ECONOMICAL TRAINING SETS FOR LINEAR ID3 LEARNING

William A. Greene, Ph.D.
Computer Science Department
University of New Orleans
New Orleans, LA 70148
bill@cs.uno.edu

Abstract: Our work is in machine learning, a subfield of artificial intelligence. We describe a variant of the ID3 algorithm [5] which is attuned to the situation that every feature's value-set is linearly ordered and finite. We then seek economical training sets, that is, ones which are small in size but result in learned decision trees of high accuracy. Our search focuses on geometric properties of the target concept, such as its extreme points, edges, faces, and surface. We categorize all concepts into three classes, from simplest to most general, and for each class we identify certain training sets, some quite small, others less so, which result in highly accurate learning of the concepts in that class. Some of our results are rigorously provable (but the proofs do not appear here), for other results our evidence is empirical.

1. Introduction

Relatively little attention has been paid to the methodical selection of training examples in machine learning problems. Typically the examples used are whatever ones chance has provided or are chosen intentionally randomized. On the more methodical and selective side, Winston [8] first introduced the notion of a "near miss", using the definition: a negative example of a concept but which "differs from that concept in only a small number of significant points". Ng, Buchanan, Rissland, Rosenbloom, & Johnson [4] have recently provided us with a more precise definition of a "near miss" and illustrated its significance in the context of their Version Space style algorithm. Their definition of a near miss is: a feature-vector which is a negative example but which can be transformed into a positive example by changing just a single feature's value. Rendell & Cho [6] discuss how learning can be influenced by "concept character" where they have in mind ideas such as how many "lumps" (our term) a concept consists of and how spread out and/or aligned are the lumps. The ideas of [6] really pre-suppose a degree of structure on featural value-sets sufficient to suggest notions of distance, position, etc.

We begin with features whose value-sets are linearly ordered, and interest ourselves in concept elements having geometrical significance, such as the corners, edges, faces, and surface.

Some precise definitions are now in order. There is a universe of interest to us. *Objects* in the universe are described in terms of the values they exhibit along various featural dimensions. A pertinent *feature* (synonym: *domain*) might be SIZE, with value-set {SMALL, MIDDLING, LARGE}. If other features are COLOR and SHAPE, then a typical object would be (identified with) the *feature-vector* (SMALL RED SQUARE). If there are d (for dimension) features $\{F_k\}_{k\in 1..d}$ with corresponding value-sets $\{V_k\}_{k\in 1..d}$ then we take the *universe* of interest to be the Cartesian product $U = V_1\times V_2\times \ldots \times V_d$. By a *concept* we mean an arbitrary subset of the universe U. The term *instance* we take as a synonym for object (or feature-vector). Given a concept at hand, by an *example* we mean an instance additionally tagged with its classification (+/–) with respect to membership in that concept.

We interest ourselves in the case that every feature's value-set is linearly ordered and finite, so, is isomorphic to a subinterval of integers. (Probably COLOR and SHAPE do not pass this test.) By a *hyper-rectangle* we mean a subset of U of the form $I_1\times I_2\times \ldots \times I_d$ where for each k, set I_k is a subinterval of V_k. Every subset of U is a union of (not necessarily disjoint) hyper-rectangles. To see this, consider the fact that a single feature-value *v* can be considered to be the subinterval [*v*, *v*] of V_k, so each single point of U is a hyper-rectangle, and a U-subset is certainly the union of its points. For economy's sake, in general we would want to choose the hyper-rectangles participating in such a union to be bigger in size than single points. In general there is no tidy canonical way to express a U-subset as a union of large hyper-rectangles. For instance, in the 2-dimensional case (that is, 2 = d = the number of features) illustrated in Figure 1, there are shown 3 rectangles with solid boundary lines, and the patterned lines suggest another choice of a rectangle that could be used to define the planar region that is their union.

