4

Linear Separability in Descent Procedures for Linear Classifiers

Mitra Basu and Tin Kam Ho

Summary. Determining linear separability is an important way to understand structures present in data. We review the behavior of several classical descent procedures for determining linear separability and training linear classifiers in the presence of linearly nonseparable input. We compare the adaptive procedures to linear programming methods using many pairwise discrimination problems from a public database. We found that the adaptive procedures have serious implementational problems that make them less preferable than linear programming.

4.1 Introduction

Usually classification approaches in pattern recognition appear to fall into two broad types: the parametric methods that often rely on an assumption that each class of vectors follows certain known forms of probability distribution, and the nonparametric methods that do not make any assumptions about the class distributions. When there are indications from the observations that the assumption of Gaussian distributions has to be abandoned, there are few alternatives left, so that one has to resort to nonparametric methods. We believe that if some knowledge about the class boundaries can be derived from the data, there are advantages in using a recognition method suited to such boundaries.

In practice, a classification problem is often presented with a finite training set assumed to be representative of the expected input. Important clues about the complexity of the problem can be obtained by studying the geometrical structures present in such a training set. Extracting important information from small amount of data is also the focus of a recent trend in statistical learning theory [32].

A geometrical property that is of fundamental importance is linear separability of the classes. Based on this property, a number of descriptors of the point set geometry can be constructed. Whether the input is linearly separable or nonseparable, in constructing a classifier, deriving a linear discriminant that is optimal in some sense is a useful first step. Linear discriminants can serve as building blocks for piecewise linear classifiers, or be used to separate points transformed into a higher dimensional space where they may become linearly separable.

The need to study the structures in data also arises from the study of neural networks. One of the most promising attributes of a neural network is its ability to learn by interacting with an information source. Learning in a neural network is achieved through an adaptive procedure, known as a *learning rule or algorithm*, whereby the weights of the network are incrementally adjusted so as to improve a predefined performance measure over time. There is usually an assumed architecture of the network and a desired mapping. Very often, the outputs of the mapping are specified only for some subset of the possible inputs, namely, the training set. One of the main issues in learning is the design of efficient learning algorithms with good generalization properties. Although generalization is not a mathematically well-defined term, it usually refers to the capability of the network to produce desirable responses to inputs that are not included in the training set. The existing learning rules can be broadly categorized into three main groups: (1) *supervised*, (2) *unsupervised*, and (3) *reinforced*. We focus our attention on *Supervised* learning where the network architecture consists of a single layer of neurons, which can represent all linear classifiers. The rules in this category can be viewed as *error-correction* or *gradient-descent* types. Gradient descent is an iterative method commonly used to search for a solution (the global minima) in a high-dimensional space that has a number of local minima. Specifically, given a criterion function, the search point can be incrementally improved at each iteration by moving in the direction of the negative of the gradient on the surface defined by this criterion function in the appropriate parameter space. It has been shown that all these rules can be derived as minimizers of a suitable criterion function [6], and the corresponding algorithms, also appropriately known as *descent algorithms*, for implementation are based on gradient-descent method.

Notations

Consider a two-class (ω_1, ω_2) problem. Let us assume that there are m sets of training pairs, namely, ($\mathbf{x}^1, d^1$), ($\mathbf{x}^2, d^2$),..., where $\mathbf{x}^j \in R^n$ is the jth input vector and $d^j \in \{-1, +1\}$, $j = 1, 2, ..., m$ is the desired output for the jth input vector. In a single unit neural network, the output y^j for an input vector $\mathbf{x}^j$ is computed as $y^j = sgn(\mathbf{w}^t\mathbf{x}^j - \theta)$, where $\mathbf{w}^t$ denotes the transpose of the (column) weight vector $\mathbf{w}$. Such a network is referred to as a *single-layer perceptron* or a linear classifier.

Remark 1: Without loss of generality we include the threshold value θ in the weight vector as its last element and increase the dimension of every input vector by augmenting it with an entry that equals 1. Thus the weight vector is $\mathbf{w}^t = [w_1, ..., w_n, \theta]$ and the input vector is ${\mathbf{x}^j}^t = [{x_1}^j, ..., {x_n}^j, 1]$, and the output of the perceptron can be written as $y^j = sgn(\sum_{k=1}^{n+1} w_k x_k^j) = sgn(\mathbf{w}^t\mathbf{x}^j)$.

From now on let us assume that the input and the weight vectors are already augmented, and let n be the augmented dimension; then the problem of learning can be defined as follows:

Proposition 4.1.1 *Given a set of input vectors* ($\mathbf{x}^1, ..., \mathbf{x}^m$)*,* $\mathbf{x}^i \in R^n$*, and a set of desired output values* $\{d^1, ..., d^m\}$*,* $d^i \in \{1, -1\}$*, find a weight vector* $\mathbf{w} \in R^n$ *such that* $sgn(\mathbf{w}^t\mathbf{x}^j) = y^j = d^j$ *for* $j = 1, ..., m$.

The goal is to determine a weight vector **w** such that the following conditions are satisfied:

$$\mathbf{w}^t\mathbf{x}^j > 0 \text{ if } d^j = +1,$$
$$\mathbf{w}^t\mathbf{x}^j < 0 \text{ if } d^j = -1. \tag{4.1}$$

The equation $\mathbf{w}^{*t}\mathbf{x} = 0$ defines a hyperplane in R^n. Therefore finding a solution vector $\mathbf{w}^*$ to this equation is equivalent to finding a separating hyperplane that correctly classifies all vectors $\mathbf{x}^j$, j=1,2,...,m. In other words, an algorithm must be designed to find a hyperplane $\mathbf{w}^{*t}\mathbf{x} = 0$ that partitions the input space into two distinct half-spaces, one containing all points $\mathbf{x}^j$ for which the desired output is +1 and the other containing all points $\mathbf{x}^j$ for which the desired output is -1. We reformulate this condition (4.1) to adopt the convention usually followed in the literature.

Remark 2: Define a vector $\mathbf{z}^j$:

$$\begin{cases} \mathbf{z}^j = +\mathbf{x}^j \text{ if } d^j = +1, \\ \mathbf{z}^j = -\mathbf{x}^j \text{ if } d^j = -1. \end{cases}$$

The data matrix **Z** is defined to be $[\mathbf{z}^1, \mathbf{z}^2, ..., \mathbf{z}^m]$. The output y^j for the modified input vector $\mathbf{z}^j$ is computed as $y^j = sgn(\mathbf{w}^t\mathbf{z}^j)$. The goal is to determine a weight vector **w** such that the following condition is satisfied:

$$\mathbf{w}^t\mathbf{z}^j > 0 \quad \text{for all } j. \tag{4.2}$$

Note that this simplification aids only in theoretical analysis. As far as the implementation is concerned, this does not alter the actual computation in a significant manner.

All learning algorithms[1] for neural networks are guaranteed to find a separating surface in a finite number of steps for *linearly separable classes*. A comprehensive review of standard learning algorithms and analysis of their behavior for linearly separable classes can be found in reference [34]. However, when it comes to *linearly nonseparable* cases, the behavior of these algorithms is not usually explored. In practice, most real-world problems are *assumed* to be nonlinear without any concrete evidence, and multilayer neural networks are used to address these problems. However, the use of a multilayer network is overkill for a problem that is linear. Moreover, effective use of a multilayer neural net involves a proper choice of the network architecture.

