Best-Case Results for Nearest Neighbor Learning

Steven Salzberg, Arthur Delcher, David Heath, and Simon Kasif

Abstract— **In this paper we propose a theoretical model for analysis of classification methods, in which the teacher knows the classification algorithm and chooses examples in the** *best* **way possible. We apply this model using the nearest-neighbor learning algorithm, and develop upper and lower bounds on sample complexity for several different concept classes. For some concept classes, the sample complexity turns out to be exponential even using this best-case model, which implies that the concept class is inherently difficult for the nearest-neighbor algorithm. We identify several geometric properties that make learning certain concepts relatively easy. Finally we discuss the relation of our work to helpful teacher models, its application to decision-tree learning algorithms, and some of its implications for current experimental work.**

Keywords— **machine learning, nearest-neighbor, geometric concepts.**

I. Introduction

Since their introduction in the 1950's, nearest-neighbor (NN) classifiers have been shown very effective in practice for many problem domains. While the framework of NN was originally proposed as a tool for pattern recognition, it is widely used in many applications of intelligent systems such as speech processing, medical diagnosis, molecular biology, and others (e.g., [1], [2], [3], [4]). In addition to these practical applications of NN, researchers have developed many important theoretical results showing bounds on the error rate that nearest-neighbor classifiers obtain [5], [6]. Most of this previous research has focussed on analysis of the performance of NN in the limit (i.e., convergence properties) or in the average case; in this paper, we emphasize best-case bounds instead; i.e., results that give the minimum number of training examples required to learn a concept exactly. Exact learning in this paper means that a classifier will assign the correct label to all possible positive and negative instances of a concept.

In this paper, we introduce a model that is based on geometric tools rather than statistical ones, and show how this model can be used to prove upper and lower bounds on the minimum number of examples required by an algorithm in order to learn a concept exactly. Geometric analysis has been successfully used in the past to understand the behavior of learning systems [7], [8], though not in the context of a best-case model such as that introduced here.

The work of S.L. Salzberg was supported in part by the National Science Foundation under Grants IRI-9116843 and IRI-9223591. The work of and S. Kasif and D. Heath was supported in part by the Air Force Office of Scientific Research under Grant AFOSR-89-0151.

S.L. Salzberg, D. Heath, and S. Kasif are with the Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218. E-mail: {lastname}@cs.jhu.edu.

A.L. Delcher is with Computer Science Department, Loyola College in Maryland, Baltimore, MD 21210.

The practical motivation for such a model can be understood best through an illustration. Consider an automated handwriting recognition system based on the nearest-neighbor algorithm that must be trained by the user. Consider the problem of training it to recognize the character T, which might be described by eight features: the endpoints of the horizontal line segment (x_1, y_1) and (x_2, y_2), and the endpoints of the vertical line segment (x_3, y_3) and (x_4, y_4). The definition of the character T may be expressed using inequalities such as:

$$|(y_1 - y_2)| < \epsilon_1$$
$$|(x_3 - x_4)| < \epsilon_2$$
$$|(y_1 - y_3)| < \epsilon_3$$
$$|(x_1 + x_2)/2 - x_3| < \epsilon_4$$

Another way to represent this concept is as a geometric region in $\Re^8$. However, although the user knows this concept, his only method of training the character recognition system is by providing positive and negative examples of the concept T. Naturally, the user prefers to minimize the time spent in training. The system's developer would like to be able to tell the user as much as possible about how many and what kinds of examples to provide to the system. Our question here is, how many examples must the user provide before the system "learns" the concept exactly? As it turns out, the answers to these questions depend on geometric properties of the concept being learned.

Our goal in this paper is to provide a model that may ultimately help to give precise answers to such questions. We have chosen to address the issue from the point of view of best-case models because we are assuming that the algorithm will be trained (as in this example) by someone who wants to help it learn; i.e., a helpful teacher. In the model we define below, the teacher knows the algorithm and tries to present a minimum number of training examples to the algorithm. Lower bounds obtained with this very optimistic model provide absolute limits on the minimum number of examples required for an algorithm to learn a concept. We apply our model principally to the nearest-neighbor learning algorithm, which is a well-known and practically useful algorithm. We will show that even with this best-case model, there are still some interesting concept classes that are very hard to learn.

In particular, we consider the following questions:

- What is the minimum number of examples needed before an algorithm will correctly classify all possible examples of a concept?
- What is the minimal size representation for a given concept, and how does this relate to the minimum number of examples needed for training an algorithm?
- What geometric properties make a concept class diffi-

cult to learn in our model?
- How stable is a minimal representation; i.e., is it robust when the algorithm is provided with additional examples?

II. The best-case model

We answer these and other questions using a best-case model for analyzing sample complexity. A learning algorithm takes a **training set** S of examples, $X_1, X_2, \ldots, X_n$, where each example consists of a set of d features plus a category label. The algorithm must use the examples to build a representation with which it will classify a **test set** T of new examples. Thus we use the term "learning algorithm" interchangeably with "classification algorithm."[1]

A *concept* is a mapping from the set of all examples in a domain to a set of labels. A *concept class* is a set of concepts. For example, in the domain defined by the unit square, the partitioning defined by the line $x = 0.5$ might be a concept where any (x, y) in which $x \leq 0.5$ is labelled 0, and $x > 0.5$ is labelled 1. An example of a concept class in this domain might be the set of all partitionings defined by lines of the form $x = r$, where $0 \leq r \leq 1$. If the examples s_i are contained in $\Re^d$, then an equivalent term for concept is *decision boundary*, where the boundary includes all surfaces separating the classes.

We will say that a learner has learned a concept *exactly* if it correctly classifies all possible examples. The distribution from which the new examples in T are taken can be different from the distribution used for S. Because our model is not statistical, the results are independent of any assumptions about these distributions.

In our model, the learning algorithm and the concept are fixed and known to the teacher. The teacher then chooses examples for S in the *best* way possible. By "best" we mean the learning algorithm requires a minimum number of training examples to learn a concept. The term *sample complexity* refers to the number of examples required for learning. The teacher is able to compute a representation of the concept in the same language used by the learner. We have also called this the "helpful teacher" model [9], [10].