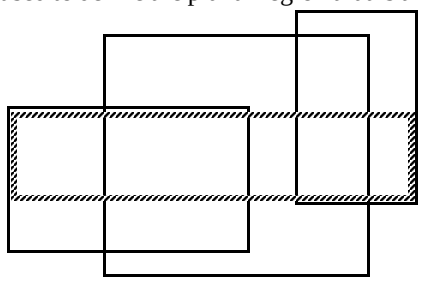

Fig. 1

A concept is by definition an arbitrary subset of the universe. If every feature's value-set is by nature linearly ordered, then there is much appeal in expressing the concept as a union of hyper-rectangles. Such an expression not only incorporates the linearity naturally present but is also very close to a logical expression of the propositional calculus in disjuntive normal

form [2]: to be inside one hyper-rectangle is to satisfy a conjunction of predicates of the form "the kth coordinate is in interval I_k", and to be in a union of hyper-rectangles is to satisfy a disjunction of such conjunctions.

2. The linear ID3 algorithm

Supposing that we are presented with a *training set* of positive and negative examples of some concept, we wish to make the inductive leap to a description of the concept, or more accurately, to a description of a plausible candidate for the unknown concept. We shall assume the training set is free of noise. Our description will take the form of a decision tree, built using an ID3 style algorithm [5] which is attuned to the situation that every value-set V_k is linearly ordered and finite. At each stage of the tree-building we make a three-way (sometimes only two-way) branching, which corresponds to partitioning an interval I_k of feature F_k's values into a left, central, and right subinterval. As in Quinlan [5], an information-theoretic heuristic drives the selection of a particular interval I_k and partitioning thereof. Geometrically, the intention is to clamp down upon a hyper-rectangle along the direction of one of the featural axes. And indeed, the leaves of the decision tree will correspond to a disjoint union of hyper-rectangles.

The tree-building algorithm is recursive, and takes two parameters. One of the parameters is a set S of examples. The other parameter is a set of d intervals $\{I_k\}_{k\in 1..d}$, where I_k is a subinterval of the value-set V_k of the kth feature F_k.

Overview: For each interval I_k, we look at all possible ways of partitioning I_k into a left, central, and right subinterval. Each such partitioning of I_k gives rise to a partitioning of the example set S into subsets S_L, S_C, S_R, where S_* (* = L, C, R) consists of those S-elements whose k^{th} featural value falls into, respectively, the left, central, or right subinterval of I_k. Each subset S_* is measured for its *purity* [1, p. 24], that is, the extent to which it approximates consisting of exclusively positive or exclusively negative examples, as measured by the information-theoretic function of Quinlan [5] (this same function is also cited by [1, p. 25]). A weighted average of the three purity measures is formed. The (feature with) interval and partitioning thereof having the best weighted average of purity measures is then what determines the next step of tree-building.

Now we flesh out in detail the ideas of the preceding paragraph. If p and n are respectively the number of positive and negative examples in some set of examples, then as in Quinlan [5], we define I(p,n) =

$$-\left((p/(p+n))\cdot\log_2(p/(p+n)) + (n/(p+n))\cdot\log_2(n/(p+n)) \right) .$$

We note that I(p,n) approximates 1 when p approximates n, also that I(p,n) approximates 0 when p is much bigger than n or n is much bigger than p, and finally that I(p,n) = 0 when p = 0 or n = 0. Thus the example set is the more nearly pure the closer is I(p,n) to zero.

Let an interval $I_k = [lo, hi]$ of the kth feature F_k, and an example-set S, be given. We consider all partitionings of I_k into a left, central, and right subinterval. Namely, we consider all possible pairs of cutpoints satisfying *lo* ≤ *left-cutpoint* ≤ *right-cutpoint* ≤ *hi*. Note we permit the case of only two subintervals, and the possibility that a subinterval contains only one value. Now, let p_L, p_C, p_R be the number of positive examples from S whose k^{th} feature-value lie in, respectively, the left, central, or right subinterval of I_k. And n_L, n_C, n_R are the analogous counts of negative examples. The weighted average of purity measures for the three subintervals we then define to be

$$\left\lfloor \sum (p_I + n_I) I(p_I, n_I) \right\rfloor / |S| \quad .$$

(Here |S| is the number of elements in S). This latter expression is more or less what Quinlan names an E-value.

