The Linear Separability Problem: Some Testing Methods

D. Elizondo

***Abstract*—The notion of linear separability is used widely in machine learning research. Learning algorithms that use this concept to learn include neural networks (single layer perceptron and recursive deterministic perceptron), and kernel machines (support vector machines). This paper presents an overview of several of the methods for testing linear separability between two classes. The methods are divided into four groups: Those based on linear programming, those based on computational geometry, one based on neural networks, and one based on quadratic programming. The Fisher linear discriminant method is also presented. A section on the quantification of the complexity of classification problems is included.**

***Index Terms*—Class of separability, computational geometry, convex hull, Fisher linear discriminant, linear programming, linear separability, quadratic programming, simplex, support vector machine.**

I. Introduction

TWO SUBSETS X and Y of $\mathbb{R}^d$ are said to be linearly separable (LS) if there exists a hyperplane P of $\mathbb{R}^d$ such that the elements of X and those of Y lie on opposite sides of it. Fig. 1 shows an example of LS and NLS set of points. Squares and circles denote the two classes.

Linear separability is an important topic in the domain of machine learning and cognitive psychology. A linear model is rather robust against noise and most likely will not over fit. Multilayer non linear neural networks, such as the back propagation algorithm, work well for classification problems. However, as the experience of perceptrons has shown, there are many real life problems in which there is a linear separation. For such problems, using backpropagation is an overkill, with thousands of iterations needed to get to the point where linear separation can bring us fast. Furthermore, multilayer linear neural networks, such as the recursive deterministic perceptron (RDP) [1], [2] can always linearly separate, in a deterministic way, two or more classes (even if the two classes are not linearly separable). The idea behind the construction of an RDP is to augment the affine dimension of the input vector by adding to these vectors the outputs of a sequence of intermediate neurons as new components. Each intermediate neuron corresponds to a single layer perceptron and it is characterized by a hyperplane which linearly separates an LS subset, taken from the non-LS (NLS) original set, and the remaining points in the input vector. A cognitive psychology study on the subject of linear separability constraint on category learning is presented in [3]. The authors stress the fact that grouping objects into categories on the basis of their similarity is a primary cognitive task. To the extent that categories are not linearly separable, on the basis of some combination of perceptual information, they might be expected to be harder to learn. Their work tries to address the issue of whether categories that are not linearly separable can ever be learned.

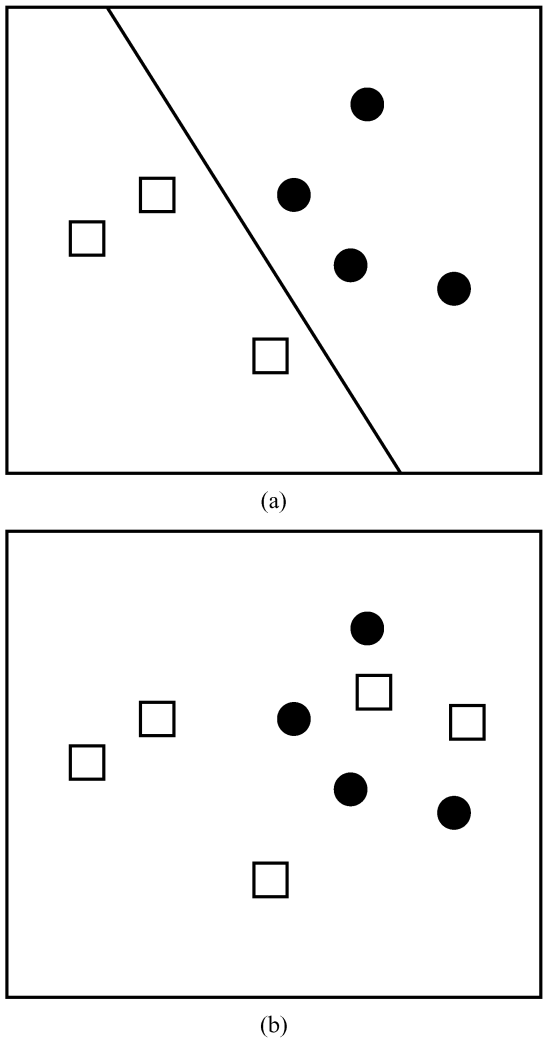

Fig. 1. (a) LS set of points. (b) Non-LS set of points.

Linear separability methods are also used for training support vector machines (SVM) used for pattern recognition [4], [5]. Similar to the RDP, SVMs are linear learning machines on LS

Manuscript received July 24, 2003; revised June 15, 2005.

The author is with the Centre for Computational Intelligence, School of Computing, De Montfort University, The Gateway, Leicester LEI 9BH, U.K. (e-mail: elizondo@dmu.ac.uk).

Digital Object Identifier 10.1109/TNN.2005.860871

and NLS data. They are trained by finding a hyperplane that linearly separates the data. In the case of NLS data, the data is mapped into some other Euclidean space. Thus, SVM is still doing a linear separation but in a different space.

This paper presents an overview of several of the methods for testing linear separability between two classes. In order to do this, the paper is divided into five sections. In Section II, some standard notations and definitions together with some general properties related to them are given. In Section III, some of the methods for testing linear separability, and the computational complexity for some of them are provided. These methods include those based on linear programming, computational geometry, neural networks, quadratic programming, and the Fisher linear discriminant method. Section IV deals with the quantification of the complexity of classification problems. Finally, some conclusions are pointed out in Section V.

II. Preliminaries

The following standard notions are used: Let $E, F \subset \mathbb{R}^d$. Card (E) stands for the cardinality of a set E. $E \backslash F$ is the set of elements which belongs to E and does not belong to F. $E \oplus F$ is the set of elements of the form $\vec{e} + \vec{f}$ with $\vec{e} \in E$ and $\vec{f} \in F$. $E \ominus F$ stands for $E \oplus -(F)$, i.e., the set of elements of the form $\vec{e} - \vec{f}$ with $\vec{e} \in E$ and $\vec{f} \in F$. If $E = \{(1,2),(-1,2.5)\}$ and $F = \{(2.2,3),(3.1,1)\}$, then $E \ominus F$ corresponds to $\{(-1.2,-1),(-2.1,1),(-3.2,-0.5),(-4.1,1.5)\}$. Let $\vec{p}_1, \vec{p}_2$ be the standard position vectors representing two points P_1 and P_2 in $\mathbb{R}^d$; the set $\{t\vec{p}_1 + (1-t)\vec{p}_2 | 0 \leq t \leq 1\}$ is called the segment between $\vec{p}_1, \vec{p}_2$ and is denoted by $[\vec{p}_1, \vec{p}_2]$. The dot product of two vectors $\vec{u} = (u_1, \ldots, u_d), \vec{v} = (v_1, \ldots, v_d)$ is defined as $\vec{u}^T\vec{v} = u_1v_1 + \cdots + u_dv_d$. $\mathrm{Adj}(\vec{u}, r) = (u_1, \ldots, u_d, r)$ and by extension $\mathrm{Adj}(S, r) = \{\mathrm{Adj}(\vec{x}, r) \,|\, \vec{x} \in S\}$. $\mathcal{P}(\vec{w}, t)$ stands for the hyperplane $\{\vec{x} \in \mathbb{R}^d \,|\, \vec{w}^T\vec{x} + t = 0\}$ of $\mathbb{R}^d$ of the normal $\vec{w}$, and the threshold t. $\mathbb{P}$ will stand for the set of all hyperplanes of $\mathbb{R}^d$.

The fact that two subsets X and Y of $\mathbb{R}^d$ are linearly separable is denoted by $X \,\|\, Y$ or $X \,\|\, Y(P)$. Thus, if $X \,\|\, Y(\mathcal{P}(\vec{w}, t))$, then $(\forall \vec{x} \in X, \vec{w}^T\vec{x} + t > 0$ and $\forall \vec{y} \in Y, \vec{w}^T\vec{y} + t < 0)$ or $(\forall \vec{x} \in X, \vec{w}^T\vec{x} + t < 0$ and $\forall \vec{y} \in Y, \vec{w}^T\vec{y} + t > 0)$. Let $X, Y \subset \mathbb{R}^d, \mathbb{P}(X, Y) = \{P \in \mathbb{R} \,|\, X \,\|\, Y(P)\}$. Let $P \in \mathbb{P}(X, Y), \mathcal{C}_Y(X, P)$ be the half space delimited by P and containing X (i.e., $\mathcal{C}_Y(X, P) = \{\vec{v} \in \mathbb{R}^d \,|\, \vec{u}^T\vec{v} + t > 0\}$ if for some $\vec{x} \in X, \vec{u}^T\vec{x} + t > 0$).