We will show below that even when using this best-case model, some concept classes still have exponential sample complexity for the nearest-neighbor algorithm. Because this is a best-case analysis, we know these classes are hard to learn—the sample complexity may be worse when one considers average-case or worst-case scenarios. On the other hand, we have also found that, for many concepts, surprisingly little information is required to teach a correct representation. In these cases, though, the representation used by the learner may be "unstable": if the learner stores additional examples, its representation will change, and it may no longer be correct.

[1]These usages are standard in the machine learning and artificial intelligence literature, but may differ from the terms used in the pattern recognition community; thus they are explicitly defined here.

III. Nearest-neighbor algorithms

The *nearest-neighbor* algorithm has been a subject of both experimental and theoretical studies for many years (see, e.g., [11], [5], [6], [12]). Experimental studies have shown that the predictive accuracy of nearest-neighbor algorithms is comparable to that of decision trees, rule learning systems, and neural net learning algorithms on many practical tasks (see, e.g., [2], [13], among many others). In addition, it has long been known that the probability of error of the nearest neighbor rule is bounded above by twice the (optimal) Bayes probability of error [5].

The basic nearest-neighbor algorithm (henceforth NN) can be summarized as follows. An example is a vector of real numbers plus a label that represents the name of a category. Assuming a domain with d features, where all features have been normalized to the same range, the distance between two examples $X = (x_1, x_2, \ldots, x_d)$ and $Y = (y_1, y_2, \ldots, y_d)$ is simply Euclidean distance: $E(XY) = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$ The algorithm itself is very simple:

1. Store all training instances, plus their classifications.
2. For a new example X, compute the distance between X and each stored instance.
3. Let A be the stored instance with the minimum distance to X. Output the classification of A as the predicted classification of X.

We will present theorems below showing that algorithm NN can learn some concepts very efficiently using our best case model. One advantage of choosing the nearest-neighbor algorithm as a target of study is that applying our model is quite easy for this algorithm. The teacher simply selects examples at the best possible locations in the d-dimensional feature space, and provides those examples to the algorithm. With other algorithms, particularly those that depend on the order of the inputs, finding the teacher's best strategies can be much more difficult. Because the nearest-neighbor algorithm does not depend on the order of the inputs, the teacher is saved from having to calculate the optimal order in which to present a set of training examples.

A. Related learning models

In the theoretical computer science community, Angluin [14] defined a different model of a helpful teacher, in which the learner is allowed to ask questions such as "is x a positive example?" She has produced a variety of results for learning regular expressions and context-free grammars. Our studies differ substantially from Angluin's work, since our teacher has more knowledge of the learner and much more control over the set of training examples. A more closely-related model is the one recently developed by Goldman and Kearns [15]. In their model, the teacher presents examples to the learner, who in turn is required to output its predictions on-line. Their teacher does not know the learning algorithm (unlike our model), but it does know the concept, and it is allowed to observe the predictions made by the learner. The learner is required to be consistent. The goal of the teacher is to select examples so that the learner will produce a correct concept while making a minimal number of on-line mistakes.

Our model is also related to the notion of "testability" introduced by Romanik [16]. In her model, a class is defined to be *testable* if, given a target object in the class and an error bound $\epsilon > 0$, a randomly generated set of test points will, with high probability, distinguish the target object from any other object in the class if the two objects differ by more than ϵ. This model includes a measure of *testing complexity* that is closely related to the sample complexity bounds produced using our best-case model. The notion of testability can be viewed as a converse of learnability. Inherent in this model is the error bound; there is not (as yet) a notion of exact testing that would correspond to exact learning.

A substantial body of research has been devoted to the notion of nearest-neighbor "editing," which involves removing examples from a training set S in order to reduce the amount of storage required for NN. In the editing (or filtering) model, the training set S is provided as input, and the learner attempts to find a subset of S that constitutes an equally good classifier. This differs from our model in the following respect: editing methods assume that the training set S is given as input, while our best-case model actually constructs the training set S; i.e., in our model the input is the concept boundary, and the output is a minimum size training set. Nearest-neighbor editing was introduced by Hart [17],and modified and extended by Swonger [18], Wilson [19], and Chang [20]. Ritter et al. [21] were the first to give an algorithm that finds a consistent subset of minimum size. Toussaint et al. [22] introduced the notion of using Voronoi boundaries for editing. Later they showed that when the points of S are in general position, their method finds a minimal-size consistent subset [23].

IV. Learning geometric concepts

A. Learning simple convex concepts

One of the simplest geometric concepts is a simple line or hyperplane, which translates into the learning problem of classifying examples from exactly two categories that are separated by a line, plane, or hyperplane. Although very simple, this concept class has proven very useful for a large number of real data sets, and is fundamental to classification techniques such as cluster analysis. Throughout this paper, we will say that an algorithm learns a geometric object when it learns a partitioning of space defined by that object. For example, when we write that an algorithm learns a polygon, we mean that the algorithm learns a partitioning of the plane in which the interior of the polygon is labelled with one class and the exterior is labelled with another.

We begin with some simple observations about best-case results for hyperplanes and other geometric objects. These observations are presented merely to illustrate the type of results we will be presenting in the next section. More details on these and additional observations can be found in Salzberg et al. [9], [10].

Observation 1: Under the best-case learning model, exactly two examples are required by algorithm NN to learn a concept defined by a hyperplane in any number of dimensions.

The strategy for the teacher here is obvious: choose any two points equidistant to the line (or hyperplane) such that the line defined by the two points is perpendicular to the original line. These two points alone will then correctly classify all future examples. A convex polygon is also very easy to learn in the best case:

Observation 2: If the concept boundary is a convex polytope (in 2 dimensions, a polygon) with n faces, then $n+1$ examples are necessary and sufficient to learn the concept exactly.