At each stage of tree-building (that is, at each node of the developing decision tree) we have at hand a set S of examples and a set $\{I_k\}_{k\in 1..d}$ of intervals. If S is pure, then we label this tree node with the common classification (+ or –) shared by all of S's elements. Otherwise, we identify the interval I_j and partitioning $\{I_{j,L}, I_{j,C}, I_{j,R}\}$ thereof with the best (= lowest) E-value and make a three-way (sometimes only two-way) branching at this node, corresponding to the three subintervals $I_{j,*}$. As the reader will by now have anticipated, we continue tree-building for, say, the left subinterval by recursing on S_L and the interval set $\{I_{j,L}\}\cup\{I_k\}_{k\neq j}$.

We must have had a reason for building our decision tree: we will use it for predicting class membership (+/–) of novel, arbitrary instances in the universe, with respect to the original (and typically unknown) concept at hand. The tree will correctly predict the class of every instance encountered in the training set (recall that the training set is assumed noise-free), but may be inaccurate on other instances. In the present research we have sought economical training sets, that is, ones small in size but which lead to highly accurate trees.

3. Experimental results

Next we describe our experiments and their results. It should be noted at the outset that we work with *known* concepts, and explore which sets of training examples consisting of U-points having geometric significance lead our ID3-variant to rather accurate decision trees. This is in contrast to reality, when the target concept's definition or structure is *unknown*. Nonetheless our results reveal much about the behavior of ID3 learning when features have linear value-sets.

3.1. Levels of difficulty of learning

Our experimental results lead us to the conclusion that there are three levels of difficulty of learning. In order of increasing difficulty they correspond to the cases that the target concept is:

- a single hyper-rectangle;
- the union of several hyper-rectangles which are spread out far apart from one another;
- the union of several hyper-rectangles which can be close together or overlap (have non-empty intersection).

Of course, the last case just listed is the general case. These conclusions, and especially the middle case, confirm some of the speculations of Rendell & Cho [6].

Many results reported below are for experiments conducted in the case where universe U is 3-dimensional, $U = V_1 \times V_2 \times V_3$, and where each V_k is the integer interval [0,25]. Three is fairly low for dimensionality; a later remark explains why this low dimensionality was used. Where feasible we have conducted experiments in higher dimensionality (10, for instance) and obtained results consistent with the ones reported here.

3.2. A single hyper-rectangle

In mathematics the Krein-Milman theorem [7, Chapter 2, Theorem 10.4] states that if set X is a compact convex subset of a locally convex topological vector space then the closed convex hull of (that is, the smallest closed convex set containing) the set of extreme points of X (points not interior to any line segment contained in X) exactly equals X itself. And another theorem [7, Chapter 2, §9] states that the closed convex hull of a set Y is the intersection of all the sets H which contain Y and which are closed half-spaces lying "to one side of" a hyper-plane. For our purposes, the first theorem implies that a hyper-rectangle $I_1 \times I_2 \times ... \times I_d$ in universe $U = V_1 \times V_2 \times ... \times V_d$ is the (closed) convex hull of its set of corners. We additionally note that our ID3 variant slices the universe U by hyper-planes which are parallel to the faces of a(ny) hyper-rectangle. Thus arises the suggestion that a good set of training examples is provided by using the corners as positive examples and using certain points nearby the corners as negative examples.

First let us count the corners in a hyper-rectangle $I_1 \times I_2 \times ... \times I_d$. There are 2^d such corners, because a feature-vector $(v_1, v_2, ..., v_d)$ is a corner if and only if each component v_k is one of the 2 endpoints of interval I_k. By an *off-corner* of a hyper-rectangle we mean a point one value away from a corner in every featural direction. Each corner has one associated off-corner. In 6-space, the off-corner of hyper-rectangle $[2,5]^6$ associated with corner (2,2,5,2,5,2) is (1,1,6,1,6,1), and that associated with corner (2,2,2,5,5,5) is (1,1,1,6,6,6). Roughly put, an off-corner can be imagined as the "next" point after the corner, on a line drawn from the hyper-rectangle's center and on through the corner.