A. Definitions and Properties

For the theoretical representations, column vectors are used to represent points. However, for reasons of saving space, where concrete examples are given, row vectors are used to represent points. We also introduce the notions of convex hull and linear separability.

Definition 2.1: A subset D of $\mathbb{R}^d$ is said to be *convex* if, for any two points $\vec{p}_1$ and $\vec{p}_2$ in D, the segment $[\vec{p}_1, \vec{p}_2]$ is entirely contained in D.

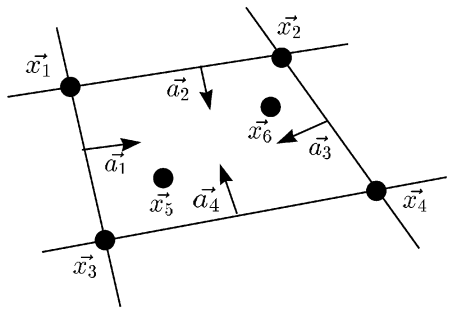

Fig. 2. Convex hull of a set of six points.

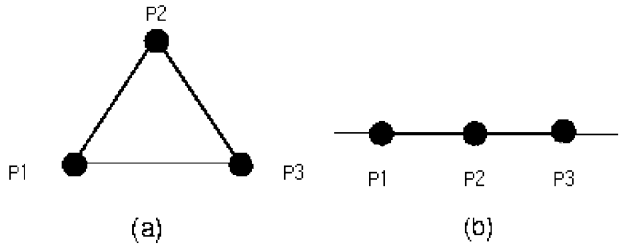

Fig. 3. (a) Affinely independent and (b) dependent set of points.

Definition 2.2: Let S be a subset of $\mathbb{R}^d$, then the convex hull of S, denoted by $\mathrm{CH}(S)$, is the smallest convex subset of $\mathbb{R}^d$ containing S.

Property 2.1: [6]: Let S be a subset of $\mathbb{R}^d$.

- $\mathrm{CH}(S) = \{t_1\vec{x}_1 + \cdots + t_k\vec{x}_k \,|\, \vec{x}_1, \ldots, \vec{x}_k \in S, t_1, \ldots, t_k \in [0,1]$ and $t_1 + \cdots + t_k = 1\}$.
- If S is finite, then there exists $\vec{a}_1, \ldots, \vec{a}_k \in \mathbb{R}^d$ and $b_1, \ldots, b_k \in \mathbb{R}$ such that $\mathrm{CH}(S) = \{\vec{x} \in \mathbb{R}^d \,|\, \vec{a}_i^T\vec{x} > b_i$ for $1 \leq i \leq k\}$. Thus, $\mathrm{CH}(S)$ is the intersection of k half spaces.

Fig. 2 represents the convex hull for a set of six points with a value of $k = 4$.

Property 2.2: If $X \,\|\, Y(P), X' \subset X$ and $Y' \subset Y$ then $X' \,\|\, Y'(P)$.

Definition 2.3: Let S be a subset of points in $\mathbb{R}^d$ and let $\vec{a} \in S'$; then $\mathrm{dimaf}(S)$ (dimension affine) is the dimension of the vectorial subspace generated by $\{\vec{x} - \vec{a} \,|\, \vec{x} \in S\}$.

In other words, $\mathrm{dimaf}(S)$ is the dimension of the smallest affine subspace that contains S. $\mathrm{Dimaf}(S)$ does not depend on the choice of $\vec{a} \in S$.

Definition 2.4: A set S of points is said to be affinely independent if $\mathrm{dimaf}(S) = \mathrm{Card}\,(S) - 1$.

In other words, given k points $P_1, P_2, \ldots, P_k$ in $\mathbb{R}^d$, they are said to be affinely independent if the $(k-1)$ vectors $P_2\overrightarrow{-}P_1, \ldots, P_k\overrightarrow{-}P_1$ are linearly independent. In an affinely dependent set of points, a point can always be expressed as a linear combination of the other points. Fig. 3 shows an example of an affinely independent and dependent set of points with coefficients summing to 1.

Property 2.3: If $S \subset \mathbb{R}^d$, then $\mathrm{dimaf}(S) < d$.

In other words, if we have a set of points in dimension d, the maximum number of affinely independent points that we can have is $d + 1$. There are no $d + 2$ affinely independent points;

so if we have for example 25 points in $\mathbb{R}^2$, these points are necessarily affinely dependent. If not all the points were placed in a straight line, we could find three points which are affinely independent.

III. Methods for Testing Linear Separability

In this section we present some of the methods for testing linear separability between two sets of points X and Y. These methods can be classified in five groups

1) methods based on linear programming;
2) methods based on computational geometry;
3) method based on neural networks;
4) method based on quadratic programming;
5) Fisher linear discriminant method.

A. Methods Based on Linear Programming

In these methods, the problem of linear separability is represented as a system of linear equations. If two classes are LS, the methods find a solution to the set of linear equations. This solution represents the hyperplane that linearly separates the two classes.

Several methods for solving linear programming problems exist. Two of the most popular ones are the Fourier–Kuhn elimination and the Simplex Method [7]–[10].

1) The Fourier–Kuhn Elimination Method: This method provides a way to eliminate variables from a set of linear equations. To demonstrate how the Fourier–Kuhn elimination method works, we will apply it to both the AND and the XOR logic problems.

a) The AND Problem: Let $X = \{(1,1)\}$ and $Y = \{(0,0),(1,0),(0,1)\}$ represent the input patterns for the two classes which define the AND problem. We want to find out if $X \,\|\, Y$.

Let

$$\begin{aligned} S &= \mathrm{Adj}(X,1) \cup -\mathrm{Adj}(Y,1) \\ &= \{(1,1,1),(0,0,-1),(0,-1,-1),(-1,0,-1)\}. \end{aligned}$$

By doing the above, a mapping of the two-dimensional (2-D) LS problem into a three-dimensional (3-D) LS problem is obtained where the separating plane passes through the origen. This mapping is done throughout the paper.

Thus, to find out if $X \,\|\, Y$, we need to find a set of values for the weights w_1, w_2, and threshold t such that

$$S = \begin{cases} w_1 + w_2 + t > 0, & C_1 \\ -t > 0, & C_2 \\ -w_2 - t > 0, & C_3 \\ -w_1 - t > 0, & C_4 \end{cases}.$$

The method consists of eliminating every time a variable from a set of inequalities. To do so, we consider all pairs of inequalities in which a variable has opposite signs, and eliminate it between each pair. Thus, if we choose to eliminate t (positive in C_1, and negative in the rest of the equations) we have

$$S' = \begin{cases} w_1 + w_2 + t > 0, & C_1 \\ w_1 + w_2 + t > 0, & C_1 + C_2 \\ w_1 > 0, & C_1 + C_3 \\ w_2 > 0, & C_1 + C_4 \end{cases}.$$

Since at this step all the variables have the same sign (they are all positive), we want to know if this new system of equations admits a solution which satisfies

$$w_1 \quad \text{and} \quad w_2 > 0 \quad \text{and} \quad 0 < -t < w_1 + w_2.$$

We can confirm that a set of values satisfying these constraints is: $w_1 = 2, w_2 = 2$, and $-t = 3$. Thus, we have proven that $X \,\|\, Y(\mathcal{P}((2,2),-3))$.

b) The XOR Problem: Let $X = \{(1,0),(0,1)\}$ and $Y = \{(0,0),(1,1)\}$ represent the input patterns for the two classes which define the XOR problem. We want to find out if $X \,\|\, Y$.