Backpropagation, the standard learning algorithm, is used to train a fixed topology multilayer neural net. In general, this approach works well only when the network architecture is chosen correctly. Too small a network may not be able to capture the characteristics of the training samples, and too large a size may lead to overfitting and poor generalization performance. Results are available to determine the *correct* size of a network. It has been shown that a single hidden-layer neural network with $N-1$ hidden units can give any N input-target relationship exactly [14, 25], whereas a network with two hidden layers can do the same with negligible error using $N/2+3$

[1] The only exceptions are the Widrow-Hoff and other linear regression-based algorithms.

hidden units [28]. Note that these results provide only an upper bound on the number of hidden neurons needed. For a specific problem one may be able to do far better by exploiting the structure in the data.

Recently, some researchers have investigated methods that alter network architecture as learning proceeds. These can be grouped into two categories, namely, (1) *pruning* algorithms, where one starts with a larger than needed network that iteratively removes inactive hidden neurons and associated connections until an acceptable solution is found; and (2) *constructive* algorithms, where one starts with a small network and adds hidden units and weights until a satisfactory solution is obtained. Each of the two methods has distinct advantages and disadvantages. For an overview on pruning algorithms, see [22]. Discussion on constructive approaches can be found in [15].

This chapter presents a review of standard learning algorithms for single-neuron/two-class problems where the set of input vectors is not linearly separable. We begin in section 4.2 with a review of the concepts of linear separability and its relationship to linear independence. In section 4.3 we categorize all such learning algorithms into two broad classes, and discuss representative algorithms from each class. In section 4.4 we choose a set of examples where the input data are not necessarily linearly separable and study the performance of each of these algorithms. The chapter concludes with a brief discussion in section 4.5.

In our analysis of performance of various learning algorithms we will draw from reference [26], which presents a detailed study on linearly nonseparable sets of input in general. It also includes specifically an analysis of the perceptron algorithm. In the rest of this chapter, we assume that the vectors are modified to satisfy the properties mentioned in remarks 1 and 2 above.

4.2 Linear Separability and Linear Independence

Let us explore the geometrical interpretation of equation (4.2) for a better understanding of the structure of the data set. For a linearly separable set of vectors, a separating hyperplane can be determined such that all vectors lie on one side of it. The normal to this hyperplane is the solution weight vector $\mathbf{w}^*$. In other words, placing a hyperplane such that all vectors lie on one side of this plane implies that the angle between any pair of vectors from the set must be less than π. Equivalently, the angle between the weight vector and any vector from the set must be less than $\pi/2$. This observation agrees with the fact presented in Nilsson [21] that input vectors that are almost perpendicular to the weight vectors are most difficult to train. Next, we explore the interesting relationship between linear separability and the more general concept of linear dependence/independence of a set of vectors.

The following theorems and definitions are taken from Siu et al. [26].

Proposition 4.2.1 *A set of vectors $\{\mathbf{z}^1, \mathbf{z}^2, ..., \mathbf{z}^m\}$ is linearly nonseparable if and only if there exists a positive linear combination of the vectors that equals* $\mathbf{0}$*, i.e., there exists $q_i \geq 0$, $1 \leq i \leq m$, such that $\sum_{i=1}^{m} q_i \mathbf{z}^i = 0$ and $q_j > \mathbf{0}$ for some j, $1 \leq j \leq m$.*

Note the similarity with the definition of a set of vectors that are linearly dependent. We provide definitions of linear dependence and linear independence for easy reference.

Proposition 4.2.2 *A set of vectors $\{\mathbf{z}^1, \mathbf{z}^2, ..., \mathbf{z}^m\}$ in vector space Z are linearly dependent if there exists numbers $q_1,\ q_2, ..., q_m$ not all equal to zero [the key difference with (4.2.1) is that here q_is may take negative values] such that $\sum_{i=1}^{m} q_i \mathbf{z}^i \;=\; \mathbf{0}$.*

We use geometry to present an alternative interpretation. The convention for measuring the angle between a pair of vectors is specified in the following remark:

Remark 3: The angle between any pair of vectors is the angle subtended on that side of the hyperplane for which the desired responses of the original set of vectors are +1.

A set of vectors is linearly dependent if one can construct a closed figure[2] using a subset (possibly the whole set) of the vectors. It is permissible to change the magnitude as well as the orientation of the vectors (change in orientation in this context means that one may either include the vector **a** or include $-\mathbf{a}$). On the other hand, a set of vectors is linearly nonseparable (that is, they cannot be made to lie on the same side of a hyperplane) if one can construct a closed figure using a subset (possibly the whole set) of the vectors. Here it is permissible to change the magnitude of the vectors but not their orientations. One may easily deduce that linear nonseparability implies linear dependence. However, linear dependence may or may not lead to linear nonseparability since if in constructing the closed figure one is required to change orientations of vectors, that would violate the restrictions imposed by linear nonseparability. Intuitively, it is more likely that one may be able to construct a closed figure (without changing the orientations) if the vectors do not lie on one side of a plane.

Proposition 4.2.3 *A set of vectors $\{\mathbf{z}^1, \mathbf{z}^2, ..., \mathbf{z}^m\}$ in vector space Z is linearly independent if and only if the following condition is satisfied: Whenever $q_1,\ q_2, ..., q_m$ are numbers such that $\sum_{i=1}^{m} q_i \mathbf{z}^i \;=\; \mathbf{0}$, then $q_i = 0\ \forall i$.*

Note that linear independence implies linear separability, but the converse is not true. That is, linear separability may or may not lead to linear independence (Fig. 4.1), since all we need to show for a set of vectors to be linearly separable is that no positive q can be found. If some negative q's (multiplication by $-q$ amounts to changing the direction of the vector) are found, then the set of vectors is linearly dependent as well as linearly separable. Intuitively, if all vectors are lying on the same side of a plane (the set is linearly separable), then it is highly unlikely that one can construct a closed figure with these vectors without changing their directions.[3] Here, we present a more general definition of linear nonseparability than one that is found in Siu et al.[26].

[2] We plot a set of n vectors by joining the head of the ith to the tail of the $(i+1)$th. A closed figure is formed if the head of the nth vector touches the tail of the 1st vector.

[3] It is rather conceivable that, for all vectors lying on one side of a hyperplane, a mixture of positive and negative q's can be found that satisfies linear dependence.

Proposition 4.2.4 *A set of vectors* $\{\mathbf{z}^1, \mathbf{z}^2, ..., \mathbf{z}^m\}$ *is linearly nonseparable if and only if there exists a linear combination of the vectors that equals* **0***, i.e., there exists* q_i*,* $1 \leq i \leq m$*, such that* $\sum_{i=1}^m q_i \mathbf{z}^i = \mathbf{0}$ *and* $q_j \neq 0$ *for some* j*,* $1 \leq j \leq m$ *and all nonzero* q_j*'s have the same sign.*

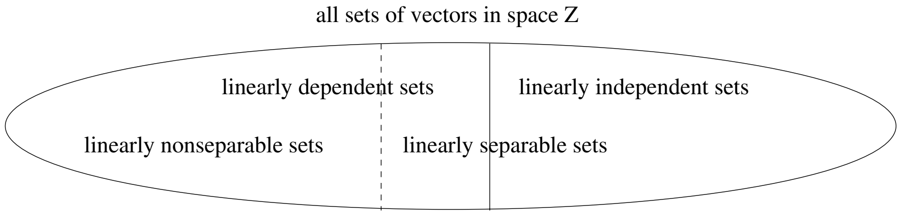
all sets of vectors in space Z
linearly dependent sets
linearly independent sets
linearly nonseparable sets
linearly separable sets

Fig. 4.1. Illustration of the relationship between linear dependence and linear separability. The dashed line divides between the linearly separable and nonseparable vector sets, and the solid line divides between the linearly dependent and independent vector sets. Linearly nonseparable sets are in a proper subset of linearly dependent sets.