Even when the target concept is a single hyper-rectangle, using the corners and off-corners as, respectively, the positive and negative examples of the concept leads to a much over-generalized decision tree. Only one feature-value, and in our implementation it is the first feature-value, is correctly constrained to its corresponding interval I_1; according to the learned tree, other feature-values can be arbitrary. Thus, the set of corners and off-corners badly fails to be a desirable training set.

Measuring accuracy

Here is a good point to discuss the tricky issue of how to measure the (in)accuracy of a decision tree. To measure inaccuracy as the percentage of U-points that are misclassified by the decision tree may not be the best choice. In 2-space $U = [1,100]^2$, consider the two isomorphic hyper-rectangles $H_1 = [2,99] \times [25,26]$ and $H_2 = [25,26] \times [2,99]$. If as described above, when corners and off-corners comprise the training set, it is only the first feature-value that is correctly constrained in the decision tree, then the decision tree for H_1 misclassifies the elements of the set $[2,99] \times ([1,24] \cup [27,100])$, which is 96.04% of U. But for H_2, only the elements of $[25,26] \times ([1,1] \cup [100,100])$, or 0.04% of U, get misclassified. For essentially the same mistake made on two isomorphic hyper-rectangles, the swing between these two error percentages is an uncomfortably wide one.

When concepts are as simple as single hyper-rectangles, then a better error indicator might be the count of features incorrectly constrained, but such a simple error indicator is not appropriate for concepts that are unions of several hyper-rectangles. Also, in general, counting misclassified U-points tends to make an error measurement sensitive to the relative size of interval I_k within the feature's value-set V_k; this sensitivity is not necessarily a good thing.

We have not been able to arrive at an ideal measurement of accuracy.

Recall that Ng *et al*. [4] define a "near miss" to be a feature-vector which is a negative example of a concept but which can be converted into a positive example by changing the value of just one component of the feature vector. An off-corner is not a near miss under this definition, since its every component value must be changed to transform it into a corner.

If feature F_k's value-set V_k is linearly ordered and consists in order of, say, the eight values

$$a < b < c < d < e < f < g < h$$

then we have a notion of distance between feature values: *d* is 2 units away from *b*, and is 4 units away from *h*. The presence of distance functions within value-sets V_k allows us to define distance metrics on the universe U. Two such U-metrics could be defined as follows: if *x* and *y* are arbitrary points in U, define

dist-1(*x*,*y*) = sum of intra-featural distances between *x* and *y* components;

dist-2(x,*y*) = max of intra-featural distances between *x* and *y* components.

For our purposes, a better definition of a *near miss* is: a negative example which under metric dist-1 is at distance 1 from a positive example. This is tantamount to modifiying the definition of [4] to: a negative example which can be converted into a positive example by changing the value of just one component *by just one unit*; their defintion of a *value near miss* subsumes our definition of a near miss.

Given a corner of a hyper-rectangular concept in $U = V_1 \times V_2 \times ... \times V_d$, by the corner's associated *corner-cappers* we

mean the d negative examples at distance 1 away from the corner under metric dist-1. For example, for hyper-rectangle $[3,8]^4$ in 4-space, its corner (3,8,8,3) has associated corner-cappers (2,8,8,3), (3,9,8,3), (3,8,9,3), and (3,8,8,2). Note that a corner-capper is one unit away from the corner in the direction of the corresponding featural axis.

Result 1: If a concept is a single hyper-rectangle, then using its 2^d corners and $d2^d$ corner-cappers as, respectively, the positive and negative training examples, results in a learned decision tree which is 100% accurate and contains one positive leaf (a decision leaf tagged for predicting "positive").

The algorithm clamps down upon the hyper-rectangle in d steps; at each step it shaves away the 2^d corner-cappers that stand away from their corresponding 2^d corners in one of the d directions.