Let $X = \mathrm{Adj}(X,1), Y = \mathrm{Adj}(Y,1)$. We now have $S = X \cup -Y = \{(1,0,1),(0,1,1),(0,0,-1),(-1,-1,-1)\}$. Thus, to find out if $X \,\|\, Y$, we need to find a set of values for the weights w_1, w_2, and threshold t such that

$$S = \begin{cases} w_1 + t > 0, & C_1 \\ w_2 + t > 0, & C_2 \\ -t > 0, & C_3 \\ -w_1 - w_2 - t > 0, & C_4 \end{cases}.$$

As in the AND example, we choose to eliminate t (positive in C_1 and C_2, and negative in the rest of the equations). This produces

$$S' = \begin{cases} w_1 > 0, & C_1 + C_3 \\ -w_2 > 0, & C_1 + C_4 \\ w_2 > 0, & C_2 + C_3 \\ -w_1 > 0, & C_2 + C_4 \end{cases}.$$

At this point, we observe that inequalities 1 and 4 (2 and 3) contain a contradiction. Therefore, we conclude that the problem is infeasible, and that $X \not\| Y$.

c) Complexity: This method is computationally impractical for large problems because of the large build-up in inequalities (or variables) as variables (or constraints) are eliminated. Its computational complexity is exponential and can result in the worst case in $O(m^{2^n})$ constraints (for m inequalities involving n variables).

2) The Simplex Method: The Simplex method is one of the most popular methods used for solving linear programs. A linear program can be seen as a set of variables which are contained in a linear expression called the *objective function*. The goal is to find values to these variables which maximize or minimize the objective function subject to constraints. These constraints of linear expressions must be either $\leq, \geq,$ or $=$ to a given value. There are three possible results when trying to solve a linear program.

1) The model is *solvable*. This means that there exists a set of values for the variables that provide an optimal value to the objective function.
2) The model is *infeasible*. This means that there are no values for the variables which can satisfy all the constraints at the same time.
3) The model is *unbounded*. This means that the value of the objective function can be increased with no limit by choosing values to the variables.

In this paper, we use the Simplex algorithm for testing linear separability among two classes. The algorithms in Tables I and II show the Simplex procedure. This algorithm consists of finding the values of q and p to pivot and repeating the process until either an optimum value is obtained, or the linear program is determined to be infeasible. This method can be viewed

TABLE I
SIMPLEX ALGORITHM [11]

SIMPLEX($\mathcal{A}$)
– data: an array $\mathcal{A}$ of size **MxN** containing the constraints linear expressions the pivot column and row represented by p and q respectively.
– result: a solution if the constraints linear expressions are solvable, otherwise an *unbounded* or *infeasible* response.
Repeat
  $q := 0$;
  Repeat
    $q := q + 1$
  Until $(q = M + 1)$ **Or** $(a[0, q] < 0)$;
  $p := 0$;
  Repeat
    $p := p + 1$
  Until $(p = N + 1)$ **Or** $(a[p, q] > 0)$;
  For $i := p + 1$ **To** N **Do**
    If $a[i, q] > 0$
    Then
      If $((a[i, M + 1]/a[i, q]) < (a[p, M + 1]/a[p, q]))$
      Then
        $p := i$;
    If $(q < M + 1)$ **And** $(p < N + 1)$
    Then
      $pivot(p, q)$)
Until$(q = M + 1)$ **Or** $(p = N + 1)$

TABLE II
PIVOT PROCEDURE FOR THE SIMPLEX ALGORITHM [11]

procedure pivot(p,q)
– data: the row p and column q of the **MxN** array containing the constraints linear expressions on which to perform the pivoting
– result: a q column containing all values equal to zero except for a 1 in row q.
This is obtained by adding multiples of row p to each row as necessary.
  Begin
  For $j := 0$ **To** N **Do**
    For $k := M + 1$ **Downto** 1 **Do**
      If $(j <> p)$ **And** $(k <> q)$
      Then
        $a[j, k] := a[j, k] - a[p, k] * a[j, q]/a[p, q]$;
  For $j := 0$ **To** N **Do**
    If $j <> p$
    Then
      $a[j, q] := 0$;
  For $k := 0$ **To** $M + 1$ **Do**
    If $k <> q$
    Then
      $a[p, k] := a[p, k]/a[p, q]$;
  $a[p, q] = 1$;
  End

as a method for organizing the procedure so that a series of combinations of the equations is tried for which the objective function increases (maximization of the objective function) or decreases (minimization of the objective function) at each step, and the optimal feasible vector is reached after a number of iterations that is almost always no longer than the order of the number of equations or the number of independent variables, whichever is larger.

We illustrate the Simplex method by showing the linear program which corresponds to the transformation of the logical AND problem.

d) The AND Problem: Let $X = \{(1,1)\}$ and $Y = \{(0,0),(1,0),(0,1)\}$ represent the input patterns for the two classes which define the AND problem. We want to find out if $X \parallel Y$.

Let

$$S = \mathrm{Adj}(X,1) \cup -\mathrm{Adj}(Y,1)$$
$$= \{(1,1,1),(0,0,-1),(0,-1,-1),(-1,0,-1)\}.$$

Thus, to find out if $X \parallel Y$, we need to find a set of values for the weights w_1, w_2, and threshold t such that they minimize any of the given inequalities.

We can *minimize*: $w_1 + w_2 + t$

$$\text{subject to} = \begin{cases} w_1 + w_2 + t > 0 \\ -t > 0 \\ -w_2 - t > 0 \\ -w_1 - t > 0 \end{cases}.$$

Since the Simplex method limits the values of the variables to being ≥ 0, and a weight value can either be positive or negative, we transform each of our original variables as the difference of two variables. This transformation produces the following variables:

$$\begin{cases} w_1 = & w_{11\_\mathrm{aux}} - w_{12\_\mathrm{aux}} \\ w_2 = & w_{21\_\mathrm{aux}} - w_{22\_\mathrm{aux}} \\ t = & t_{1\_\mathrm{aux}} - t_{2\_\mathrm{aux}} \end{cases}.$$

In addition, since the Simplex method does not accept strict inequalities, we change our >0 conditions to ≥ 1.

Using the above transformations, our new set of constraints becomes as shown in the equation at the bottom of the page. By applying the simplex method, we obtain a feasible solution which gives the following result:

$$\begin{aligned} w_{11\_\mathrm{aux}} &= 2 \quad w_{12\_\mathrm{aux}} = 0 \\ w_{21\_\mathrm{aux}} &= 2 \quad w_{22\_\mathrm{aux}} = 0 \\ t_{1\_\mathrm{aux}} &= 0 \quad t_{2\_\mathrm{aux}} = 3. \end{aligned}$$

We can thus conclude that the problem is LS. By using these intermediate values we obtain the following values for our original set of variables: $w_1 = 2, w_2 = 2$, and $t = -3$. These variables form the actual hyperplane which separates the two classes.

$$\begin{cases} (w_{11\_\mathrm{aux}} - w_{12\_\mathrm{aux}}) + (w_{21\_\mathrm{aux}} - w_{22\_\mathrm{aux}}) + (t_{1\_\mathrm{aux}} - t_{2\_\mathrm{aux}}) \geq 1 \\ (-t_{1\_\mathrm{aux}} + t_{2\_\mathrm{aux}}) \geq 1 \\ (-w_{11\_\mathrm{aux}} + w_{12\_\mathrm{aux}}) - (t_{1\_\mathrm{aux}} + t_{2\_\mathrm{aux}}) \geq 1 \\ (-w_{21\_\mathrm{aux}} + w_{22\_\mathrm{aux}}) - (t_{1\_\mathrm{aux}} + t_{2\_\mathrm{aux}}) \geq 1 \end{cases}.$$

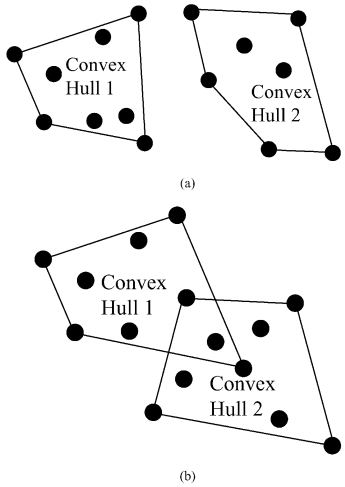

Fig. 4. Convex hulls for (a) LS and (b) non-LS set of points.

e) Complexity: One of the problems that the Simplex Method presents is that there is no known best strategy for selecting the pivot element because there are no results to tell us how many pivot steps to expect for any reasonable size problem.

The complexity of this method depends on the pivot rule used. A description of apparently all currently known bad examples for various pivot rules in a uniform way can be found in [12]. These rules include

- greatest increase rule;
- Dantzig(c)s largest coefficient rule;
- Bland(c)s least index rule;
- Gass-Saaty shadow vertex rule.

The largest number of vertices, and therefore, the maximal number of steps of the simplex algorithm with the worst possible pivot sequence on a linear programming problem in d fixed dimensions, and containing n facets, is equal to $O(n^{d/2})$.