In other words, a set of vectors can be (1) linearly nonseparable and dependent, (2) linearly separable and dependent, and (3) linearly separable and independent, but not (4) linearly nonseparable and independent.

Example 1: Linearly nonseparable and linearly dependent set

Consider a linearly nonseparable set **A** (XOR problem)

$$A_1 = [-1\ -1\ 1]^t;\ A_2 = [1\,1\,1]^t;\ A_3 = [-1\,1\ -1]^t;\ A_4 = [1\ -1\ -1]^t.$$

One solution to the equation $q_1A_1 + q_2A_2 + q_3A_3 + q_4A_4 = \mathbf{0}$ is

$$q_1 = q_2 = q_3 = q_4 = a,$$

where a can be any real number. The vectors in set **A** are linearly nonseparable since they satisfy proposition (4.2.1) and more generally proposition (4.2.4). Note that these vectors are linearly dependent since they satisfy proposition (4.2.2).

Example 2: Linearly separable and linearly dependent set

Consider a linearly separable set **B**

$$B_1 = [1\,1\,1]^t;\ B_2 = [1\ -1\,1]^t;\ B_3 = [1\ -1\ -1]^t;\ B_4 = [1\,1\ -1]^t.$$

A linear combination with either **all** positive or **all** negative coefficients that equals to zero cannot be found for the vectors in set **B**.

The only possible solution, with all q's bearing the same sign, to $q_1B_1 + q_2B_2 + q_3B_3 + q_4B_4 = \mathbf{0}$ is

$$q_1 = q_2 = q_3 = q_4 = 0.$$

Thus we conclude that vectors in this set are linearly separable. Note that these vectors are linearly dependent, since the equation can be satisfied with $q_1 = 1, q_2 = -1, q_3 = 1, q_4 = -1$.

Example 3: Linearly separable and linearly independent set

The following example illustrates that a set of linearly separable vectors can be linearly independent.

$$C_1 = [1\,1\,1]^t;\; C_2 = [1\,-1\,1]^t;\; C_3 = [1\,-1\,-1]^t.$$

The only values of coefficients that satisfy $q_1C_1 + q_2C_2 + q_3C_3 = \mathbf{0}$ are $q_1 = q_2 = q_3 = 0$.

In discussions of linear separability, one usually refers to a set of vectors of different classes. However, if the input vectors have been modified to carry the class membership as in remark 2, discussion on linear separability can be extended to single vectors. Such a discussion turns out to be helpful for characterization of the detailed structure of a data set.

Definition: A vector $\mathbf{z}^i$ is defined to be separable if it never participates in a positive linear combination that equals 0, i.e., $\sum_{j=1}^{m} q_j\mathbf{z}^j = \mathbf{0}$; $q_j \geq 0$ implies that $q_i = 0$.

Definition: A vector $\mathbf{z}^i$ is defined to be nonseparable if it participates in a positive linear combination that equals 0, i.e., $\sum_{j=1}^{m} q_j\mathbf{z}^j = \mathbf{0}$; $q_j \geq 0$ implies that there is some $q_i \neq 0$.

Two properties are noted on a set of nonseparable vectors [26]:

1. If the set of nonseparable vectors is nonempty, then it must consist of at least two vectors.
2. There exists a hyperplane passing through the origin, such that all the nonseparable vectors lie on the plane, and all the separable vectors lie on one side of it.

Geometrically, any given set of vectors are in one of the three configurations [17] (as shown in Fig. 4.2):

(a) there exists a vector x that makes a strict acute angle ($< \pi/2$) with all the vectors in A;
(b) there exists a vector x that makes an acute angle ($\leq \pi/2$) with all the vectors in A, and the origin 0 can be expressed as a nonnegative linear combination of vectors in A with positive weights assigned to those vectors in A that are orthogonal to x; and
(c) the origin can be expressed as a positive linear combination of all the vectors in A.

The proof is from Tucker's [30] first existence theorem, which states:
For any given $p \times n$ matrix A, the systems

$$Ax \geq 0 \text{ and } A'y = 0, y \geq 0$$
$$\text{possess solutions } x \text{ and } y \text{ satisfying}$$
$$Ax + y > 0.$$

The set of vectors is linearly separable in case (a), and nonseparable in cases (b) and (c). The nonseparability is caused by all the vectors in case (c), but only some of the vectors in case (b).

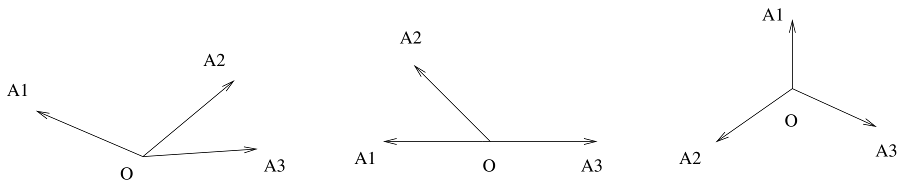

Fig. 4.2. Any given set of vectors A must be in one of these three configurations [17] shown by examples of three vectors in a two-dimensional space

.

Roychowdhury et al. [24] give a simple procedure to determine the set of separable and nonseparable vectors using a linear programming (LP) formulation. Let e_i be a column vector with component i being 1 and all other components being zero. Vector z_i is nonseparable if the objective function of the linear program

$$\begin{aligned} \text{minimize} \quad & {e_i}^t \mathbf{q} \\ \text{subject to } & \mathbf{Z}^t \mathbf{q} = \mathbf{0} \\ & \mathbf{q} \geq \mathbf{0} \end{aligned} \tag{4.3}$$

is unbounded. This happens if and only if vector z_i participates, with a nonzero coefficient, in a positive linear combination of the input vectors that equals zero. Hence, the set of separable and nonseparable vectors can be determined by solving one such LP problem for each input vector, i.e., a total of m LP problems.

4.3 Analysis of Representative Learning Algorithms

Descent procedures are those that modify the weight vector as the algorithms examine the input vectors one by one. There are non–descent- based methods for obtaining linear classifiers such as Fisher's linear discriminant analysis. In this chapter we focus on descent procedures. We broadly categorize them into two groups, namely, (A) error correction procedures and (B) error minimization procedures. However, it should be noted that the term *error* is defined differently in each context. In group A it refers to an instance of misclassification, and in group B it refers to a measure of distance of a point from a hyperplane. The remainder of this section discusses representative methods from each group.

4.3.1 Group A: Error Correction Procedures

Fixed-Increment Perceptron Training Rule: Among the adaptive procedures, the fixed-increment perceptron training rule (from now on called the perceptron rule) is the most well known. We discuss the performance of the classic perceptron rule and its several variations for the nonseparable case. The perceptron weight update rule [23] can be stated as

$$\begin{array}{rcl} \mathbf{w}^0 & & \text{arbitrary} \\ \mathbf{w}^{j+1} = & \mathbf{w}^j + \alpha \mathbf{z}^j & \text{if } {\mathbf{z}^j}^t \mathbf{w}^j \leq 0 \\ \mathbf{w}^{j+1} = & \mathbf{w}^j & \text{otherwise.} \end{array} \tag{4.4}$$

The learning coefficient $\alpha > 0$ and the length of the input vector control the magnitude of change. It can be shown [23] that if the vectors are linearly separable, this rule will produce a solution vector $\mathbf{w}^*$ in a finite number of steps. However, for input vectors that are not linearly separable the perceptron algorithm does not converge, since, if the input vectors are nonseparable, then for any set of weights $\mathbf{w}$, there will exist at least one input vector, $\mathbf{z}$, such that $\mathbf{w}$ misclassifies $\mathbf{z}$. In other words, the algorithm will continue to make weight changes indefinitely. The following theorem, however, states that even if the algorithm iterates indefinitely, the length of the weight vector will remain bounded.