The teacher in this case chooses any point inside the polytope as a positive example of the concept. It then simply reflects that point through each of the n faces (or extensions of the faces), generating n new points labelled as negative examples. These $n+1$ points now define the polytope exactly. Our introductory problem of training a system to recognize the letter "T" illustrates this concept class; in particular, after normalizing coordinates, the concept we described there is a 10-sided convex object in $\Re^6$ and can be learned from 11 examples (1 positive and 10 negative).

B. Learning complex geometric concepts

Next we consider the sample complexity of learning a number of more complex geometric concepts. By virtue of their complexity, these objects probably have the ability to capture many real-world concepts, including those with complex boundaries. Interestingly, some of these concept classes may require a large number of examples, even in the best case. Unless other learning algorithms (besides NN) turn out to be significantly more powerful, these concept classes will remain "hard" for learning.

To provide some tools for learning complex objects, we will first need a few techniques for learning triangles. These observations will be useful in the proofs that follow. First recall the definition of a Voronoi diagram: given a set of points S, a point is in the Voronoi diagram of S if it is equidistant to its two nearest neighbors in S. In other words, if a point p has two or more nearest neighbors in S, then p is in the Voronoi diagram of S. (Distance can be measured in different ways, but Euclidean distance is the most common method.) Thus, the partitioning of space induced by the NN algorithm is, by definition, a Voronoi diagram [24].

Observation 3: Suppose we are learning a concept with a triangular boundary. Suppose also that the triangle is not obtuse; *i.e.*, no angle is greater than 90°. Then exactly nine points are sufficient for Algorithm NN to learn the triangle, using the following strategy.

If l is the length of the shortest side of the triangle, and h is the shortest altitude of the triangle (i.e., the minimum distance of any vertex to its opposite side), let $\delta = 1/2 \min\{l, h\}$. Choose a distance $\delta_1 < \delta$, and place a point this distance from each corner of the triangle in its interior, such that the point is equidistant from the two

Fig. 1. Learning an acute triangle

adjacent sides. Then reflect each corner point through the two adjacent sides. The resulting Voronoi diagram defined by the nine points contains the triangle as a subset. Figure 1 illustrates such a diagram. In the figure, the solid lines are sides of the triangle, and dotted lines are other edges of the Voronoi diagram. The intersections of the Voronoi diagram occur on the vertices, edges, and the interior of the triangle. None of the Voronoi intersections occur outside the triangle. This observation also applies to d-dimensional objects in which all angles between adjacent faces (e.g., dihedral angles in 3-D) are less than or equal to 90°.

Of course, a triangle can be learned with just four points (by Observation 2) rather than nine, but the technique depicted in Figure 1 can be extended to triangulations of the plane with convex boundaries and non-obtuse triangles. For our purposes, a triangulation is a partitioning of the plane with a polygonal boundary in which all interior regions (w.r.t. the boundary) are triangles. When more than one triangle shares a single vertex, then we can use a pair of points around each edge incident to each vertex. This pair of points would be placed a distance $\delta_1 < \delta$ from the vertex, and equidistant on either side of the edge at a distance less than $\delta_1 \sin^{-1} \frac{\alpha}{2}$ from the edge, where α is the smallest angle at the vertex.

Observation 4: For any triangulation in which all triangles are non-obtuse (*i.e.*, either right or acute triangles), a teacher can select points around every vertex of every triangle, using the strategy just described. These points allow algorithm NN to learn the triangulation exactly. The boundary of the triangulation must be convex. If the triangulation contains n vertices, this strategy requires $O(n)$ points.

Note that Observation 4 does not hold when the triangulation contains obtuse triangles. Consider Figure 2, in which an obtuse triangle is shown attached to an acute one. If the training examples are placed around each vertex, then the Voronoi diagram created will not correctly demarcate the edge shared by the triangles. Part of the Voronoi diagram is shown as dotted lines in the figure, and the resulting mis-classified area is shaded.

C. Triangulating with nonobtuse triangles

Consider the set of concepts defined by non-convex simple polygons (*i.e.*, polygons that are not self-intersecting). This very general concept class is, it turns out, much harder to learn than that of convex polygons. It would be desirable to use the observations above to select points for any polygon; *e.g.*, by first triangulating the polygon and then learning the triangulation. However, Figure 2 shows the problem that arises when the triangulation contains obtuse triangles. If we could encase a polygon P in its convex hull and triangulate the resulting figure using only nonobtuse triangles, then by Observation 4, algorithm NN could learn any concept in the class of non-convex polygons using $O(n)$ examples. It turns out that this triangulation problem is quite hard, even in 2-D, and our best result for learning non-convex polygons uses another method, described in Section IV-E.

The problem of triangulating polygons with nonobtuse triangles has until recently been an open problem in computational geometry. Baker *et al.* [25] produced the first solution, which used a relatively large number of triangles (the precise bound was not given). Bern and Eppstein [26] reported a solution in two dimensions that uses $O(n^2)$ triangles for an n-sided polygon, and later Bern et al. [27] improved this result to $O(n)$ nonobtuse triangles. Their method makes some progress towards solving our best-case learning problem, although a complete solution requires tri-

Fig. 2. Problem with an obtuse triangle

angulating the shape formed by a polygon plus its convex hull.

To see why triangulating the interior of a non-convex polygon with nonobtuse triangles is not sufficient, consider the example in Figure 6 below. (This figure is used again later, in Section IV-D.) Any triangulation of the interior, even one using nonobtuse triangles, will have vertices that coincide with the corners of the top of each vertical "finger" in the figure. The point placement strategy of Observation 4 will not work because it will place exterior points just outside these vertices, and those points will be closer to the interior of the object's horizontal "fingers" than any interior point. To define this shape for algorithm NN, one needs first to create the convex hull containing all the vertices in the figure, and then to produce a triangulation that contains all the interior edges of the figure as well as the boundary. If this triangulation of the interior and exterior of the polygon contains only nonobtuse triangles, then we can define the object using the strategy of Observation 4.