B. The Methods Based on Computational Geometry Techniques

In this section, we discuss two methods for testing linear separability that are based on computational geometry techniques [13].

1) The Convex Hull Method for Testing Linear Separability: The following method makes use of the notion of convex hull. Two classes X and Y are LS if the intersection of the convex hulls of X and Y is empty. Fig. 4 presents the convex hulls for an LS set of points, with an empty intersection, and an NLS set of points, with a nonempty intersection.

Lemma 3.1: Let $X, Y \subset \mathbb{R}^d, X \parallel Y(P) \Leftrightarrow \mathrm{CH}(X) \| \mathrm{CH}(Y)(P)$.

Lemma 3.2: Let $X, Y \subset \mathbb{R}^d$ then $-\mathrm{CH}(X) = \mathrm{CH}(-X)$ and $\mathrm{CH}(X) \oplus \mathrm{CH}(Y) = \mathrm{CH}(X \oplus Y)$. Thus, $\mathrm{CH}(X \ominus Y) = \mathrm{CH}(X) \ominus \mathrm{CH}(Y)$.

Lemma 3.3: Let X, Y be two finite subsets of $\mathbb{R}^d, \mathrm{CH}(X) \| \mathrm{CH}(Y) \Leftrightarrow \mathrm{CH}(X) \cap \mathrm{CH}(Y) = \emptyset$.

Theorem 3.1: Let X, Y be two finite subsets of $\mathbb{R}^d$, then $X \parallel Y$ iff $\mathrm{CH}(X) \cap \mathrm{CH}(Y) = \emptyset$.

Property 3.1: Let S be a finite subset of $\mathbb{R}^d$, then there exists $\vec{x} \in S$ such that $S - \{\vec{x}\} \parallel \{\vec{x}\}$.

TABLE III
CONVEX HULL SEPARABILITY ALGORITHM

CHSA$(X, Y, \vec{w}, t)$
– data: two data set vectors, X and Y representing two classes
– result: a weight vector, $\vec{w}$ and a threshold value t which separates the two classes if $X \parallel Y$)
Begin
Compute $\vec{v}_1, ..., \vec{v}_k \in I\!R^d, b_1, ..., b_k \in I\!R$ such that $CH(X \ominus Y) = \{\vec{z} \in I\!R^n \mid \vec{v}_i^T \vec{z} \geq b_i \text{ for } 1 \leq i \leq k\}$.
If $(\forall i \leq k,\ b_i \leq 0)$ **Then** not$(X \parallel Y)$
Else
  Begin
    chose i such that $b_i > 0$;
    $\alpha := Max(\{\vec{v}_i^T \vec{y} \mid \vec{y} \in Y\})$;
    $\beta := Min(\{\vec{v}_i^T \vec{x} \mid \vec{x} \in X\})$;
    $\vec{w} := \vec{v}_i$;
    $t := -\frac{\beta+\alpha}{2}$;
    (i.e. $X \parallel Y\ (\mathcal{P}(\vec{w}, t))$ *)*
  End;
End

This property states that there exists a point $\vec{x} \in S$ which is LS from the rest of the points in S.

Remark: The Lemmas 3.1, 3.2, and 3.3 and the Property 3.1 are intuitively evident and their proofs are not relevant for the global comprehension of this work. Formal proofs can be found in [2]. A more general result than the one shown in Theorem 3.1 can be found in [14].

Property 3.2: Let X, Y be a finite subset of $\mathbb{R}^d$, and assume that $\mathrm{CH}(X \ominus Y) = \{\vec{x} \in \mathbb{R} \,|\, \vec{v}_x^T \leq b_i\}$ We have the following:

- if for all $ib_i \leq 0$, then $X \not\parallel Y$;
- if there exists i such that $b_i > 0$, then $X \| Y(\mathcal{P}(\vec{v}_i, t))$, where $t = -(\beta + \alpha/2), \alpha = \mathrm{Max}(\{\vec{v}_i^T \vec{y} \,|\, \vec{y} \in Y\})$, and $\beta = \mathrm{Min}(\{\vec{v}_i^T \vec{x} \,|\, \vec{x} \in X\}$.

Proof:

- If for all $ib_i \leq 0$, then $\vec{0} \in \mathrm{CH}(X \ominus Y)$ and $\mathrm{CH}(X) \cap \mathrm{CH}(Y) \neq \emptyset$; thus, by the Theorem 3.1, we conclude that $X \not\parallel Y$.
- If there exists i such that $b_i > 0$ then, for every $\vec{x} \in X$ and $\vec{y} \in Y$ we have $\vec{v}_i^T(\vec{x} - \vec{y}) \geq b_i > 0$, that is $\vec{v}_i^T \vec{x} > \vec{v}_i^T \vec{y}$. Let $\alpha = \mathrm{Max}(\{\vec{v}_i^T \vec{y} \,|\, \vec{y} \in Y\})$, and $\beta = \mathrm{Min}(\{\vec{v}_i^T \vec{x} \,|\, \vec{x} \in X\}$. It is clear that $\alpha < \beta$, and therefore

$$\forall \vec{y} \in Y \quad \vec{v}_i^T, \vec{y} - \frac{\alpha+\beta}{2} \leq \alpha - \frac{\alpha+\beta}{2} = \frac{\alpha-\beta}{2} < 0$$
$$\forall \vec{x} \in X \quad \vec{v}_i^T, \vec{x} - \frac{\alpha+\beta}{2} \geq \beta - \frac{\alpha+\beta}{2} = \frac{\beta-\alpha}{2} > 0.$$

We conclude that $X \parallel Y(\mathcal{P}(v_i, -(\alpha + \beta/2))$. ■

The convex hull separability algorithm presented in Table III is based in this property. Given $X, Y \subset \mathbb{R}^d$, this algorithm computes $w \in \mathbb{R}^d$ and $t \in \mathbb{R}$ (if they exist) such that $X \parallel Y(\mathcal{P}(\vec{w}, t))$, where w and t represent the weight vector and threshold value which make the hyperplane that linearly separates X and Y.

Following Property 3.2, the convex hull $\mathrm{CH}(XQ \ominus Y)$ is obtained as the set of points which verify the constraints $\{\vec{x} \in \mathbb{R}^n | \vec{v}_1^T \vec{x} \geq b_i \textbf{ for } 1 \leq i <\leq k\}$, where k is the number of vertices in the convex hull. This algorithm is applied in two steps.

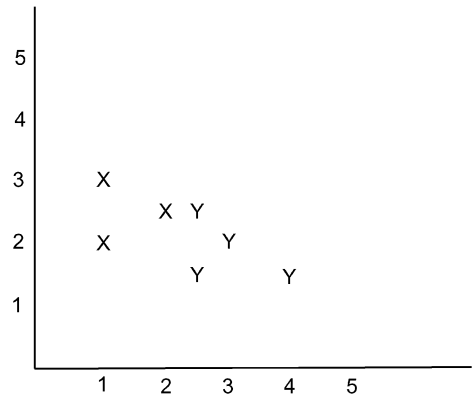

Fig. 5. XY PLOT of the 2-D classification problem.

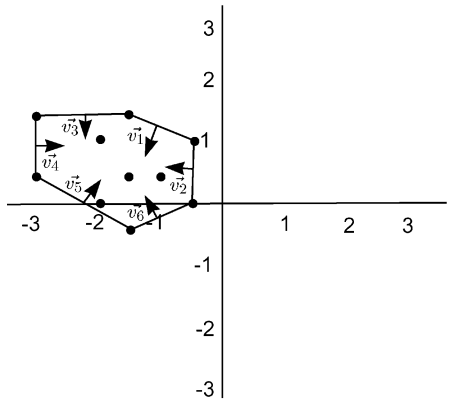

Fig. 6. Convex Hull of $X \ominus Y$ for the 2-D classification problem.

1) If for all i $b_i < 0$, then the two classes are NLS.

2) If there exists i such that $b_i > 0$, then the two classes are LS: With this value of b_i we compute the values of α and β which correspond respectively to the maximum value of the vector product of $\vec{v}_i^T \vec{y}$ with $\vec{y}$ belonging to the finite set of points of the second class Y, and the minimum value of the vector product of $\vec{v}_i^T \vec{x}$ with $\vec{x}$ belonging to the finite set of points of the first class X. These two values are then used to compute the threshold value t which is defined by $-(\beta + \alpha/2)$. The set of weights $\vec{w}$ that represents the hyperplane that linearly separates the two classes X and Y, corresponds to $\vec{v}_i$.