Result 1 first became known to us via experimental methods, namely, directly testing with particular hyper-rectangles. We have since been able to devise a mathematical proof of Result 1. The same remarks can be made about Results 2 and 3 below; they too are provable. But the proofs are non-trivial and would increase this paper's length by fifty percent or more, and are not included here. Interested readers can obtain the proofs by contacting the author. In our Appendix we gloss the proof of Result 1. (For Results 4, 5, and 6 below, we have no theoretical proofs, only the empirical observations given below.)

For the hyper-rectangle $[3,12]^6$ in 6-space, the corner (3,3,3,12,3,12) has *opposing corner* (12,12,12,3,12,3). Letting this example suffice to inform the reader what we mean by an opposing corner, we have the:

Result 2: If a concept is a single hyper-rectangle, then using any 2 opposing corners and their 2d associated corner-cappers as, respectively, the positive and negative training examples, results in a learned decision tree which is 100% accurate and contains one positive leaf.

This is a remarkably economical training set. Supposing a concept is a single hyper-rectangle and that there exists an oracle that can report whether instances presented to it are positive or negative for the concept, then an interesting problem would be to devise an algorithm that identifies the concept by searching for two opposing corners of the hyper-rectangle. We have not investigated this issue to any depth. But below we do give a (related and) fast simple algorithm that, by querying an oracle, will exactly identify a hyper-rectangular concept.

Another very economical training set exists, when a concept is a single hyper-rectangle. The hyper-rectangle $I_1 \times I_2 \times ... \times I_d$ has 2d faces, 2 in each of the d featural directions (put another way, 2 perpendicular to each of the d featural axes), corresponding to the 2 endpoints of a typical interval I_k. By a *face-center* we mean a feature-vector constructed as follows: pick a feature F_k; when $j \neq k$, use as component value of the feature-vector the middle value of the interval I_j; when $j = k$, use one of the two endpoints of I_k; the 2 feature-vectors so constructed are the centers of the 2 faces perpendicular to the kth axis. And an *off-face-center* is a feature-vector defined similarly except that for $j = k$ we use a value one unit outside the interval I_k. Roughly, a face-center is where a line drawn from the center of the hyper-rectangle and perpendicular to a face intersects the face, and an off-face-center is the next point beyond that intersection.

Result 3: If a concept is a single hyper-rectangle, then using the 2d face-centers and 2d off-face-centers as, respectively, the positive and negative training examples, results in a learned decision tree which is 100% accurate and contains one positive leaf.

Exact centrality of the face-center within the face is probably not important to the algorithm's behavior, which is to clamp down upon the hyper-rectangle in d steps. So as to segregate away two opposing off-face-centers, on each step it shaves down to two opposing faces. Choosing one point at random from each of the 2d faces and similarly choosing points from the off-faces will, we believe, again result in a 100% accurate tree, but we have not tested this idea.

If a concept is a single hyper-rectangle $I_1 \times I_2 \times ... \times I_d$, and supposing there exists an oracle that will report whether an instance presented to it is positive or negative, then the following simple algorithm will quickly and exactly identify the concept. We must begin with a positive instance, that is, one inside the hyper-rectangle; call it $v = (v_1, v_2, ..., v_d)$. For each direction k in 1..d, do the following: changing just the kth component v_k of v, make a binary search towards the least value in V_k, using the oracle to determine when one exits the concept, i.e., has reached the low endpoint of the hyper-rectangle's k^{th} slice I_k. Do likewise, searching towards the greatest value in V_k, to identify the high endpoint of I_k. If the value-sets have maximum size M, then this algorithm takes at most 2d log M steps and exactly identifies the hyper-rectangle.

3.3. Several hyper-rectangles spread far apart

Our original hoped-for goal was to identify relatively small training sets of points with geometric significance that would result in learned decision trees that were 100% accurate. This goal has partially eluded us. We can find small-ish training sets that produce somewhat inaccurate trees; conversely, we can produce 100% accurate trees if we allow somewhat large-ish training sets.