D. Rectilinear solids

Before considering learning the general class of all convex polygons (i.e., learning partionings of space defined by polygonal boundaries), we can first consider learning a subclass that is somewhat easier. *Rectilinear polygons* are polygons in which all the sides are parallel to the axes. Figure 3 shows a rectilinear polygon in two dimensions. We will consider two-dimensional objects first, but our results here extend easily to d dimensions. We give both upper and lower bounds for this concept class. Rectilinear concepts play an important role in our analysis because they can be used to learn general geometric concepts with arbitrary precision.

Theorem 1: Any rectilinear polygon with n edges in two dimensions can be learned in the best-case model with n^2 examples.

Proof: Draw the smallest possible rectangle around the rectilinear polygon. For each edge of the polygon, extend the edge in both directions until it meets the containing rectangle. The resulting grid will contain at most $n^2/4$ vertices. Since every vertex of the grid has degree 4, we can place 4 points symmetrically around each vertex (arbitrarily close to the vertex). This set of points will define a Voronoi diagram that contains the grid, and therefore contains the rectilinear polygon. With appropriate labelling of points, the learning is complete, having used n^2 examples. Figure 4 illustrates the grid and the placement of points for the rectilinear polygon from Figure 3. For visual clarity, a single dot appears in the figure at every vertex around which points need to be placed. Note that points only need to be placed around grid vertices where the grid coincides with one of the vertices or edges of the original polygon. ■

Any set of non-overlapping rectilinear objects, including a rectangular tiling of the plane, can be learned with the same strategy, and the same sample complexity. Note that for particular rectilinear objects (*e.g.*, grids), the complexity may be linear. We also can improve the algorithm slightly by using a smaller object to enclose it—instead of a convex rectangle, we can use a "rectilinearly convex" object.[2]

Suppose next that we wish to learn a concept defined by an arbitrary partitioning of the plane into polygons; for example, a triangulation of the plane. For such a concept, we may wish to allow each region of the partitioning to have a different class label. Despite the apparent complexity of this concept class, we can use the rectilinear concepts just described to help learn these concepts. The idea behind this solution is that instead of learning the given partitioning, we will learn a very similar concept.

[2] A polygon is rectilinearly convex iff every axis-parallel line segment connecting two points on the polygon is contained in the interior of the polygon.

Fig. 4. Learning a rectilinear polygon

First we need to define what we mean by a "similar" concept. Let the input concept C be defined as a polygonal tesselation (a planar partitioning that uses polygons) on the unit square. Next we define a measure ϵ that represents the area of the symmetric difference between a new concept C' and C. Included in ϵ is the area that belongs to C and not C' plus the area that belongs to C' and not C. Thus we use ϵ to represent the difference between the true concept and the learned concept; i.e., these are the regions of the unit square that would be misclassified. When ϵ is very small, we say that the concepts C and C' are ϵ-similar. Another way to view this is that C' is an approximation on a grid to C.

Corollary 1.1: Given any polygonal tesselation C in the unit square, algorithm NN can learn exactly a concept C' that differs from C by at most ϵ, using only kn^2/ϵ points, where n is the number of edges in the tesselation and k is a constant.

Proof: Given a tesselation, we transform it by laying it on a $g \times g$ grid. We create a new rectilinear tesselation from the four edges of every grid cell that intersects the edges of the original tesselation. An example is shown in Figure 5. In the figure, the grid squares that become part of the rectilinear tesselation have been darkened. The original tesselation can intersect no more than $2ng$ grid cells. Because the area of each grid square is $1/g^2$ (the grid is in the unit square), and at most $1/2$ of each square is misclassified, the total area in which the rectilinear tesselation differs from the original tesselation is less than $(2ng)(1/g^2)(1/2) = n/g$. Therefore, if we can learn the rectilinear tesselation exactly and use it for classification, then the area in which errors can occur will be no more than

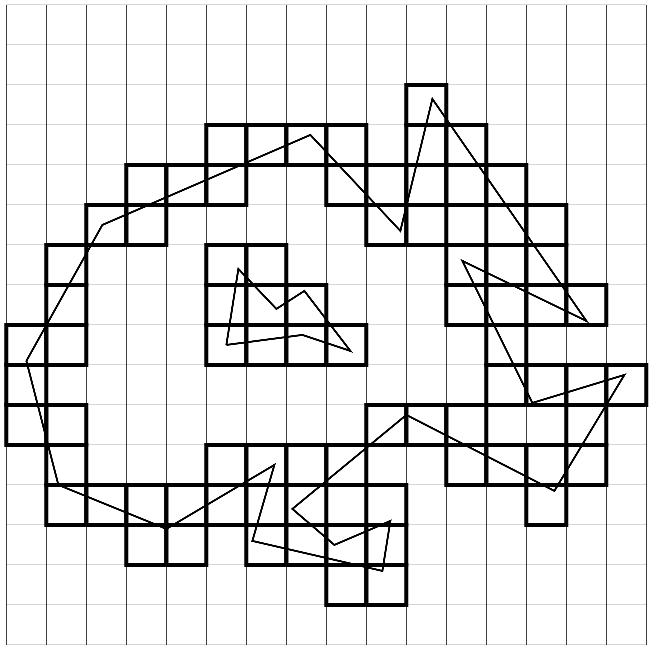

Fig. 5. Rectilinear concept that is similar to a polygonal tesselation

n/g. In order to guarantee that the error area $n/g < \epsilon$, we can choose $g > n/\epsilon$. This implies the rectilinear tesselation will have no more than $(2ng)(4) = 8n^2/\epsilon$ edges.

We can use the rectilinear polygon algorithm to teach the transformed tesselation to algorithm NN. Generally, given e edges, this algorithm would generate e^2 points. In particular, this would mean that the rectilinear tesselation we create would require n^4/ϵ^2 points. Because the grid is a square grid, however, extending each line segment in the tesselation does not generate any new vertices. Thus the rectilinear teaching algorithm only uses a linear number of points, and only $8n^2/\epsilon$ points are necessary to learn a concept whose symmetric difference with the original tesselation is less than ϵ. ■

This construction extends easily to higher dimensions.