We will use the two-class, two-dimensional classification problem shown in Fig. 5 to illustrate the Convex Hull separability algorithm.

Let $X = \{(1,2),(2,2.5),(1,3)\}$ and $Y = \{(2.5,1.5),(2.5,2.5),(3,2),(4,1.5)\}$ represent the input patterns for the two classes which define these classification problem. We want to find out if $X \parallel Y$.

We calculate

$$\begin{aligned}(X \ominus Y) =\{&(-1.5,0.5),(-1.5,-0.5),(-2,0),(-3,0.5),\\ &(-0.5,1),(-0.5,0),(-1,0.5),(-2,1),\\ &(-1.5,1.5),(-1.5,0.5),(-2,1),(-3,1.5)\}.\end{aligned}$$

We then compute the convex hull of $(X \ominus Y)$

$$\begin{aligned}\mathrm{CH}(X \ominus Y) = \{&(-0.5,0),(-0.5,1),(-1.5,1.5),\\ &(-3,1.5),(-3,0.5),(-3,0.5)\}.\end{aligned}$$

Thus, we now know that $k = 6$(k represents the number of vertices in the CH). Next, we compute all the vectors $\vec{v}_k$ which are perpendicular to each of the facets that form the CH together with the corresponding values of b_k (see Fig. 6).

$$\begin{aligned}\vec{v}_1 &= (-1,0) \quad b_1 = 0.5\\ \vec{v}_2 &= (-1,-1) \quad b_2 = -0.5\\ \vec{v}_3 &= (0,-1) \quad b_3 = -1.5\\ \vec{v}_4 &= (1,0) \quad b_4 = -3.0\\ \vec{v}_5 &= (2,3) \quad b_5 = -4.5\\ \vec{v}_6 &= (-1,1) \quad b_6 = 0.5.\end{aligned}$$

We can now select any of the pairs $(\vec{v}_i, b_i)$ having a value $b_i > 0$. In our example, we have two choices for the value $i : 1$ and 6. Taking, $i = 1$ we have the following values of α and β:

$$\begin{aligned}\alpha &:= \mathrm{Max}(\{\vec{v}_i^T \vec{y} \,|\, \vec{y} \in Y\}) := -2.5\\ \beta &:= \mathrm{Min}(\{\vec{v}_i^T \vec{x} \,|\, \vec{x} \in X\} := -2.\end{aligned}$$

We can now compute the values of $\vec{w}$ and t

$$\begin{aligned}\vec{w} &:= \vec{v}_i := (-1,0)\\ t &:= -\frac{\beta+\alpha}{2} := 2.25.\end{aligned}$$

These weight vector and threshold values produce

Input P.	NetInput
$(1.0, 2.0)$	$(-1 \times 1.0) + (0 \times 2.0) + 2.25 = 1.25$
$(2.0, 2.5)$	$(-1 \times 2.0) + (0 \times 2.5) + 2.25 = 0.25$
$(1.0, 3.0)$	$(-1 \times 1.0) + (0 \times 3.0) + 2.25 = 1.25$
$(2.5, 1.5)$	$(-1 \times 2.5) + (0 \times 1.5) + 2.25 = -0.25$
$(2.5, 2.5)$	$(-1 \times 2.5) + (0 \times 2.5) + 2.25 = -0.25$
$(3.0, 2.0)$	$(-1 \times 3.0) + (0 \times 2.0) + 2.25 = -0.75$
$(4.0, 1.5)$	$(-1 \times 4.0) + (0 \times 1.5) + 2.25 = -1.75$

NetOutput	Class
$1.25 > 0 \Rightarrow +1$	X
$0.25 > 0 \Rightarrow +1$	X
$1.25 > 0 \Rightarrow +1$	X
$-0.25 \leq 0 \Rightarrow 1$	Y
$-0.25 \leq 0 \Rightarrow 1$	Y
$-0.75 \leq 0 \Rightarrow 1$	Y
$-1.75 \leq 0 \Rightarrow 1$	Y

Fig. 7 shows the hyperplane which linearly separates the two classes X and Y. Enclosed in a circle are the three points which represent α and β, the two values used to calculate the threshold value t.

There are other methods based on computational geometry that can be found in the literature for testing linear separability. Examples of this are the open hemisphere method [15], and the rotating calipers algorithm [16].

f) Complexity: A common algorithm used for computing convex hulls is the Quick-hull. The complexity of this algorithm is of $O(n \log n)$ for the average case, and of $O(n^2)$ for the worst case with an overall complexity similar to that of the Quick sort

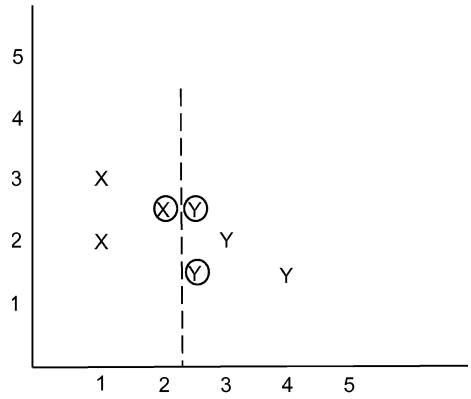

Fig. 7. Hyperplane that linearly separates the 2-D classification problem (dotted line).

TABLE IV
CLASS OF LINEAR SEPARABILITY METHOD

CLASSSEPAR$(X, Y, d, \vec{w}, t)$
– data: two data set vectors, X and Y representing two classes
– result: a weight vector, $\vec{w}$ and a threshold t which linearly separate the two classes if $X \,||Y$
Begin
If $h(X,Y) = \emptyset$ **Then** $not(X \,||\, Y)$
Else
Begin
Select $Z \in h(X,Y)$;
$ClassSepar(X \cap P_z, Y \cap P_z, d-1, \vec{w}', t')$;
$\varepsilon := Min(\{|\vec{w}_Z^T \vec{a} + t_Z| \,;$
$\vec{a} \in ((X \cup Y) \setminus \mathcal{P}(\vec{w}_Z, t_Z))\})$; $(\mathcal{P}_Z = \mathcal{P}(\vec{w}_Z, t_Z))$;
$\alpha := Max(\{|\vec{w}'^T \vec{a} + t'| \,;$
$\vec{a} \in ((X \cup Y) \setminus \mathcal{P}(\vec{w}_Z, t_Z))\})$;
$\vec{w} := \vec{w}_Z + \frac{\varepsilon}{2\alpha}\vec{w}'$;
$t := t_Z + \frac{\varepsilon}{2\alpha} t'$;
End
End

algorithm (these complexities are for the case of two dimensions). An in depth article concerning the complexity of different algorithms for computing the convex hull can be found in [17].

2) The Class of Linear Separability Method: In this section, we characterize the set of points of $\mathbb{R}^d$ by which passes a hyperplane that linearly separates two given LS classes. This characterization is going to allow us to infer the algorithm for testing linear separability described in Table IV.

g) Characterization of the Class of Linear Separability: Let X and Y be two finite and disjoint subsets of $\mathbb{R}^d$, and let $\varepsilon(X,Y)$ be the set of points by which passes a hyperplane that linearly separates X and Y. Clearly, $\varepsilon(X,Y) = \cup_{P \in \mathbb{P}}(X,Y)^P$.

Before characterizing the set $\varepsilon(X,Y)$, we give the following result which presents the linear separability property as a local property.

Proposition 3.1: Let X and Y be subsets of points of $\mathbb{R}^d$, then X and Y are linearly separable if and only if for every subset $M \subseteq X \cup Y$ containing at most $d+1$ points, $M \cap X$ and $M \cap Y$ are linearly separable.

After giving some specific notations, we will characterize the set $\varepsilon(X,Y)$.

Property 3.3: Let X, Y be finite subsets of $\mathbb{R}^d$ such that $X \parallel Y$, and let $\mathcal{C}(X,Y) = \mathbb{R}^d \backslash \varepsilon(X,Y)$. Then, $\mathcal{C}(X,Y) = \mathcal{C}_Y(X) \cup \mathcal{C}_X(Y)$ with $X \subset \mathcal{C}_Y(X), Y \subset \mathcal{C}_X(Y), \mathcal{C}_Y(X) \cap \mathcal{C}_X(Y) = 0$, and $\mathcal{C}_Y(X)$ and $\mathcal{C}_X(Y)$ are sets which are closed, unbounded, and convex. $\mathcal{C}_Y(X)$ is called a linear separability class of X relatively to Y. In other words, if we have the class of separabilities between two LS classes, we can take a point that does not belong to either of the two classes of separability, and we can always find a hyperplane that passes through this point and linearly separates the two classes. We characterize here all the points by which pass all the hyperplanes that linearly separate the two LS classes.