Here we consider a concept that is the union of several hyper-rectangles which are somewhat spread apart and distant from one another. In this case we got best results from using as positive (respectively, negative) examples the totality of corners (respectively, corner-cappers) of the various hyper-rectangles. For example, consider the concept which is the union of the 8 hyper-rectangles in 3-space $U = [0,25]^3$ which can be built by using either integer interval [3,7] or interval [12,16] in the role of interval I_k (k = 1,2,3) in $I_1 \times I_2 \times I_3$. With some exaggeration we call this a *space-filling* concept. Using the grand

total of 256 corners and corner-cappers as training examples for this concept produced a 100% accurate tree.

Using instead the grand total of 96 face-centers and off-face-centers of this same space-filling concept resulted in a much over-generalized tree that purported 4456 positive instances instead of the 1000 positive instances that the concept actually has. (Incidentally, over-generalization is not the only way our trees can err. In general, one of our trees can predict an instance to be positive when in fact it is negative, and also predict some other instance to be negative when in fact it is positive.)

Turning our attention back to corners and corner-cappers, inaccuracies begin to appear in the tree when participating hyper-rectangles get too close together or overlap. For example, the learned tree is correct for the concept $[2,10]^3 \cup [12,20]^3$ in 3-space $U = [0,25]^3$. But for concept $[2,10]^3 \cup [11,19]^3$ (note the second hyper-rectangle has moved closer to the first one), the learned tree disagrees with the concept at 567 of the 26^3 = 17,576 points of U.

Continuing the previous paragraph, we observe that 567 is 3.2% of the points of U. Is this acceptable accuracy? That would depend on the setting. Again we note that counting points of disagreement between concept and tree may not be an ideal measure of inaccuracy. In particular for this example, the tree overgeneralizes and it follows that inaccuracy so measured would decrease if the surrounding universe shrank to, say, $[0,21]^3$.

Earlier we promised to explain why many of our reported experiments were conducted in 3-space. Now we can do so. To count the number of points in U where a concept disagreed with its associated tree, we used the brute force method of generating all the points of U and comparing the concept and tree at each such point. Higher dimensional spaces made this brute force counting too time consuming. That is not to say that tree construction is overly time costly. For instance, using the altogether 40 face-centers and off-face-centers of hyper-rectangle $[3,7]^{10}$ in 10-space $U = [0,19]^{10}$, (note $|U| = 20^{10}$), construction of the (100% accurate) tree took only 140 seconds.

Although describing hyper-rectangles as being "spread far apart" is inexact, we will nonetheless grant ourselves the:

Result 4: If a concept is the union of several hyper-rectangles which are spread far apart, then using their totality of corners and corner-cappers as, respectively, the positive and negative training examples, appears to produce accurate trees.

An interesting question, unexplored by us, is whether it would suffice to use not all 2^d corners but only two opposing corners from each hyper-rectangle.

3.4. An arbitrary union of hyper-rectangles

When the participating hyper-rectangles in a concept are close together or overlap, finding a small-ish training set that produces highly accurate trees becomes more elusive. The empirical evidence (Result 5 below) is that one can achieve 100% accurate learning by using the large-ish training set of all points at the boundary between a concept and its complement. For a small-ish training set producing highly accurate learning, it occurred to us to think in terms of, say, corners and corner-cappers of the various hyper-rectangles, plus additional points situated where the hyper-rectangles are intersecting one another.

We assume the reader understands what we mean by the *surface* of the hyper-rectangle $I_1 \times I_2 \times ... \times I_d$. By the *off-surface* of this hyper-rectangle we mean the surface of the hyper-rectangle obtained by inflating every interval I_k by one unit at both ends. Most points in the off-surface stand 1 unit away from the surface under metric dist-1, but some are farther away -- the off-surface includes, for instance, the off-corners.

Given two hyper-rectangles that intersect, by *crease-points* we mean points in the intersection of the two hyper-rectangles that are not interior to either hyper-rectangle. (The crease-points comprise the intersection of the surfaces of the hyper-rectangles.) By *positive-crease-neighbors* (respectively, *negative-crease-neighbors*) we mean those points in the off-surface of the intersection that belong to either hyper-rectangle (respectively, belong to neither hyper-rectangle).