Corollary 1.2: Over the unit d-cube in d dimensions, given a target concept C, the number of points used by this algorithm to learn a similar concept C' is kdn^d/ϵ^{d-1}, where k is a constant and ϵ now represents the volume of the symmetric difference between C and C'.

More details can be found in [9]. Note also that, in any number of dimensions, any point that is classified incorrectly using this technique is within a very small distance—no greater than the diagonal length of a grid cell—from an edge of the original tesselation. Thus the similar concept C' not only differs by a very small amount (i.e., ϵ) from the target concept C, but its boundary is always close to the boundary of C.

For the rectilinear concept class, we can also show a lower bound for the best-case model.

Theorem 2: At least $n^2/32$ examples are required for learning some rectilinear polygons in the best-case model, where n is the number of sides of the polygon.

Proof: We prove this theorem by constructing a rectilinear object that requires $n^2/32$ points to define it. See Figure 6 for an example of such an object.[3] The object shown has approximately $n/8$ vertical "fingers" and $n/8$ horizontal "fingers" (since each finger uses four sides of the polygon). These fingers can be made arbitrarily thin and arbitrarily long. By placing the vertical fingers far apart, as we have illustrated in the figure, we make it impossible to define the topmost (horizontal) edge of each finger without interfering with *all* of the horizontal fingers. Thus a point placed above the topmost edge of each vertical finger will propagate through all of the horizontal fingers, requiring us to draw a pair of points around each of the edges above it. There are $n/4$ such edges in the $n/8$ horizontal fingers, which means that each vertical finger will require at least $n/4$ new points for learning (some of the points used to define horizontal edges may be shared). Therefore the total number of points required is at least $(n/8)(n/4) = n^2/32$. ■

It is worth noting that this lower bound also implies a similar lower bound on the number of nonobtuse triangles required for triangulating the shape formed by a polygon plus its convex hull. Although the interior of a non-convex polygon can be triangulated using just $O(n)$ nonobtuse triangles [27], the interior plus exterior will require $\Omega(n^2)$ nonobtuse triangles for some polygons.

The theorems for exact learning of rectilinear objects both extend quite straightforwardly into d dimensions. Let a *rectilinear polytope* be a d-dimensional object for which every face (or facet) of the object is parallel to one of the axes.

Corollary 2.1: No more than $(2n/d)^d$ examples are required in the best case for exact learning of rectilinear polytopes with n faces (or facets) in d dimensions.

Proof: The algorithm is basically identical to the algorithm given for Theorem 1. We simply enclose the rec-

[3]The idea of this construction was proposed by Michael Goodrich.

Fig. 6. A rectilinear polygon that requires at least $n^2/32$ examples.

tilinear polytope in a hyperrectangle, and extend all the faces of the polytope until they meet the sides of the enclosing hyperrectangle. The resulting grid contains at most $(n/d)^d$ vertices. Since every vertex is just the meeting point of 2^d hyperrectangles, we can place 2^d points around each vertex, arbitrarily close to the vertex. These points will define a Voronoi diagram that contains the grid and therefore contains the rectilinear polytope. The total number of such points is $(2n/d)^d$. Note that we can reduce this number somewhat by observing that points only need to be placed around vertices that coincide with faces in the original polytope. ■

Corollary 2.2: $(\frac{kn}{d})^d$ examples are required to learn some rectilinear polytopes with n faces in d dimensions, where k is a small constant.

Proof: This lower bound can be proven by construction, as was done above. In three dimensions, we place $kn/3$ planar slices above the object shown in Figure 6. (k is a constant, $k < 1$.) If each slice is very thin and all the slices are very close together, then all of the points required to define Figure 6 in 2-D will propagate through the $kn/3$ slices. In this manner we can construct a $(\frac{kn}{d})^3$ lower bound in 3-D. Similarly, we can extend the construction to any number of dimensions. ■

E. Learning polygonal partitionings

The class of simple non-convex polygons is much harder to learn than rectilinear polygons. In fact, it appears to be as hard as learning arbitrary triangulations of the plane. Triangulations are at least as hard as non-convex polygons, in the sense that we always can triangulate an n-sided polygon by adding a linear number of sides. The proof that non-convex polygons are as hard as triangulations is an open problem.

Because nearest-neighbor partitions are by definition Voronoi diagrams, the problem we would like to solve is equivalent to the following Minimum Voronoi Cover problem: Given a planar tesselation, what is the smallest Voronoi diagram that contains that tesselation as a subgraph? Heath [28] showed that for arbitrary planar polygonal tesselations this problem is $\mathcal{NP}$-complete.

E.1 Triangulations of the plane

We now turn to the class of triangulations. A triangulation is a tesselation of some region into triangles, where the boundary of the triangulation may be any simple polygon. This class is very large, subsuming all geometric concept classes in which the component concepts have straight edges.

Our best current result for exact learning of triangulations in the best-case model requires a number of points that is proportional to the largest aspect ratio (the altitude with respect to the longest edge) of any triangle. This ratio can be arbitrarily large; however, Corollary 1.1 above gives an algorithm that will use kn^2/ϵ examples to learn a very similar concept to any polygonal tesselation, where n is the number of edges and ϵ is the total area of the difference between the learned concept and the target (input) concept. Thus if the aspect ratio is too large, the preferred approach might be to learn a very similar concept instead.

First, recall our earlier observation, that if all triangles are acute, algorithm NN in the best case can learn a triangulation with $O(n)$ examples.

Theorem 3: Any triangulation of the plane with n edges can be learned with $\frac{2L}{h} + n$ points in the best-case model, where L is the total length of all edges of the triangulation, and h is the minimum altitude of all triangles.
Note that even if all edges are very long, h might be extremely small if the triangulation contains very long, narrow triangles.

Proof: We give an algorithm that uses at most $\frac{2L}{h} + n$ points to define any triangulation. Beginning simultaneously at both ends of every edge in the triangulation, place a pair of points symmetrically on either side of the edge as close as possible to the edge itself and to each end of the edge. (The distance between each point and the edge will depend on how many bits of precision are used; as long as this distance is much less than h, the resulting Voronoi diagram will contain the edge.) Next, moving from the ends towards the middle of each edge, place a pair of points at intervals of length h symmetrically on either side of the edges. Stop when the midpoint of the edge is reached. The points placed near the midpoint may be closer together than h, but this detail will not affect the correctness of the algorithm. If the pairs of points placed near the midpoint are at a distance greater than h from each other, place one more pair of points around the midpoint itself.