Proof: Let $\mathcal{P}(\vec{u},t) \;\varepsilon\; \mathbb{p}(X,Y)$, and note $C_Y(X) = \cap_{P \in \mathbb{IP}(X,Y)} \mathcal{C}X(Y,P), \mathcal{C}_X(Y) = \cap_{P \in \mathbb{P}(X,Y} \mathcal{C}_Y(X,P)$ then $\mathcal{C}_Y(X)$ and $\mathcal{C}_X(Y)$ are convex because $\forall P \in \mathbb{IP}(X,Y), \mathcal{C}_X(Y,P)$ and $\mathcal{C}_Y(X,P)$ are convex.

Moreover, if $\vec{x} \in \mathbb{R}^d \backslash \mathcal{C}(X,Y)$ then $\exists P \in \mathbb{IP}(X,Y)$ such that $\vec{x} \in P$. Let $\varepsilon = \text{Min}(d(X,P), d(Y,P)(\varepsilon > 0$ because $X \cap P = Y \cap P = \emptyset)$, hence $B(\vec{x},\varepsilon) = \{\vec{y} \in \mathbb{R}^d | \;||< \varepsilon\} \subset (\mathbb{R}^d \backslash \mathcal{C}(X,Y))$, thus $\vec{C}(X,Y)$ is a closed set in $\mathbb{R}^d$. Let $\vec{x} \in X$ and $\vec{y} \in Y$ and consider the straight line $D = \{s\vec{x} + (1-s)\vec{y} | s \in \mathbb{R}\}$ defined by $\vec{x}, \vec{y}$.

Assume that $C_Y(X)$ is bounded, then there exists $s_0 < 0$ such that for all $s < s_0, s\vec{x} + (1-s)\vec{Y} \notin \mathcal{C}Y(X)$. Let $s_1 < s_0$, then $s_1\vec{x} + (1-s_1)\vec{y} \in \mathcal{C}(X,Y)$ (because X $\parallel$ Y). Then, there exists $\vec{w}, t$ such that $s_1\vec{x} + (1-s_1)\vec{y} \in \mathcal{P}(\vec{w},t)$ and $X || Y(\mathcal{P}(\vec{w},t))$. Thus, $(\vec{w}^T\vec{x} + t)(\vec{w}^T\vec{y} + t) < 0$. Then, there exists $0 < s_2 < 1$ such that $s_2\vec{x} + (1-s_2)\vec{y} \in \mathcal{P}(\vec{w},t)$; hence, $\vec{x}, \vec{y} \in \mathcal{P}(\vec{w},t))$ which is absurd, thus $\mathcal{C}_Y(X)$. ■

Property 3.4: Let $X, Y \subset \mathbb{R}^d$ then $X \,||\, Y$; if there exists $\mathcal{P}(\vec{w},t)$ such that $\text{dimaf}((X \cup Y) \cap \mathcal{P}(\vec{w},t)) = d - 1[(X \backslash \vec{\mathcal{P}}(\vec{w},t)) || (Y \backslash \mathcal{P}(\vec{w},t))(\mathcal{P}(w,t))$ and $(X \cap \vec{\mathcal{P}}(\vec{w},t)) || (Y \cap \vec{\mathcal{P}}(\vec{w},t)))\}$ or $[(X \cup Y) \subset \mathcal{P}(\vec{w},t))$ and $X \parallel Y]$.

Proof:

1) Assume that there exists $\mathcal{P}(\vec{w},t)$ such that $\text{dimaf}((X \cup Y) \cap \mathcal{P}(\vec{w},t)) = d - 1; (X \backslash \mathcal{P}(\vec{w},t))$ $s || (Y \backslash \mathcal{P}(\vec{w},t))(\mathcal{P}(\vec{w},t))$ and $(X \cap \mathcal{P}(\vec{w},t)) || (Y \cap \vec{\mathcal{P}}(\vec{w},t))(\mathcal{P}(\vec{w}',t'))$ and assume that $\forall \vec{x} \in (X \backslash \mathcal{P}(\vec{w},t))$, $\vec{w}^T\vec{x} + t > 0$ and $\forall \vec{z} \in (X \cap \mathcal{P}(\vec{w},t)), \vec{w}'^T\vec{z} + t' > 0$. Let $\alpha = \text{Max}(\{\vec{w}^T\vec{z} + t'; \vec{z} \in ((X \cup Y) \backslash \mathcal{P}(\vec{w},t))\})$; and $\varepsilon = \text{Min}(\{(|\vec{w}^T\vec{z} + t|; \vec{z} \in ((X \cup Y) \backslash \mathcal{P}(\vec{w},t))\})$. Let $0 < \delta \leq (\varepsilon/2\alpha), \vec{w}^\delta = \vec{w} + \delta\vec{w}'$ and $t^\delta = t + \delta t'$. Then, by construction $X || Y(\mathcal{P}(\vec{w}^\delta, t^\delta)$.
2) Assume that $X \parallel Y(\mathcal{P}(\vec{w},t)$ and assume by induction that there exists $\mathcal{P}(\vec{w}_i,t_i)$ such that $(X \backslash \mathcal{P}(\vec{w}_i,t_i)) || (Y \backslash \mathcal{P}(\vec{w}_i,t_i))(\mathcal{P}(\vec{w}_i,t_i)), (X \cup Y) \backslash \mathcal{P}(\vec{w}_i,t_i) \neq \emptyset$ and $i \leq k = \text{dimaf}((X \cup Y) \cap \mathcal{P}(\vec{w}_i,t_i)) < d-1$. Let $\vec{z} \in (X \cup Y) \cap \mathcal{P}(\vec{w}_i,t_i)$ and $\{\vec{v}_1, \ldots, \vec{v}_{d-1}\}$. An orthogonal family (i.e., $\{\vec{v}_i, \ldots, \vec{v}_{d-1}\} \cup \{\vec{w}_i\}$ is an orthogonal basis of $\mathbb{R}^d$ such that $\mathcal{P}(\vec{w}_1,t_1) = \{\vec{z} + \lambda_i\vec{v}_i + \cdots + \lambda_{d-1}\vec{v}_{d-i}; \lambda_1, \ldots, \vec{v}_{d-1} \in \mathbb{R}^d$ and $(X \cup Y) \cap \mathcal{P}(\vec{w}_i,t_i) \subset \{\vec{z} + \lambda_1\vec{v}_1 + \cdots + \lambda k\vec{v}_k; \lambda_1, \ldots, \lambda_k \in \mathbb{R}\}(k < d-1)$. Two vectors $\vec{X} = (x_1, \ldots, x_d)$ and $\vec{Y} = (y_1, \ldots, y_d)$ are orthogonal if $\vec{X}^T\vec{Y} = 0$.

Assume that $(X \cup Y) \backslash \mathcal{P}(\vec{w}_i,t_i) = \{\vec{z}_1 \ldots \vec{z}_m\}$ and let $f_1 \ldots f_m$ be the continuous functions such that $f_j(\theta) =$

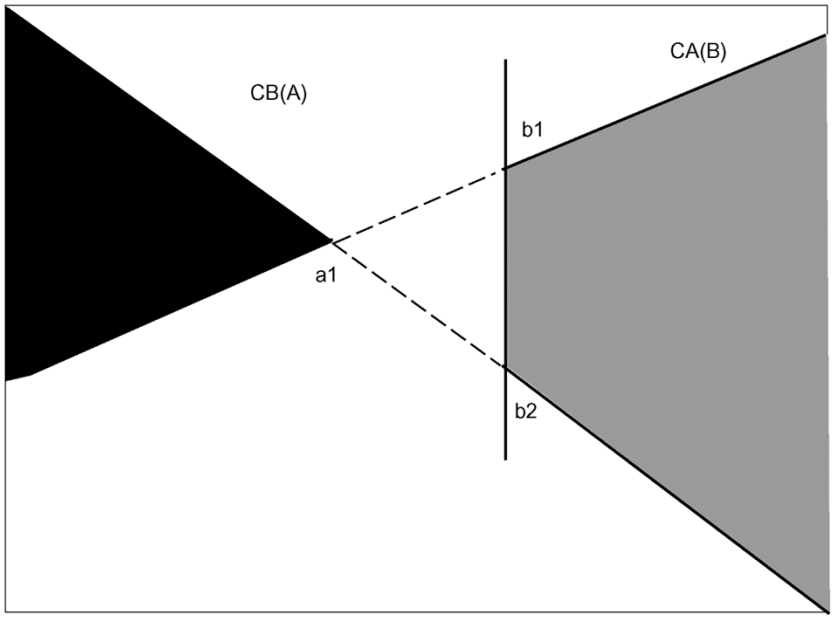

Fig. 8. Example of a class of linear separability.