Proposition 4.3.1 The perceptron cycling theorem:[4] *Given a finite set of linearly nonseparable training samples* $\{\mathbf{z}^1, \mathbf{z}^2, ..., \mathbf{z}^m\}$, *there exists a number N such that if the perceptron learning rule is applied to this set, then the weight vector* $\mathbf{w}^l$ *at any iteration l remains bounded, i.e.,* $|\mathbf{w}^l| \leq N$.

Instead of formally proving this theorem (interested readers can find the proof in [3]), for intuitive purposes we explore the geometric aspect of the perceptron updating rule to see why the perceptron cycling theorem holds. Let $\mathbf{z}^n$ be the training sample that is misclassified by the current weight vector $\mathbf{w}^n$. Therefore, the angle between the two vectors must satisfy

$$\pi/2 < \theta(\mathbf{z}^n, \mathbf{w}^n) < 3\pi/2 \tag{4.5}$$

in order for the inner product to be negative. We observe that, maintaining the satisfiability of the criterion $|\mathbf{w}^n|^2 < |\mathbf{w}^{n+1}|^2 < (|\mathbf{w}^n|^2 + |\mathbf{z}^n|^2)$, with $|\mathbf{z}^n|$ remaining fixed, becomes increasingly difficult as $|\mathbf{w}^n|$ grows. Ultimately, the above criterion becomes unsatisfiable at some point as $|\mathbf{w}^n|$ becomes larger than certain limiting value. Therefore, $|\mathbf{w}^n|$ cannot grow without bound.

In cases where the input vectors are integer-valued, the bounded weight vectors cycle through a finite set of values [19]. An observation of such cycling is an indication that the input is linearly nonseparable, though there are no known time bounds for this to become observable. Therefore, this theorem is of little use for practical applications.

Gallant [8] has proposed a modification of the perceptron rule, the pocket algorithm, so that it finds the optimal weight vector, i.e., a weight vector that will classify as many training samples as possible. The idea is to keep a separate set of weights, $\mathbf{w}^{pocket}$, along with the number of consecutive training samples that it has classified correctly. Whenever the current perceptron weight, $\mathbf{w}^n$, has a longer run of correct classification, the pocket weight $\mathbf{w}^{pocket}$ is replaced by $\mathbf{w}^n$. Unfortunately, there is no known bound on the number of iterations required to guarantee optimal weight.

[4] This theorem is also true for separable classes.

Usually suboptimal weight is achieved by this algorithm. A further revised version, the pocket algorithm with ratchet, is proposed [8] to ensure that the pocket weight strictly improves as it gets replaced by the current weight. This is implemented by checking whether the current weight $\mathbf{w}^n$ classifies more training examples than the current $\mathbf{w}^{pocket}$. Only then $\mathbf{w}^{pocket}$ is replaced by $\mathbf{w}^n$. Note that this improvement comes with increased computational burden, especially when there are many training samples.

In a much earlier and rarely cited reference, we find that Butz [5] has proposed to modify the perceptron update rule to address nonseparable data. A small positive *reinforcement* factor $\mu, \mu \le \mu_0 < 1$ is introduced, with μ_0 being a certain constant, so that in place of (4.4) one has

$$\begin{aligned} \mathbf{w}^0 & \quad \text{arbitrary} \\ \mathbf{w}^{j+1} &= \mathbf{w}^j + \alpha \mathbf{z}^j \text{ if } {\mathbf{z}^j}^t \mathbf{w}^j \le 0 \\ \mathbf{w}^{j+1} &= \mathbf{w}^j + \mu \mathbf{z}^j \text{ otherwise.} \end{aligned} \tag{4.6}$$

Note the similarity in the underlying concept in (4.6) and the pocket algorithm. Both attempt to reward the weight vector that correctly classifies the training samples. This approach moves the weight vector closer to the sample that is correctly classified, unlike the perceptron rule (and most others) where no action is taken in such cases. Let $\mathbf{z}^j$ and $\mathbf{z}^{j+1}$ be two consecutive training samples from the same class. Let us assume that $\mathbf{z}^j$ gets correctly classified by the current weight vector $\mathbf{w}^j$. The next weight vector is $\mathbf{w}^{j+1} = \mathbf{w}^j + \mu \mathbf{z}^j$. Let us assume that the next sample $\mathbf{z}^{j+1}$ is misclassified by the weight vector $\mathbf{w}^{j+1}$. The resulting weight vector $\mathbf{w}^{j+2}$ is more likely to keep $\mathbf{z}^j$ correctly classified in the current procedure than in that of the perceptron update procedure. Butz has shown that the error rate $k(n)/n$ decreases considerably with the introduction of a small reinforcement factor where n is the number of samples and $k(n)$ is the number of misclassifications.[5] The author goes on to show that

- A $\mu_0 < 1$ exists.
- If $\{\mathbf{z}^n\}$ is a sequence of mutually independent sample vectors, then $k(n)/n \le G(\mu)$ with probability 1 as $n \to \infty$. $G(\mu)$ is a continuous and monotone nonincreasing function of μ, with $G(0) < 1$ and $G(\mu_0) = p_0$. The quantity p_0 is the lower limit of the error probabilities associated with the weight vector set.

In other words, with the choice of *right* $\mu \le \mu_0$, one can find a weight vector $\mathbf{w}'$ such that the rate of misclassification decreases.

However, the quantity μ_0 (thus μ) rarely is known or can be estimated with reasonable accuracy. Successful application of this rule depends on a good value of μ

[5] The comparison between Butz's result and the perceptron behavior is not quite appropriate. With nonlinear input (i.e., linearly nonseparable input) the weight vectors produced by the perceptron rule may go from the best possible to the worst possible classification result in one iteration. It is almost impossible to pick the best weight vector from the perceptron procedure to make comparisons on misclassification especially for a large data set.

which has to be searched for, and the search may be computationally expensive. The other major concern is the stopping criterion, which is not discussed in this chapter.

It has been shown in [26] that perceptron learning algorithm can be used to learn the set of separable vectors and identify the set of nonseparable vectors. The definitions of separable and nonseparable vectors are given at the end of section 4.2. The major results related to the perceptron algorithm with nonlinear input are as follows:

Proposition 4.3.2 *The perceptron algorithm can separate a set of given vectors into (i) a set of separable vectors and (ii) a set of nonseparable vectors in a finite number of steps.*

A note of caution is in order. The proof of this theorem establishes the finiteness of the number of steps by showing that the upper bound for the number of steps is the constant N, which appears in the perceptron cycling theorem. With no concrete information on the upper bound and no stopping criterion, the usefulness of the above theorem in practical applications is questionable.

Proposition 4.3.3 *Given a set of vectors, the perceptron algorithm can be used to determine a linearly separable subset of maximum cardinality.*

Proposition 4.3.2 can be implemented in polynomial time using linear programming formulation (not the perceptron algorithm). This yields a set of separable vectors that may not lead to the best possible classification result. Proposition 4.3.3 is much more useful because it can be used to produce a weight vector that gives the least number of misclassifications. In this sense it is a more powerful result than the least-square methods (Widrow-Hoff, since there is no problem with local minima), Gallant's pocket algorithm, and Butz's reinforcement approach. Unfortunately, the authors show that an algorithm for this proposition is NP-complete. They propose a heuristic approach to solve proposition 4.3.3, which does not guarantee optimal results.