It was our hope that using corners, corner-cappers, crease-points, and positive/negative-crease-neighbors to train would result in 100% accurate trees. In our experiments, often this happened, but as often as not inaccurate trees were learned. There is a certain fragility; consider the following. We used $[2,12]^3$ as the first hyper-rectangle in the union of 2 such. When we take $[4,14] \times [3,13] \times [11,17]$ as second hyper-rectangle, their union resulted in a learned tree that was 100% accurate. But when the second hyper-rectangle is slightly shifted to $[4,14] \times [4,14] \times [11,17]$, the resulting tree disagreed with the concept at 576 points of $U = [0,25]^3$, or 3.3% of U.

We were unable to achieve 100% accurate trees using other combinations of bounding points plus points around the intersection. For example, for concept $[2,12]^3 \cup [10,20]^3$ in $U = [0,25]^3$, using face-centers, off-face-centers, crease-points, and negative-crease-neighbors producted a tree that disagreed with the concept at 1656 points, or 9.5% of U.

There does appear to be a large-ish training set that always produces 100% accurate trees. We have not been able to mathematically prove the next result, but in many experiments, some using randomly generated concepts, it has always been borne out.

Result 5: For an arbitrary concept, using its surface and off-surface points as, respectively, the positive and negative training examples, appears to always result in a learned decision tree which is 100% accurate.

To construct the surface of a concept, we take the union of the surfaces of its hyper-rectangles; it seems to make no difference if we discard elements which are on the surface of one hyper-rectangle but are interior to another hyper-rectangle. To construct the concept's off-surface, we take the union of off-

surfaces of participant hyper-rectangles and then discard elements which belong to the concept. The surface of a hyper-rectangle is comprised of its 2d faces, each of which is isomorphic to a (d–1)-dimensional hyper-rectangle. If the edges of the faces have average size L, then the surface consists of approximately $2dL^{d-1}$ points.

The entirety of the set of surface and off-surface elements for a concept is not needed. Smaller, randomly chosen subsets of this set can produce trees of high accuracy. This is demonstrated by the experiments next described.

A concept is formed which is the union of 5 hyper-rectangles in 3-space $U = [0,25]^3$. Every hyper-rectangle is built by forming the Cartesian product of 3 intervals, each chosen at random from within [0,25]. The concept's surface and off-surface training set is built. This training set is winnowed to smaller and smaller subsets; with each, a tree is learned, and tested for accuracy. Since hyper-rectangle's intervals are chosen at random within [0,25], we take as a fair measure of inaccuracy the percentage of points in U where the learned tree disagrees with the concept.

The activities just described were repeated for 7 concepts. The results are summarized in Table I, below. The tabulated results suggest a:

Result 6: For an arbitrary concept in 3-space, using as few as 10% of the total of its surface and off-surface elements as, respectively, the positive and negative training examples, appears to result in a learned decision tree with inaccuracy seldom worse than 5%.

In this paragraph we briefly remark upon another training set with which we experimented. It occurred to us that a concept's edge-points (points on line segments joining adjacent corners) might provide something of an informative "wire frame" of positive examples, which if coupled with corner-cappers as negative information around where edges meet, could lead to accurate learning. (We chose to discard an edge-point of one hyper-rectangle if the point was interior to another hyper-rectangle. And of course we discarded a corner-capper on one hyper-rectangle if it belonged to another hyper-rectangle in the concept.) The accuracy of trees produced from such training sets showed surprising variance. As above, we experimented with concepts that were the union of 5 randomly constructed hyper-rectangles in 3-space $U = [0,25]^3$. Sometimes tree inaccuracy was low; sometimes the tree disagreed with the concept on as much as 44% of U.

4. Conclusions and future work

When all featural value-sets are linear and finite, we have succeeded in identifying small-ish training sets having geometrical significance which result in learned decision trees of high accuracy. We have categorized concepts into levels of difficulty of learnability. Results 1, 2, and 3 are mathematically provable (not done here, but see the Appendix below). In the case of an arbitrary concept, the reader may protest that 10% of the surface and off-surface is not necessarily so small-ish; this is an arguable point.