$(\cos(\theta)\vec{w}_i + \sin(\theta)\vec{v}_{d-1})^T(\vec{z}_j - \vec{z})$ for $1 \leq j \leq m$. Then, $\forall j \leq m, \exists\theta_j \in]-)\pi/2), (\pi/2)[f_j(\theta j) = 0$, if $\vec{z}_j - \vec{z} = \lambda_1\vec{v}_1 + \cdots + \lambda_{d-1}\vec{v}_{d-1} + \lambda_d\vec{w}_i$, then $\theta_j = \arctan(-(\lambda_d|\vec{w}_i|^2/\lambda_{d-1}|\vec{v}_{d-1}|^2)$. Let $\theta_n = \mathrm{Min}(\{\theta_j; 1 \leq j \leq m\})$, $\vec{w}_{i+1} = \cos(\theta_n)\vec{w}_i + \sin(\theta_n)\vec{v}_{d-1}$ and $t_{i+1} = -w_{i+1}^T\vec{z}$, then $X\backslash\mathcal{P}(\vec{w}_{i+1}, t_{i+1}))\|(Y\backslash\mathcal{P}(\vec{w}_{i+1}, t_{i+1}))\mathcal{P}(\vec{w}_{i+1,t_i+1})$ and $\mathrm{dimaf}((X\cup Y)\cap\mathcal{P}(\vec{w}_i, t_i) < \mathrm{dimaf}((X\cup Y)\cap\mathcal{P}(\vec{w}_{i+1}, t_{i+1}))$, because $(X\cup Y)\cap\mathcal{P}(\vec{w}_i, t_i) \subset (X\cup Y)\cap\mathcal{P}(\vec{w}_{i+1}, t_{i+1}), z_n \in \mathcal{P}(\vec{w}_{i+1}, t_{i+1})$ and $z_n \notin \mathcal{P}(\vec{w}_i, t_i)$.

Hence, $\exists\vec{w} \in \mathbb{R}^d, \exists t \in \mathbb{R}((X \cup Y) \subset \mathcal{P}(\vec{w}, t)$ or $(\mathrm{dimaf}((X \cup Y) \cap \mathcal{P}(\vec{w}, t)) = d - 1$ and $(X\backslash\mathcal{P}(\vec{w}, t))\|(Y\backslash\mathcal{P}(\vec{w}, t))(\mathcal{P}(\vec{w}, t))$. ■

h) Class of Linear Separability Method: Let $\mathcal{P}$ be a hyperplane of $\mathbb{R}^d$. $X \subset \mathbb{R}^d, H_\mathcal{P}(X)$ is the half of $\mathbb{R}^d$ delimited by $\mathcal{P}$ and containing X. We assume that all points of X are in the same size of $\mathcal{P}_z$. Given $X, Y \subset \mathbb{R}^d$, the class of linear separability procedure, presented in the algorithm IV, computes recursively $\vec{w} \in \mathbb{R}^d$ and $t \in \mathbb{R}$. (*if there exists*) such that $X \parallel Y(\vec{\mathcal{P}}(\vec{w}, t))$.

Property 3.5: Let $X, Y \subset \mathbb{R}^d$, such that $\mathrm{dimaf}(X\cup Y) = d$ then, $\mathcal{C}_Y(X) = \cap_{Z\in h(X,Y)}H_{\mathcal{P}Z}(X)$ where

$$h(X,Y) = \begin{cases} \{Z \subset X \cup Y \mid \mathrm{Card}\,(Z) = d, \\ Z \text{ is a set of affinely independent} \\ \text{points and} \\ (X\backslash\mathcal{P}_z) \parallel (Y\backslash\mathcal{P}_z)(\mathcal{P}_z)\} \\ \text{if } \mathrm{dimaf}\,(X \cup Y) = d \\ \{P\} \text{ if } \mathrm{dimaf}\,(X \cup Y) < d, \\ \text{where } P \text{ is a hyperplance} \\ \text{containing } X \cup Y \end{cases}$$

The proof of the this property can be found in [2].

Fig. 8 presents an example of a class of linear separability. In this figure, we have two classes A and B having some hypothetical values $A = \{a_1\}$ and $B = \{b_1, b_2\}$. We then have that $C_B(A)$ represents the linear separability class of class A with respect to class B (black area), and $C_A(B)$ the linear separability class of class B with respect to class A (gray area).

To demonstrate how the class of linear separability method works, we will apply it to a two-class, two-dimensional classification problem. Let $X = \{(4,5), (8,6), (2,7), (6,7), (8,7), (5,8), (7,8), (6,9), (1,8)\}$ and $Y = \{(1,1), (3,1), (5,1), (7,2), (1,3), (6,3), (5,4)\}$ represent the input patterns for the two classes which define our problem. We want to find out if $X \parallel Y$. Fig. 9(a) shows a plot of the two classes. Following the algorithm, we want to identify all the hyperplanes with the following characteristics:

1) pass by one point of each of the two classes;

2) linearly separate the remaining points.

In this example, there is only one such hyperplane as illustrated in Fig. 9(b). This hyperplane is represented by $\mathcal{P}((1,1), -9)$. We now recursively reduce the original dimension of the problem to one dimension so as to locate all the points, by which the selected hyperplane passes in one line. Once this is done, we calculate the middle point between the two original points belonging to each of the classes. Next, we calculate a hyperplane in the original dimension that passes by this middle point and is different than the first hyperplane. This is illustrated in Fig. 9(c). The selected two points are highlighted with a circle ($\odot$ and $\otimes$). They correspond to the points (4,5), and (5,4) respectively. Thus the middle point has a value equal to 4.5. This second hyperplane is represented by $\mathcal{P}((1,-1),0)$. With these two hyperplanes we now compute the values of ε and α in the following way:

$$\varepsilon := \mathrm{Min}\left(\left\{\left|\vec{w}_Z^T\vec{a} + t_Z\right| \quad \vec{a} \in (A \cup B)\right\}\right) := 6$$
$$\alpha := \mathrm{Max}(\{|\vec{w}^T\vec{a} + t' \mid \quad \vec{a} \in (A \cup B)\}) := 4$$

where A and B correspond to the set of points by which the first hyperplane does not pass as illustrated in Fig. 9(d). We use these values for ε and α to calculate the final hyperplane which linearly separates the two classes in the following way:

$$\vec{w} := \vec{w}_Z + \frac{\varepsilon}{2\alpha}\vec{w}' := (-0.25, -1.75)$$
$$t := t_Z + \frac{\varepsilon}{2\alpha}t' := (9).$$