Projection Learning Rule (Alias fractional correction rule:) The projection learning rule [1] (more commonly known as the fractional correction rule), a variation of the perceptron rule, is also based on the error correcting principle. However, its behavior with nonlinear input is quite different from that of the perceptron rule. The weights are updated in the following manner:

$$\begin{array}{ll} \mathbf{w}^0 & \text{arbitrary} \\ \mathbf{w}^{j+1} = \mathbf{w}^j - \alpha \frac{(\mathbf{w}^{j^t}\mathbf{z}^j)\mathbf{z}^j}{|\mathbf{z}^j|^2} & \text{if } \mathbf{z}^{j^t}\mathbf{w}^j \le 0. \end{array} \tag{4.7}$$

It can be shown that this algorithm converges for linearly separable input. The result pertaining to nonlinear input can be stated as follows [1]:

Proposition 4.3.4 *If the pattern set is not linearly separable, then the projection learning rule converges to a* **0** *solution for* $0 < \alpha < 2$.

Proof: Since no hyperplane can separate the input, the weight vector has to be updated at least once in each cycle. From equation 4.7 one can derive that

$$\|\mathbf{w}^{j+1}\|^2 - \|\mathbf{w}^j\|^2 = \frac{\alpha(\alpha-2)({\mathbf{z}^j}^t\mathbf{w}^j)^2}{\|\mathbf{z}^j\|^2}.$$

Note that $\forall\ \alpha, 0 < \alpha < 2$, $\|\mathbf{w}^{j+1}\|^2 < \|\mathbf{w}^j\|^2$. This indicates that the sequence $\|\mathbf{w}^0\|, \|\mathbf{w}^1\|, ...$ is a strictly monotonically decreasing sequence with the lower bound 0. Therefore, $\|\mathbf{w}^j\|$ approaches zero as j approaches infinity. This proves that the projection learning rule converges to the only solution (i.e., $\mathbf{w} = \mathbf{0}$) in the linearly nonseparable case for $0 < \alpha < 2$.

For one-dimensional linearly nonseparable input, one can show that $\|w\|$ falls within a small distance ϵ from zero in a number of steps that can be expressed as a function of the initial weight vector, α, ϵ, and the angles between the input vectors [1]. However, no similar expression has been known for higher dimensional input.

A link can be established between this result and the concept of linear independence. From the given set of m n-dimensional vectors $\{\mathbf{z}^1, \mathbf{z}^2, ..., \mathbf{z}^m\}$, construct a set of n m-dimensional vectors$\{\hat{\mathbf{z}}^1, \hat{\mathbf{z}}^2, ..., \hat{\mathbf{z}}^n\}$ in the following manner:

$$\hat{\mathbf{z}}^j = \{z_j^1, z_j^2, ...z_j^m\}$$

where z_j^i is the jth component of the $\mathbf{z}^i$ vector. If the original set of vectors is not linearly separable, the weight update rule converges to the only solution, which is $\mathbf{w} = 0$. This leads to the fact that the constructed set of vectors will be a linearly independent set in m-dimensional space.

4.3.2 Group B: Error Minimization Procedures

The error correction procedure focuses on misclassified samples. Other procedures modify the weight vector using all samples at each iteration. Moreover, thus far a weight vector $\mathbf{w}$ is sought such that $\mathbf{w}^t\mathbf{z}^j\ \forall j$ is positive. Next, we discuss attempts to reformulate the problem of finding the solution to a set of linear inequalities as a problem of finding a solution to a set of linear equations. Let $\mathbf{b} = (b^1, b^2, ..., b^m)^t$ be a column vector. The decision equation (4.2) can be restated as

$$\mathbf{Z}^t\mathbf{w} = \mathbf{b}. \tag{4.8}$$

The solution vector $\mathbf{w}$ is overdetermined since $\mathbf{Z}^t$ is rectangular with more rows than columns, assuming $m > n$. The idea is to search for a weight vector that minimizes some function of the error between the left and the right sides of equation (4.8). The usual choice of a function to be minimized is the one that represents the sum-of-squared error:

$$J = |\mathbf{Z}^t\mathbf{w} - \mathbf{b}|^2.$$

The central theme in both the Widrow-Hoff and the Ho-Kashyap procedures is to minimize J, though the difference in the details of the algorithms leads to drastically different results.

Widrow-Hoff delta rule: The α-Least-Mean-Square (α-LMS) algorithm or Widrow-Hoff delta rule embodies the minimal disturbance principle[6] [33]. It is designed to handle both linearly separable and linearly nonseparable input. The criterion function is minimized with respect to the weight vector $\mathbf{w}$ using the gradient decent method. The unknown vector $\mathbf{b}$ is chosen arbitrarily and held constant throughout the computation. See [35] for derivation of the weight update equation using the gradient decent method. The iterative version of the weight update equation can be written as follows [35]:

$$\begin{aligned} &\mathbf{w}^0 && \text{arbitrary} \\ &\mathbf{w}^{j+1} = \mathbf{w}^j + \alpha\tfrac{(b^j - \mathbf{w}^{j^t}\mathbf{z}^j)\mathbf{z}^j}{|\mathbf{z}^j|^2}. \end{aligned} \tag{4.9}$$

Note that the class labels (or desired output for all input samples) d^j, which are the output after the nonlinearity, are known. Since the error $\epsilon_l^j = (b^j - \mathbf{w}^{j^t}\mathbf{z}^j)$ is measured at the linear output, not after the nonlinearity, as in the case of the perceptron rule, one must choose the magnitude as well as the sign of b^j (arbitrarily) to continue. It has been shown [35] that, in both linearly separable as well as linearly nonseparable cases, this rule converges in the mean square sense to the solution $\mathbf{w}^*$ that corresponds to the least mean square output error if all input vectors are of the same length. It is known that in some cases this rule may fail to separate training vectors that are linearly separable [18]. This is not surprising, since the mean square error (MSE) solution depends on the margin vector $\mathbf{b}$. Different choices for $\mathbf{b}$ give the solution different properties. Hence, when one does not have any clue about the distribution of the input data and arbitrarily fixes a margin vector, it is very likely that the resulting weight vector may not classify all vectors correctly even for a linearly separable problem.