This strategy correctly defines every edge of the triangulation. To see this, consider the triangle shown in Figure 7. The edges of the triangle are labelled e_1, e_2, e_3 and the vertices are v_{12}, v_{23}, v_{13}. Consider a point x on an edge e_1 between two pairs of points, p_1 and p_2. Without loss of generality, assume that x is closer to the pair p_1. By definition, x is equidistant to each of the points in p_1. Therefore, it will have a Voronoi diagram edge running through it unless some other point chosen by the teacher is closer. The distance from x to the points in p_1 is at most $\frac{h}{2}$. Since h is the shortest altitude (or edge) of the triangulation, there cannot be a vertex closer to x than the pair of points p_1. Likewise, if the closest edge, e_3 (as shown in Figure 7) is within h of point x, then it must share an endpoint with e_1 (*i.e.*, it is part of the same triangle). In the worst case, suppose that the closest point along e_3 is one of the points placed by our Helpful-Teacher strategy. Call that point q.

If q is closer to v_{13} than to v_{23}, then it must have been placed the same distance from v_{13} as was p_2. Simple geometry then shows that p_2 will be closer to x than will q. However, if q is closer to edge v_{23} (as shown in the figure), then the vertical distance from q to e_1 is more than $\frac{h}{2}$ since the altitude at v_{23} is at least h. Therefore either p_1 or p_2 must still be closer to x. Thus, finally, we can be sure that x does have a Voronoi diagram edge running through it which is precisely collinear with the edge e_1.

Clearly, the number of points placed on any one edge by this strategy is no greater than $\lceil \frac{2L_e}{h} \rceil$, where L_e is the length of the edge. Thus the total number of points used by the teacher in the best case is bounded by $\frac{2L}{h} + n$, where n is the number of edges. ■

Note that if all triangles are nonobtuse, the strategy of Observation 4 above is much more efficient than the one just described.

To learn a polygon-shaped concept using this method, one would first triangulate the polygon in such a way as to maximize h. Bern et al. [29] have developed an algorithm that can construct a triangulation of a polygon in $O(n^2 \log n)$ time that maximizes the minimum triangle altitude. Although this result does not give a lower bound for h, it does provide the maximum possible value for h. However, this method does not necessarily minimize $\frac{L}{h}$, and in fact the problems of minimizing L and minimizing $\frac{L}{h}$ remain open problems in computational geometry, even for two dimensions.

For many triangulations, the strategy of Theorem 3 will work well, but since L can be exponentially larger than h, the number of examples required in the worst case is very large. (This assumes finite precision on the inputs; if one allows infinite precision, L/h is unbounded). To handle such cases, we would use the strategy for learning a rectilinear concept that is ϵ-similar to the triangulations, presented in Corollary 1.1. If an area no larger than ϵ is allowed to be misclassified, this strategy requires kn^2/ϵ edges, regardless of the ratio L/h.

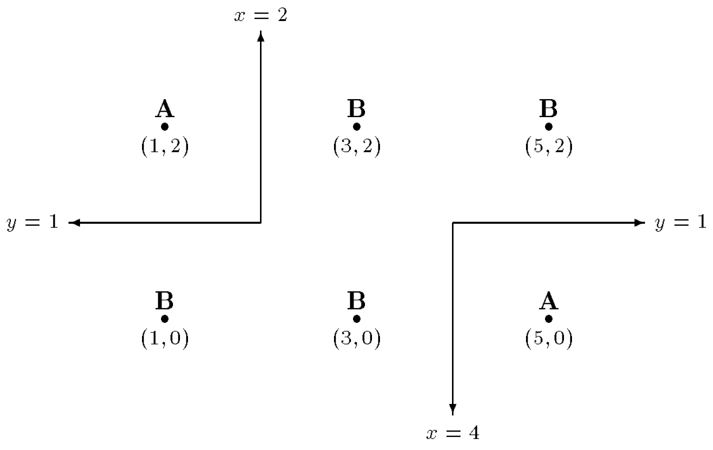

Fig. 8. Target concept for learning by C4.5

Corollary 3.1: In d dimensions, the algorithm of Theorem 3 suffices to define a simplicial decomposition (a multidimensional triangulation), with two modifications. First, pairs of points have to surround each edge in each dimension. (*E.g.*, in three dimensions, when edges are the intersection of two faces, each edge has two pairs of points, rather than one pair, at intervals of length h along the edge.) Second, pairs of points need to be placed on each side of every face (or hyperface) so that the distance from any location on the hyperface to a point is always less than $h/2$. If A is the total area of all faces of the decomposition, and n is the number of edges, then the number of examples required in d dimensions is at most $(A/h)^{d-1} + n^{d-1}$.

V. Decision trees and minimal representations

Although we have chosen to focus on the nearest-neighbor method in this paper, the best-case model applies to any learning algorithm. In all of our results using the nearest-neighbor learning algorithm, the examples chosen for the best-case model form a minimal representation of the concept—i.e., with respect to the Voronoi diagrams formed by the examples, if any example is deleted from the training set, the Voronoi diagram will be altered so that the decision boundary is incorrect. This observation naturally leads to the question, is the best-case model doing anything other than looking for the minimal representation of a concept? Although the answer to this question is yes in the case of nearest-neighbor, the answer is no for many other learning algorithms. This is best illustrated with a brief example using a decision tree algorithm. (More details can be found in [9].)

Consider Figure 8. The concept here is a two-category problem, where the categories are labelled A and B in the figure. Category A occurs in two disjoint regions of the 2-dimensional plane, in the upper left and lower right quadrants. The horizontal boundaries of the two A regions are

Fig. 7. Learning a very narrow triangle

collinear, but the vertical boundaries are not. In order to teach this concept accurately, a teacher must provide at least six points to C4.5, two in category A and four in category B. Those points are indicated in the figure.

