Rumelhart, G.E. Hinton and R.J. Williams. Learning Internal Representations by Error Propagation. In D.E. Rumelhart and J.L. McClelland, (eds.) *Parallel Distributed Processing*, Vol 1.

Appendix:

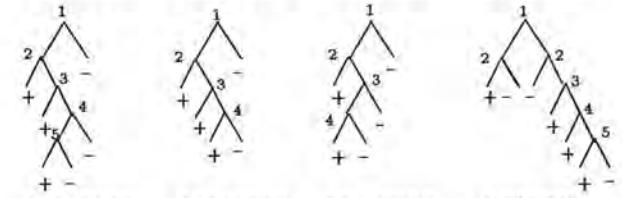
The table on the right shows the representative concepts that were found to be FAC learned by at least one of the algorithms we tested. The FAC learnability result for any concept f in the table, applies to all the concepts obtained by permuting and/or negating the features of f. All other concepts are not FAC learned by any of the algorithms under consideration. A $\surd$ is used to indicate that the concept was FAC learned ($\tilde{\delta} < 0.093$), while a $\times$ means that the concept was not FAC learned ($\tilde{\delta} > 0.107$). Concepts at the border ($0.093 \leq \tilde{\delta} \leq 0.107$) for which no confident decision is made (and, thus, fall in the range between the upper and lower estimates of coverage) are marked by a Δ. In the table, only the representative concepts of weight 0 to 16 are listed. For ID3, MST and SRC, the results of these determine the results for concept classes of weight 17 to 32, since these algorithms are symmetric with respect to complementation of concepts. For FRINGE, however, for each representative concept f of weight 0 to 15, the result of $\bar{f}$ is given explicitly in the table. "size" indicates the number of concepts in the equivalence class. The decision tree representation of each concept is given below.



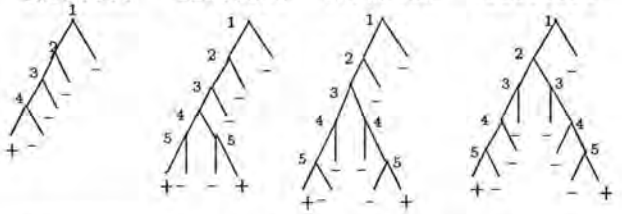
ffff0000 ff000000 ff0000ff 80000000



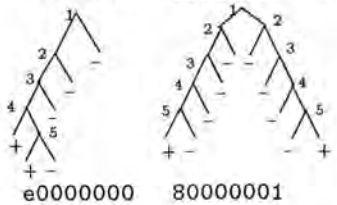
fffe0000 ff800000 fe000000 ff000080



fff80000 fffc0000 ffc00000 ff0000fe



c0000000 90000000 81000000 80010000



e0000000 80000001

f	m	id3 $f, \bar{f}$	fringe f	fringe $\bar{f}$	mdt $f, \bar{f}$	rsc $f, \bar{f}$
0	8	√	√	√	√	√
	10	√	√	√	√	√
size:	12	√	√	√	√	√
1	14	√	√	√	√	√
	16	√	√	√	√	√
ffff0000	8	√	√		√	×
	10	√	√		√	×
size:	12	√	√		√	×
10	14	√	√		√	×
	16	√	√		√	×
ff000000	8	×	×	×	×	×
	10	×	×	×	×	×
size:	12	×	×	×	△	×
40	14	√	√	√	√	×
	16	√	√	√	√	×
ff0000ff	8	×	×		×	×
	10	×	×		×	×
size:	12	×	×		×	×
20	14	×	×		√	×
	16	×	△		√	×
80000000	8	×	×	×	×	√
	10	×	×	×	×	√
size:	12	√	√	√	√	√
32	14	√	√	√	√	√
	16	√	√	√	√	√
fffe0000	8	×	×	×	×	×
	10	√	√	√	×	×
size:	12	√	√	√	×	×
160	14	√	√	√	√	×
	16	√	√	√	√	×
ff800000	8	×	×	×	×	×
	10	×	×	×	×	×
size:	12	×	×	×	×	×
640	14	√	√	√	×	×
	16	√	√	√	√	×
fe000000	8	×	×	×	×	×
	10	×	×	×	×	×
size:	12	×	×	×	×	×
320	14	×	×	×	×	×
	16	√	√	√	√	×
ff000080	8	×	×	×	×	×
	10	×	×	×	×	×
size:	12	×	×	×	×	×
320	14	×	×	×	×	×
	16	△	△	×	√	×
fff80000	8	×	×	×	×	×
	10	×	×	×	×	×
size:	12	×	×	×	×	×
960	14	×	×	×	×	×
	16	√	√	√	×	×
fffc0000	8	×	×	×	×	×
	10	×	×	×	×	×
size:	12	×	×	×	×	×
320	14	×	×	×	×	×
	16	△	△	△	×	×
ffc00000	8	×	×	×	×	×
	10	×	×	×	×	×
size:	12	×	×	×	×	×
960	14	×	×	×	×	×
	16	×	×	△	×	×
ff0000fe	8	×	×	×	×	×
	10	×	×	×	×	×
size:	12	×	×	×	×	×
320	14	×	×	×	×	×
	16	×	×	×	√	×
c0000000	8	×	×	×	×	×
	10	×	×	×	×	×
size:	12	×	×	×	×	√
80	14	×	×	×	×	√
	16	×	×	×	×	√
90000000	8	×	×	×	×	×
	10	×	×	×	×	×
size:	12	×	×	×	×	×
160	14	×	×	×	×	×
	16	×	×	×	×	√
81000000	8	×	×	×	×	×
	10	×	×	×	×	×
size:	12	×	×	×	×	×
160	14	×	×	×	×	×
	16	×	×	×	×	√
80010000	8	×	×	×	×	×
	10	×	×	×	×	×
size:	12	×	×	×	×	×
80	14	×	×	×	×	×
	16	×	×	×	×	√
e0000000	8	×	×	×	×	×
	10	×	×	×	×	×
size:	12	×	×	×	×	×
320	14	×	×	×	×	×
	16	×	×	×	×	√
80000001	8	×	×	×	×	×
	10	×	×	×	×	×
size:	12	×	×	×	×	×
16	14	×	×	×	×	×
	16	×	×	×	×	√