Ho-Kashyap algorithm: Ho and Kashyap [13] modified the Widrow-Hoff procedure to obtain a weight vector $\mathbf{w}$ as well as a margin vector $\mathbf{b}$. They imposed the restriction that the m-dimensional margin vector must be positive-valued, i.e., $\mathbf{b} > 0$, ($b_k > 0, \ \forall \ k$). The problem is equivalent to finding $\mathbf{w}$ and $\mathbf{b} > 0$ such that $J = |\mathbf{Z}^t\mathbf{w} - \mathbf{b}|^2$ is minimized with respect to both $\mathbf{w}$ and $\mathbf{b}$. Note that since both $\mathbf{w}$ and $\mathbf{b}$ (subject to the imposed constraint) are allowed to play a role in the minimization process, the minimum value (i.e., 0) for J can be achieved in this case. Thus the $\mathbf{w}$ that achieves that minimum is the separating vector in the linearly separable case. The weight update rule is (for detailed derivation see [13])

[6] The rule aims at making minimum possible change in the weight vector during the update process such that the output for as many of the previously correctly classified samples as possible remains unperturbed.

$$\begin{aligned}
\mathbf{b}^0 &> 0 \quad \text{otherwise arbitrary} \\
\mathbf{w}^0 &= (\mathbf{Z}^t)^\dagger \mathbf{b}^0 \\
\epsilon^j &= \mathbf{Z}^t \mathbf{w}^j - \mathbf{b}^j \\
\mathbf{b}^{j+1} &= \mathbf{b}^j + \alpha(|\epsilon^j| + \epsilon^j) \\
\mathbf{w}^{j+1} &= \mathbf{w}^j + \alpha(\mathbf{Z}^t)^\dagger(|\epsilon^j| + \epsilon^j)
\end{aligned} \tag{4.10}$$

where $(\mathbf{Z}^t)^\dagger = (\mathbf{Z}\mathbf{Z}^t)^{-1}\mathbf{Z}$ is the pseudoinverse of $\mathbf{Z}^t$. Computation of the pseudoinverse may be avoided by using the following alternate procedure [12]:

$$\begin{aligned}
\epsilon_k^j &= \mathbf{z}^{k^t} \mathbf{w}^j - {b_k}^j \\
{b_k}^{j+1} &= {b_k}^j + \rho_1/2(|\epsilon_k^j| + \epsilon_k^j) \\
\mathbf{w}^{j+1} &= \mathbf{w}^j - \rho_2 \mathbf{z}^k(\mathbf{z}^{kt}\mathbf{w}^j - b_k^{j+1}).
\end{aligned} \tag{4.11}$$

This algorithm yields a solution vector in the case of linearly separable samples in a finite number of steps if $0 < \rho_1 < 2$, and $0 < \rho_2 < 2/\|\mathbf{z}^k\|^2$. Since the focus of this chapter is on data that are not separable, let us examine the behavior of this algorithm under nonseparable situation.

Note the following two facts for nonseparable case: **fact 1,** $\epsilon^j \neq 0$ for any j; and **fact 2,** $|\epsilon^{j+1}|^2 < |\epsilon^j|^2$, i.e., the sequence $|\epsilon^1|^2$, $|\epsilon^2|^2$, ... is a strictly monotonically decreasing sequence and must converge to the limiting value $|\epsilon|^2$, though the limiting value cannot be zero. It can be shown that $(\epsilon^j + |\epsilon^j|)$ converges to zero, suggesting a termination of the procedure. Now consider the following two cases:

- Suppose the error vector has no positive component for some finite j, then $|\epsilon^j| + \epsilon^j = 0$. In that case the correction will cease and neither the weight vector nor the margin vector will change [see (4.10)]. Thus an error vector with no positive components conclusively points to the nonlinear nature of the data.[7]
- Suppose ${\epsilon_j}^+ = (\epsilon^j + |\epsilon^j|)$ is never zero for finite j. We can derive from **fact 2** that $|{\epsilon_{j+1}}^+|^2 < |{\epsilon_j}^+|^2$. Therefore, $|{\epsilon_j}^+|$ must converge to zero. However, its distance from zero is unknown for any fixed j.

In summary, the Ho-Kashyap algorithm indicates nonseparability, but there is no bound on the number of steps.

Hassoun and Song [12] propose a variation of the Ho-Kashyap algorithm equipped to produce an *optimal* separating surface for linearly nonseparable data. The authors claim that it can identify and discard nonseparable samples to make the data linearly separable with increased convergence speed. Again we notice similarity with the heuristic approach proposed in Siu et al. [26]. However, the authors do not provide any theoretical basis to their claim. The algorithm is tested only on simple toy problems. We have no knowledge of any extensive testing on real-world problems.

[7] It can be shown that for linearly separable data, it is impossible for all components of the error vector to be negative at any given iteration.

linear programming: In searching for a linear classifier, the input vectors give a system of linear inequalities constraining the location and orientation of the optimal separating hyperplane. With a properly defined objective function, a separating hyperplane can be obtained by solving a linear programming problem. Several alternative formulations have been proposed in the past ([4, 9, 16, 24, 27]) employing different objective functions. An early survey of these methods is given in Grinold [11]. Here we mention a few representative formulations.

In a very simple formulation described in [24], the objective function is trivial, so that it is simply a test of linear separability by finding a feasible solution to the LP problem

$$\begin{aligned} \text{minimize} \quad & \mathbf{0}^t\mathbf{w} \\ \text{subject to } \mathbf{Z}^t\mathbf{w} \ & \geq \ \mathbf{1} \end{aligned} \tag{4.12}$$

where $\mathbf{w}$ is the weight vector of a separating hyperplane, $\mathbf{Z} = [\mathbf{z^1}, \mathbf{z^2}, ..., \mathbf{z^m}]$ is a matrix of column vectors $\mathbf{z}^j \ \ (j = 1, ..., m)$, the m augmented input vectors are as defined before, and **0** and **1** are vectors of zero's and one's, respectively. The constraint requires that all training samples must fall on the same side of the hyperplane (the linear classifier to be found). This is possible only if the points are linearly separable. **1** is an arbitrarily chosen nonzero constant vector that specifies a margin, so that the points do not lie on the hyperplane.

This formulation gives only a test for linear separability (whether the constraints give a feasible region) but does not lead to any useful solution if the data are not linearly separable [24]. Another formulation suggested by Smith ([27]; see also [10]) minimizes an error function:

$$\begin{aligned} \text{minimize} \quad & \mathbf{a}^t\mathbf{t} \\ \text{subject to } & \mathbf{Z}^t\mathbf{w} + \mathbf{t} \geq \mathbf{b} \\ & \mathbf{t} \geq \mathbf{0} \end{aligned} \tag{4.13}$$

where $\mathbf{Z}$ is the augmented data matrix as before, $\mathbf{a}$ is a positive vector of weights, $\mathbf{b}$ is a positive margin vector chosen arbitrarily (e.g., **b** = **1**), and $\mathbf{t}$ and $\mathbf{w}$ are the error and weight vectors, which are also decision variables for the LP problem. In [27] each component of **a** was set to be $1/m$. The variable error vector $\mathbf{t}$ allows that some points locate on the other side of the hyperplane. $\mathbf{t}$ is required to be positive so it will decrease the fixed margin specified by $\mathbf{b}$. Using this formulation, the error function will be minimized at zero with linearly separable data, and at a nonzero value with linearly nonseparable data.

Bennett and Mangasarian [4] modified this formulation to use different weights for input vectors belonging to each of the two classes. Let m_1, m_{-1} be the number of vectors belonging to the two classes, respectively, $\mathbf{a} = (a_1, a_2, ..., a_m)^t$, for $j = 1, ..., m$,

$$a_j = 1/m_1 \text{ if } d^j = 1,$$
$$a_j = 1/m_{-1} \text{ if } d^j = -1.$$

It is argued that this formulation is more robust in the sense that it can guarantee a nontrivial solution **w** even if the centroids of the two classes happen to coincide.

Instead of minimizing an error function that depends on distances of all input points to the separating hyperplane, Vapnik and Chervonenkis [31] proposed maximizing the distances of only those points that are closest to the hyperplane. This approach of maximal margin hyperplanes led to the development of support vector machines [32], which optimize the same objective function for points transformed to a different space.

Though arguably algorithms for solving LP problems are more sophisticated than the previously discussed iterative procedures, an important advantage of finding **w** by solving an LP problem is that if the feasible region is nonempty, the solution can be determined in a finite number of steps.