If we give these six points to the decision tree program C4.5 [30], it will construct a tree that correctly partitions the space. The tree itself, however, will be larger (by one node) than the minimal size tree. In order to force C4.5 to construct the minimal size tree, at least eight examples must be provided, rather than six. Thus the best-case model for learning will produce the larger tree, because the number of examples required by the learning algorithm is smaller.

This example illustrates clearly that the teacher in our model is not simply looking for minimal representations. Instead, our helpful teacher looks for the smallest set of examples that will allow the learner to build a correct representation. Although the problem of finding minimal representations often is related to the problem of teaching efficiently, it is not the same.

VI. Conclusion

The results presented in this paper show that, for a given algorithm, it is possible to demonstrate lower and upper bounds for many different concept classes using a best-case model that can also be viewed as a helpful teacher. Our results are summarized in Table I. These bounds are useful for a variety of reasons. First is that our analysis shows how geometric properties such as the convexity and aspect ratio of concept classes are related to complexity of learning, at least for the NN algorithm. In addition, the lower bounds provide an *absolute* limit on the number of examples required for learning. Such limits can tell us in advance how many examples must be provided if we expect our algorithm to learn a concept.

Limits derived using the best-case model may provide an instructive basis for comparison between learning algorithms. We now have theoretical models giving worst-case, average-case, and best-case analyses. Although most of the studies here involve only the nearest neighbor algorithm, this is only a first step. Next steps include finding best-case bounds for other algorithms and helpful teaching strategies for non-geometric concepts such as Boolean formulas.

Some concept classes appear to be "hard" for the nearest-neighbor algorithm, in that the number of examples required for learning, even in the best case, is exponential in the number of dimensions d. In particular, the class of non-convex simple polygons behaves this way. We have shown (by construction) a lower bound on a subclass of non-convex polygons—rectilinear polygons—that exhibits this exponential behavior. This result points out a weakness in the nearest-neighbor algorithm itself; thus, if another learning algorithm can be shown to learn this class more efficiently, that algorithm could justifiably be described as more powerful than nearest neighbor.

TABLE I
Best-case bounds for the nearest-neighbor algorithm.

Concept boundary	*Lower Bound*	*Upper Bound*
Hyperplane	2	2
n-face convex polytope	$n+1$	$n+1$
n concentric rectangles	$4n+1$	$16n-4$
Nonobtuse n-triangulation with convex boundary	$\Omega(n)$	$O(n)$
n-sided, d-dimensional rectilinear polytope	$(\frac{kn}{d})^d$	$(2n/d)^d$
2-D triangulation, n sides, total length of sides L, minimum altitude h	-	$\frac{2L}{h}+n$
Concept that differs by volume ϵ from polygonal tesselation with n faces, d dimensions	-	kdn^d/ϵ^{d-1}

We note that in the context of nearest-neighbor learning, finding the minimum number of examples is equivalent to finding the minimum size Voronoi diagram that contains the decision boundary. For other learning methods, providing the minimum number of examples may generate something larger than the minimum representation. For example, we have shown that a strategy to produce the minimum size decision tree for an algorithm such as

C4.5 sometimes requires more examples than our best-case model. Although the issue of minimal representations is certainly related to our model, our main interest is in determining the minimum number of examples and in strategies for generating these examples.

We should also take note of the following: it is sometimes the case that providing *more* examples to the nearest-neighbor algorithm reduces its accuracy. This is a partial answer to the question we asked at the beginning of this paper, namely, what is the effect on a learned concept of providing the learner with additional examples? With nearest neighbor, the simplest example of this effect occurs for learning a plane. Only two examples need to be provided; however, if one provides additional examples, the concept definition will shift and become inaccurate unless those examples are given in pairs that are reflections of each other through the plane. Thus, although best-case results for sample complexity may be vastly smaller than worst-case results, the representation created by nearest-neighbor might be unstable in the best case. An interesting question for further study is how many examples are required to produce stable representations; *i.e.*, representations to which one can add more examples with only a limited loss in accuracy.

Acknowledgements:

Thanks to Michael Goodrich, John Johnstone, Gregory Sullivan, and Godfried Toussaint for their insightful comments and discussions on the ideas presented in this paper. Thanks also to the anonymous reviewers for additional helpful comments.

References

[1] B.V. Dasarathy, *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, IEEE Computer Society Press, Los Alamitos, California, 1991.

[2] D. Aha, D. Kibler, and M. Albert, "Instance-based learning algorithms", *Machine Learning*, vol. 6, no. 1, 1991.

[3] S. Salzberg, *Learning with Nested Generalized Exemplars*, Kluwer Academic Publishers, Norwell, MA, 1990.

[4] S. Cost and S. Salzberg, "A weighted nearest neighbor algorithm for learning with symbolic features", *Machine Learning*, vol. 10, no. 1, pp. 57–78, 1993.

[5] T. Cover and P. Hart, "Nearest neighbor pattern classification", *IEEE Transactions on Information Theory*, vol. 13, pp. 21–27, 1967.

[6] P.A. Devijver, "An overview of asymptotic properties of nearest neighbor rules", in *Pattern Recognition in Practice*. 1980, pp. 343–350, Elsevier Science Publishers B.V.

[7] T. Cover, "Geometric and statistical properties of systems of linear inequalities", *IEEE Transactions on Computers*, vol. 14, pp. 326–334, 1965.

[8] M. Minsky and S. Papert, *Perceptrons*, MIT Press, Cambridge, MA, 1969.

[9] S. Salzberg, A. Delcher, D. Heath, and S. Kasif, "Learning with a helpful teacher", Tech. Rep. 90/14, Dept. of Computer Science, Johns Hopkins University, 1990 (revised 1992).

[10] S. Salzberg, A. Delcher, D. Heath, and S. Kasif, "Learning with a helpful teacher", in *Proc. of the Twelfth International Joint Conference on Artificial Intelligence*, Sydney, Australia, August 1991, pp. 705–711, Morgan Kaufmann.