With regard to future work, here we pose some unexplored and unanswered questions. A learned decision tree's leaves correspond to a disjoint union of hyper-rectangles. Sometimes what started out as just two or three overlapping hyper-rectangles winds up as a complex union of disjoint slabs; for comprehensibility's sake, how might we simplify the tree by coalescing hyper-rectangles tagged positive into a smaller number of such, now no longer necessarily disjoint? If a tree misclassifies a future instance, then the leaf used to classify it corresponds to a hyper-rectangle which is mis-constrained in one or more of its components; is there an easy way to patch the tree? How should noise in the training set be handled? How should the tree be pruned [3]?

5. Appendix

In this appendix we summarize the direction taken in proving our Result 1.

Definition: Let S be an example set, let P be a partitioning of S, $P = \bigcup_{i=1}^{n} S_i$. Let p_i, n_i, resp., be the number of, resp., posi-

Table I. Disagreement counts, as surface/off-surface subsets are winnowed.

First column gives number of one of 7 random concepts; second gives size of the random concept; third gives size of full training set. Other entries are disagreement counts when example set is winnowed to percentage given in column heading. Column for 10% shows in parentheses percent disagreement over $U = [0,25]^3$.

#	CncptSize	EgSize	100%	75%	50%	30%	20%	15%	10%	5%
1	774	1904	0	4	28	74	101	125	685(3.9%)	5380
2	250	1095	0	22	50	459	807	662	1602(9.1%)	2479
3	489	1333	0	9	22	55	231	250	876(5.0%)	2942
4	4296	3967	0	7	18	85	117	133	289(1.6%)	495
5	4283	5180	0	0	0	37	114	330	499(2.8%)	1475
6	7244	5762	0	11	16	58	78	197	252(1.4%)	1539
7	4126	5476	0	8	26	54	150	246	667(3.8%)	1261

tive, negative examples in S_i. Define the *E-value* of partition *P* as $E(P) = \left[\sum^{n} (p_i + n_i) I(p_i, n_i) \right] / |S|$

A partition *P´* is a *pure simple refinement* of *P* if *P´* is obtained from *P* by splitting one of the elements of *P*, say the j-th one S_j, into two subsets, one of which is pure.

Theorem: If *P´* is a pure simple refinement of *P* then E(*P*) ≥ E(*P´*).

Recall that for Result 1 the concept is a single hyper-rectangle and the training set S consists of the corners and corner-cappers of the hyper-rectangle. When one projects the training set in the kth direction onto feature F_k, training set S maps to four values in F_k, namely, the two endpoints of interval I_k and each's neighbor outside I_k. There are essentially only five different ways that three intervals in F_k can partition S's image. One of these five, call it B (for best) is characterized by its central subinterval being identical to I_k. One argues that B is a pure simple refinement or is comparable to a pure simple refinement of each of the other four ways. Choice B results in shaving S down to two pure subsets plus a third S´ on which one argues that this same foregoing reasoning can be applied again.

Bibliography

1. Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Pacific Grove, CA: Wadsworth.
2. Mendelson, E. (1979). *Introduction to mathematical logic* (2nd ed.). New York: Van Nostrand.
3. Mingers, J. (1989). An empirical comparison of pruning methods for decision tree induction. *Machine Learning 4*, 227-243.
4. Ng, H. T., Buchanan, B. G., Rissland, E. L., Rosenbloom, P. S., & Sullivan, J. Intelligent example selection for learning systems. In press.
5. Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning 1*, 81-106.
6. Rendell, L., & Cho, H. (1990). Empirical learning as a function of concept character. *Machine Learning 5*, 267-298.
7. Schaefer, H. H. (1971). *Topological vector spaces.* New York: Springer-Verlag.
8. Winston, P. H. (1975). Learning structural descriptions from examples. In P. H. Winston (Ed.), *The psychology of computer vision*. New York: McGraw-Hill.