The number of steps it takes to arrive at the solution, however, is dependent on the geometrical configuration of the data points. If the LP is solved by the simplex method, in the worst case the algorithm may have to visit every vertex formed by the intersections of the constraining hyperplanes before reaching an optimum. Empirical evidence shows that in practice this rarely happens. More recently, interior-point methods [36], such as Karmarkar's, are shown to have a better worst-case time complexity. Still, the comparative advantages of such methods for an arbitrary problem remain unclear, partly because there has not been a good way to characterize the structure of a particular problem and relate that to the detailed operations of the algorithms.

4.4 Experimental Results

In this section we present some experimental results on applying several representative learning algorithms to data sets that are not necessarily linearly separable. The purpose of these experiments is more for assessing the practicality of the methods than for a comprehensive evaluation and comparison.

4.4.1 Algorithms for Determining Linear Separability

We applied the three procedures [Ho-Kashyap (HK), fractional correction rule (FCR), and linear programming (LP)] that are claimed to indicate linear nonseparability to a collection of two-class discrimination problems. The original Ho-Kashyap rule involves computing a pseudoinverse matrix, which turned out to be overly expensive for large problems. So the adaptive algorithm (4.11) was used instead. With LP, we used the simple formulation (4.12) to test for linear separability (referred to as LPtest), and also Smith's formulation (4.13) to derive a minimum error separating hyperplane (referred to as LPme). We included the perceptron training rule too for comparison purpose (denoted by PER for PERceptron), although it is understood that it does not converge for nonseparable input.

Recall that as reviewed in previous sections, LPtest determines linearly separability by checking whether the constraints give a feasible region, and LPme by checking whether the optimal value for the objective function is zero. HK reports linear separability by arriving at an error vector with zero norm, and linear nonseparability by that the error vector has only negative components, or the norm of the positive part of the error is very close to zero. FCR determines separability by arriving at a solution, or the norm vector as well as the difference in subsequent norm vectors are very close to zero; and PER determines separability by arriving at a solution. Experimentally, a solution is not pursued further after some fixed time, and no solution indicates that the programs do not stop within that time.

The problem was discrimination between all pairs of classes in each of 14 data sets from the University of Californa, Irvine (UCI) Machine Learning Depository [2]: abalone, car, german, kr-vs-kp, letter, lrs, nursery, pima, segmentation, splice, tic-tac-toe, vehicle, wdbc, and yeast. The data sets were chosen so that each set has at least 500 input vectors and no missing values in the features. For those sets containing categorical features, the values were numerically coded. There are a total of 844 two-class discrimination problems. Outcomes from the algorithms may be a conclusion of whether the problem is linearly separable or not, or inconclusive after a chosen number of iterations. We compared such outcomes from all three procedures and the relative time it took for them to derive the results.

Table 4.1 shows the number of problems reported by each algorithm as separable, nonseparable, or inconclusive. Since LPtest always gives a definite answer, conclusions of other algorithms are compared to its results.

Table 4.1. Conclusion on linear separability by each algorithm (entries are number of problems; sep: separable, non: nonseparable, inc: inconclusive).

	LPtest	HK			FCR			PER		
		sep	non	inc	sep	non	inc	sep	non	inc
separable	452	272	31	149	449	0	3	428	0	24
nonseparable	392	0	1	391	0	4	388	0	0	392

We found that within similar, affordable run-time limits, linear programming always arrived at a definite conclusion, while HK and FCR often ended the runs inconclusively (100,000 iterations were used for HK, FCR, and PER). Of the 844 problems, LPtest reported that 452 are linearly separable and 392 are not. For 54% (453/844) of these problems, FCR arrived at a conclusion, but the fraction is only 36% (304/844) for HK. For the separable problems, FCR arrived at the conclusion generally sooner than HK. For the nonseparable problems, both algorithms have a problem fulfilling the claim of giving an indication; within the affordable time, only one problem was reported to be nonseparable by HK, and only four by FCR.

We also found that the results were very sensitive to the choices of the learning coefficient and convergence thresholds for the Ho-Kashyap rule. Other than affect-

Table 4.2. Correlation coefficients of number of iterations for each pair of algorithms to converge to a separating hyperplane.

	LPme	HK	FCR	PER
LPtest	0.6974	0.0164	0.1930	0.1256
LPme		−0.0275	0.1586	0.1341
HK			0.7306	0.5723
FCR				0.5421

Table 4.3. Correlation coefficients of problem size measures and number of iterations for each algorithm to converge to a separating hyperplane. n: no. of dimensions; m: no. of input vectors.

	LPme	LPtest	HK	FCR	PER
n	0.1970	0.5749	−0.0928	−0.0746	−0.0555
m	0.8856	0.4056	−0.0067	−0.0704	−0.0768
nm	0.4778	0.7636	−0.0429	−0.0388	−0.0287

ing the speed of convergence, they can change the conclusion on separability; for 31 separable problems HK reported that they were nonseparable. An improper learning coefficient may cause overly large correction steps. The tolerance threshold determines when a number is considered zero, which leads to a conclusion. Of those 31 problems falsely reported to be nonseparable by HK, 26 contain only one vector in the smaller class. The other five problems have two vectors in the smaller class. With a change in the learning coefficient (to 10% of original value), six were reported separable, nine remained nonseparable, and the other 16 became inconclusive.

Table 4.2 shows, for the separable problems, the correlation coefficients of the number of iterations it took for each pair of algorithms to converge to a hyperplane, computed over only those cases where both algorithms converged. In LP each iteration involves an input vector, but in the adaptive algorithm each iteration involves a loop through all relevant vectors in the entire set. For this reason, significant correlation exists only between the adaptive algorithms or the two LP formulations, but not between any of the adaptive algorithms and LP. The correlation is stronger between HK and FCR than between each of them and PER.

Correlation coefficients were also calculated between measures of the problem size and the number of iterations before convergence for each algorithm (Table 4.3). For the adaptive procedures, the absence of significant correlation suggests that the difficulty of a problem does not necessarily depend on the problem size.

4.4.2 Analysis of the Linearly Nonseparable Cases

For each of the 392 problems that LPtest reported to be linearly nonseparable, we applied procedure (4.3) to determine the set of separable and nonseparable vectors. Several of those problems can be considered nearly linearly separable, since only a small fraction of vectors are found responsible for nonseparability. However, for the majority (332) of those problems, the LP procedure found no separable vectors. That is to say, every point in those problems is (jointly) responsible for linear nonseparability. This is a rather surprising result. It suggests a need to further examine the difference between these problems and those with some separable points.

We attempted to relate the fraction of separable vectors to the number of steps taken for the three algorithms (LPtest, HK, FCR) to reach the conclusion of linear separability. However, since HK and FCR arrived at the conclusion for only a few of these nonseparable cases, we studied only the run time of LPtest. It appears that for most of the problems with no separable vectors and for almost all problems with many separable vectors, LPtest determined nonseparability fairly quickly. However, occasionally it did take many iterations for LPtest to arrive at that conclusion. This suggests a need for a more detailed examination of the locations of the separable and nonseparable vectors to understand the implications.

4.4.3 Discussion

Our experiments show that linear programming, although long neglected in classification studies, generally yields more affordable and dependable results. In contrast, the Ho-Kashyap and the fractional correction rules frequently do not converge within affordable time limits. The Ho-Kashyap rule may even lead to the wrong conclusions. It is very difficult to choose the learning coefficients and error tolerance thresholds to get all the conclusions right. This reinforces the theoretical results that there is no known proof of convergence for HK and FCR in a predictable number of steps, and that linear programming has known, predictable time bounds for convergence on both linearly separable and nonseparable cases. The implementation difficulties raise doubts about the practicality of these adaptive algorithms.