[11] E. F. Fix and J. Hodges, "Discriminatory analysis: Small sample performance", Tech. Rep. Project 21-49-004, Report No. 11, USAF School of Aviation Medicine, Randolph Field, Texas, August 1952.

[12] L. Devroye, "Automatic pattern recognition : A study of the probability of error", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 4, pp. 530–543, 1988.

[13] S. Salzberg, "A nearest hyperrectangle learning method", *Machine Learning*, vol. 6, pp. 251–276, 1991.

[14] D. Angluin, "Learning regular sets from queries and counterexamples", *Information and Computation*, vol. 75, pp. 87–106, 1987.

[15] S. Goldman and M. Kearns, "On the complexity of teaching", in *Proc. of the Fourth Annual Workshop on Computational Learning Theory*, Santa Cruz, CA, August 1991, pp. 303–314, Morgan Kaufmann.

[16] K. Romanik and S. Salzberg, "Testing orthogonal shapes", in *Proc. of the 1992 Canadian Conference on Computational Geometry*, St. Johns, Newfoundland, Canada, August 1992, pp. 216–222.

[17] P. Hart, "The condensed nearest neighbor rule", *IEEE Transactions on Information Theory*, vol. 14, no. 3, May 1968.

[18] C. Swonger, "Sample set condensation for a condensed nearest neighbor decision rule for pattern recognition", in *Frontiers of Pattern Recognition*, S. Watanabe, Ed., pp. 511–519. Academic Press, 1972.

[19] D. Wilson, "Asymptotic properties of nearest neighbor rules using edited data", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 2, no. 3, pp. 408–421, July 1972.

[20] C.-L. Chang, "Finding prototypes for nearest neighbor classifiers", *IEEE Transactions on Computers*, vol. 23, no. 11, pp. 1179–1184, November 1974.

[21] G. Ritter, H. Woodruff, S. Lowry, and T. Isenhour, "An algorithm for a selective nearest neighbor decision rule", *IEEE Transactions on Information Theory*, vol. 21, no. 6, pp. 665–669, 1975.

[22] G. Toussaint, B. Bhattacharya, and R. Poulsen, "The application of voronoi diagrams to nonparametric decision rules", in *Computer Science and Statistics: Proc. of the 16th Symposium on the Interface*, L. Billard, Ed., New York, 1984, pp. 97–108, Elsevier Science Publishers.

[23] B. Bhattacharya, R. Poulsen, and G. Toussaint, "Application of proximity graphs to editing nearest neighbor decision rules", Tech. Rep. SOCS 92.19, School of Computer Science, McGill University, Montreal, December 1992.

[24] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.

[25] B. Baker, E. Grosse, and C. Rafferty, "Nonobtuse triangulation of polygons", *Discrete Computational Geometry*, vol. 3, pp. 147–168, 1988.

[26] M. Bern and D. Eppstein, "Polynomial-size nonobtuse triangulation of polygons", in *Proc. of the 7th Annual Symposium on Computational Geometry*, New York, 1991, pp. 342–350.

[27] M. Bern, S. Mitchell, and J. Ruppert, "Linear-size nonobtuse triangulation of polygons", in *Proc. 10th Annual ACM Symp. on Computational Geometry*, Stony Brook, New York, 1994, pp. 221–230.

[28] D. Heath, *A Geometric Framework for Machine Learning*, PhD thesis, Johns Hopkins University, 1992.

[29] M. Bern, H. Edelsbrunner, D. Eppstein, S. Mitchell, and S. Tan, "Edge insertion for optimal triangulations", in *LATIN '92: 1st Latin American Symposium on Theoretical Informatics*, I. Simon, Ed., Berlin, 1992, pp. 46–60, Springer-Verlag.

[30] J. R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, CA, 1992.

Steven Salzberg received the B.A. degree in English and the M.S. and M.Phil. degrees in Computer Science from Yale University in 1980, 1982, and 1984 respectively. He received the Ph.D. degree in computer science from Harvard University, Cambridge, Massachusetts in 1989. From 1985 to 1987, he was a research scientist with Applied Expert Systems of Cambridge, Massachusetts. In 1988 and 1989 he was a Research Associate at the Harvard Business School. Since 1989, he has been an Assistant Professor in the Department of Computer Science at Johns Hopkins University. His research interests include machine learning and computational biology.

Arthur L. Delcher is Associate Professor and Chairman of the Computer Science Department at Loyola College in Maryland. His research interests include parallel algorithms, artificial intelligence and probabilistic networks. Delcher received his PhD degree in Computer Science from Johns Hopkins University in 1987. He is a member of ACM and the IEEE Computer Society.

David Heath received the B.S. degree in electrical engineering from the California Institute of Technology, Pasadena, California, in 1987. He received the Ph.D. degree in computer science from Johns Hopkins University, Baltimore, Maryland, in 1992. Currently, he is an instructor with the Department of Radiology at Johns Hopkins University. His research interests include machine learning and medical image analysis.

Simon Kasif did his undergraduate and graduate studies in mathematics in Tel-Aviv University. He received the M.S. and Ph.D. degrees in Computer Science from the University of Maryland, College Park in 1983 and 1985 respectively. At the University of Maryland he was associated with the Computer Vision Laboratory and the Parallel Logic Programming Project. He was one of the primary designers and implementors of PRISM, the first parallel logic programming system that was actually implemented on a real multiprocessor.

Since 1985 Dr. Kasif has been a faculty member at Johns Hopkins University, where he is currently an associate professor in the Computer Science Department. During sabbatical leaves he also taught at the University of Maryland at College Park and Princeton University, and performed research at Tel-Aviv University, the Weizmann Institute of Science, and NEC Research. Dr. Kasif's contributions include design and analysis of sequential and parallel algorithms for structural matching, logical inference, constraint satisfaction networks, decision tree induction and probabilistic networks. He has authored over 50 papers in these areas. Dr. Kasif's current main interests include large scale constraint satisfaction problems, real-time probabilistic dynamic networks, machine learning, parallel computation and computational biology.