The only reservation we have about this conclusion is related to the efficiency of the implementations. For linear programming we used the MINOS solver [20] through the AMPL interface [7], which was a highly optimized commercial code, whereas the adaptive procedures were run with simple implementations in C written by ourselves. So, affordable (elapsed) time may not mean the same thing for the two groups. Also, we have not tested the dependence of the run time on the order in which the input vectors were presented to the adaptive procedures, and we have not investigated the dual problems that can be formulated for a given problem and solved by any of these procedures [24]. Nevertheless, we advocate that linear programming methods warrant more serious attention in classification studies. Without more sophisticated derivatives such as simultaneous primal-dual algorithms, the only apparent advantages of the adaptive procedures such as HK and FCR rules seem to be that (1) they can be implemented on very simple machines; and (2) their adaptive nature permits easier inclusion of new input that may become available during the training process, and thus they are better suited for online learning.

4.5 Conclusion

We reviewed a set of representative descent procedures for constructing linear classifiers, and experimented, using a public database, with three of those procedures that are claimed to detect linear nonseparability. We found that both the fractional correction rule and the adaptive Ho-Kashyap algorithm are not delivering on their promises, and that linear programming is the only reliable and efficient method. We suggest that linear programming methods can play a more significant role in classification studies. Moreover, we tested a linear programming formulation that reveals how close a problem is to linear separability by identifying vectors that are responsible for nonseparability. We believe further investigations along this line will lead to better understanding of the complexity of a given data set.

Acknowledgments

We thank Ken Clarkson, David Gay, Linda Kaufman, George Nagy, and Margaret Wright for helpful discussions and suggestions of references. Mitra Basu thanks Bell Labs for summer support in 1998.

References

[1] M. Basu, Q. Liang. The fractional correction rule: a new perspective. *Neural Network,* 11, 1027–1039, 1998.

[2] C. Blake, E. Keogh, C.J. Merz. UCI repository of machine learning databases [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science, 1998.

[3] H.D. Block, S.A. Levin. On the boundedness of an iterative procedure for solving a system of linear inequalities. *Proc. AMS*, 26, 229–235, 1970.

[4] K.P. Bennett, O.L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1, 23–24, 1992.

[5] A.R. Butz. Perceptron type learning algorithms in nonseparable situations. *Journal of Mathematical Analysis and Applications,* 17, 560–576, 1967.

[6] R.O. Duda, P.E. Hart. *Pattern Classification and Scene Analysis*. NewYork:Wiley, 1973.

[7] R. Fourer, D.M. Gay, B.W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. South San Francisco:The Scientific Press, 1993.

[8] S.I. Gallant. *Neural Network Learning & Expert Systems*. Cambridge, MA: MIT Press, 1993.

[9] F. Glover. Improved linear programming models for discriminant analysis. *Decision Sciences*, 21(4), 771–785, 1990.

[10] R.G. Grinold. Comment on “Pattern Classification Design by Linear Programming”. *IEEE Transactions on Computers*, C-18(4), 378–379, April 1969.

[11] R.G. Grinold. Mathematical programming methods of pattern classification. *Management Science*, 19(3), 272–289, 1972.

[12] M.H. Hassoun, J. Song. Adaptive Ho-Kashyap rules for perceptron training. *IEEE Transactions on Neural Networks*, 3, 51–61, 1992.

[13] Y.C. Ho, R.L. Kashyap. An algorithm for linear inequalities and its applications. *IEEE Transactions on Electronic Computers*, 14, 683–688, 1965.

[14] S.-C. Huang, Y.-F. Huang. Bounds on the number of hidden neurons in multilayer perceptrons. *IEEE Transactions on Neural Networks*, 2, 47–55, 1991.
[15] T.Y. Kwok, D.Y. Yeung. Objective functions for training new hidden units in constructive neural networks. *IEEE Transactions on Neural Networks*, 8(5), 1131–1148, 1997.
[16] O.L. Mangasarian. Linear and nonlinear separation of patterns by linear programming. *Operations Research*, 13, 444–452, 1965.
[17] O.L. Mangasarian. *Nonlinear Programming*, New York:McGraw-Hill, 1969.
[18] C.H. Mays. *Adaptive Threshold Logic*. Ph.D. thesis, Stanford Electronics Labs, Stanford, CA, 1963.
[19] M. Minsky, S. Papert. *Perceptrons*, expanded edition. Cambridge, MA:MIT Press, 1988.
[20] B.A. Murtagh, M.A. Saunders. Large-scale linearly constrained optimization. *Mathematical Programming*, 14, 41–72, 1978.
[21] N.J. Nilsson. *The Mathematical Foundations of Learning Machines*. San Mateo, CA: Morgan Kaufmann, 1990.
[22] R. Reed. Pruning algorithms—a survey. *IEEE Transactions on Neural Networks*, 4, 740–747, 1993.
[23] F. Rosenblatt. *Principles of Neurodynamics: Perceptron and the Theory of Brain Mechanism*. Washington, D.C.: Spartan Press, 1962.
[24] V.P. Roychowdhury, K.Y. Siu, T. Kailath. Classification of linearly nonseparable patterns by linear threshold elements. *IEEE Transactions on Neural Networks*, 6(2), 318–331, March 1995.
[25] M.A. Sartori, P.J. Antsaklis, A simple method to derive bounds on the size and to train multilayer neural networks. *IEEE Transactions on Neural Networks*, 2, 467–471, 1991.
[26] K.Y. Siu, V.P. Roychowdhury, T. Kailath. *Discrete Neural Computation*. Englewood Cliffs, NJ: Prentice Hall, 1995.
[27] F.W. Smith. Pattern classifier design by linear programming. *IEEE Transactions on Computers*, C-17(4), 367–372, April 1968.
[28] S. Tamura, T. Masahiko. Capabilities of a four-layered feedforward neural network: four layer versus three. *IEEE Transactions on Neural Networks*, 8, 251–255, 1997.
[29] J.T. Tou, R.C. Gonzalez. *Pattern Recognition Principles*. Reading, MA:Addison-Wesley, 1974.
[30] A.W. Tucker. Dual systems of homogeneous linear relations. In H.W. Kuhn, A.W. Tucker, eds., *Linear Inequalities and Related Systems*. Annals of Mathematics Studies Number 38, Princeton, NJ: Princeton University Press, pages 3–18, 1956.
[31] V.N. Vapnik, A.J. Chervonenkis. *Theory of Pattern Recognition* (in Russian). Nauka, Moscow, 1974; German translation: W.N. Wapnik, A.J. Tschervonenkis. *Theorie der Zeichenerkennung*. Berlin: Akademia, 1979.
[32] V.N. Vapnik. *Statistical Learning Theory*. New York: Wiley, 1998.
[33] B. Widrow, M.E. Hoff, Jr. *Adaptive switching circuits*. Tech. Report 1553-1, Stanford Electronics Labs, Stanford, CA, 1960.

[34] B. Widrow, M.A. Lehr. 30 years of adaptive neural networks: perceptron, madeline, and backpropagation. *Proceedings of the IEEE*, 78, 1415–1442, 1990.

[35] B. Widrow, S.D. Stearns. *Adaptive Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1985.

[36] M.H. Wright, Interior methods for constrained optimization. In A. Iserles, ed. *Acta Numerica*, pages 341–407, Cambridge:Cambridge University Press, 1992.