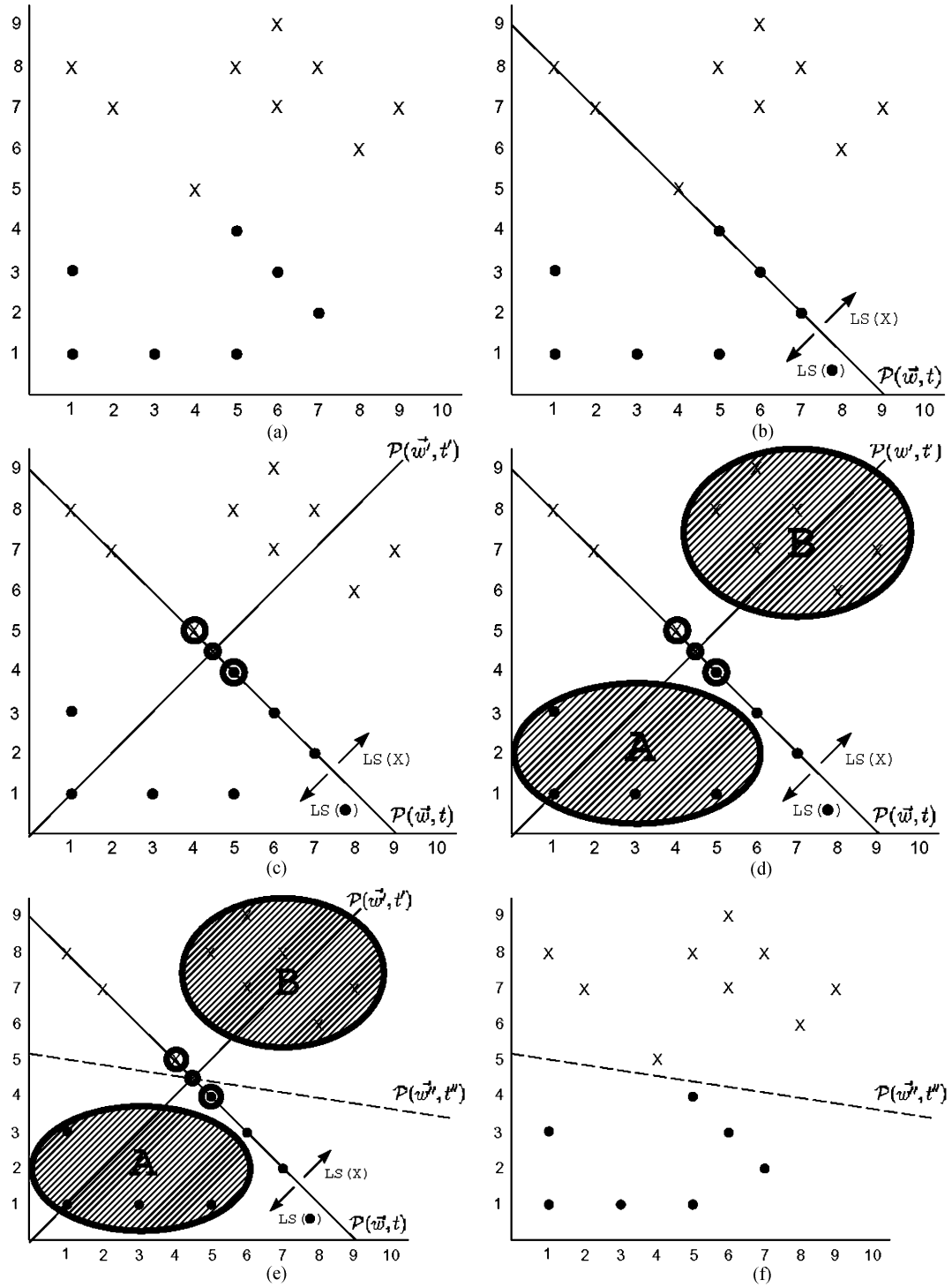

Fig. 9. Steps followed by the class of linear separability for finding the hyperplane which linearly separates classes X and $\bullet$.

This is illustrated in Fig. 9(e). The final hyperplane is represented by a dotted line and corresponds to $\mathcal{P}((-0.25,-1.75),9)$. This final hyperplane is illustrated in Fig. 9(f).

i) Complexity: This algorithm transforms recursively the problem of linear separability from d dimensions to $d-1$. This transformation may require $O(n^d)$ operations in the worse case. Thus, in order to simplify the search of this hyperplane, any hyperplane can be used to begin with, which can be moved in function of the points until it contains a set of d affinely independent points of the two classes, and that linearly separates the rest of the points.

C. The Methods Based on Neural Networks

1) The Perceptron Neural Network: The first neural network applied to the problem of separating patterns into two categories was proposed by McCulloch and Pitts [18]. This network computes the weighted sum of the input patterns and compares it to a threshold value T. If the net input is greater than the threshold,

TABLE V
PERCEPTRON LEARNING ALGORITHM

PERCETRON($S, \vec{w}$)
– data: a vector S containing the input and output data sets
– result: a set of weights $\vec{w}$ which separates the two classes if they are LS
$\vec{w}_0 := \vec{x}_0; (\vec{x}_0 \in I\!R^{d+1})$
$k := 0;$
While $(\exists \vec{x}_k \in S)$ such that $(\vec{w}_k^T \vec{x}_k \leq 0)$ **Do**
Begin
$\vec{w}_{k+1} := \vec{w}_k + \vec{x}_k;$
$k := k + 1;$
End

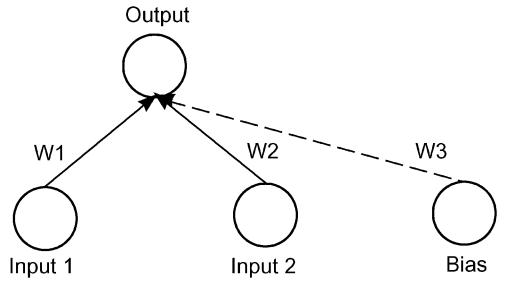

Fig. 10. Perception for a 2-D classification problem.

the output of the network is $+1$; otherwise, the output value is -1. This network is represented as follows:

$$\text{NetInput} = \sum_{i=1}^{d} w_i x_i$$

$$\text{NetOutput} = \begin{cases} +1, & \text{if NetInput} > \text{Threshold} \\ -1, & \text{if NetInput} < \text{Threshold} \end{cases}$$

where $(w_1, \ldots, w_d)$ and $(x_1, \ldots, w_d)$ correspond to the weight and input vector. The threshold value represents the minimal activity needed for the network to produce a positive output. Rosenblatt [19] transformed this algorithm into the first trainable neural network which he called the perceptron. Table V shows the pseudocode for the perceptron procedure.

Fig. 10 shows a perceptron network for a two-dimensional classification problem. We can take, as an example to illustrate the perceptron algorithm, the binary function AND.

Let $X = \{(1,1)\}$ and $Y = \{(0,0),(1,0),(0,1)\}$ represent the input patterns for the two classes, X and Y, which define the AND problem. We want to find a weight vector $\vec{w}$ and a threshold t such that $X||Y(\vec{\mathcal{P}}(\vec{w}, t))$. We note, from previous calculations, $\tilde{X} = \{(1,1,-1)\}, \tilde{Y} = \{(0,0,-1),(1,0,-1),(0,1,-1)\}$, and $S = \{\vec{x}_0, \vec{x}_1, \vec{x}_2, \vec{x}_3\} = \{(1,1,-1),(0,0,1),(-1,0,1),(0,-1,1)\}$.

Following the algorithm V, we have

$\vec{w}_0 = \vec{x}_0 = (1,1,-1)$
$k = 19$
$\vec{w}_k = \vec{w}_0 + \vec{x}_1 + \vec{x}_2 + \vec{x}_3 + \vec{x}_0 + \vec{x}_2 + \vec{x}_0 + \vec{x}_2 + \vec{x}_3 + \vec{x}_0 + \vec{x}_3$
$\vec{w}_k = \vec{w}_k + \vec{x}_0 + \vec{x}_2 + \vec{x}_0 + \vec{x}_2 + \vec{x}_3 + \vec{x}_0 + \vec{x}_3 + \vec{x}_0 + \vec{x}_2$
$\vec{w}_k = (8 \times \vec{x}_0) + (1 \times \vec{x}_1) + (6 \times \vec{x}_2) + (5 \times \vec{x}_3)$
$\vec{w}_k = (8,8,-8) + (0,0,1) + (-6,0,6) + (0,-5,5)$
$\vec{w}_k = (2,3,4).$

TABLE VI
ALPHA VALUES FOR THE QP SOLUTION TO THE 2-D CLASSIFICATION PROBLEM

Vector	Class	Alpha	SV
(-2.0,-1.0)	1	0	
(-1.0,-2.0)	1	0	
(0.0,-2.0)	1	0	
(-1.0, 0.0)	1	0	
(-2.0, 1.0)	1	0	
(-2.0, 2.0)	1	0	
(1.0, 2.0)	1	1.1111	*
(0.5, 2.0)	2	0.1667	*
(2.5,-2.0)	2	0.9444	*
(2.0, 1.0)	2	0	
(2.0, 2.0)	2	0	
(2.5, 1.0)	2	0	
(1.5, 1.0)	2	0	
(2.5, 0.0)	2	0	

Thus, $(X||Y)(\mathcal{P}(2,3),4))$. In other words, a set of values for the weight vector $\vec{w}$ and the threshold t that linearly separates the two classes X and Y are: $w_0 = 2, w_1 = 3$, and $t = 4$, respectively. These weight vector and threshold values produce

InputPattern	NetInput	NetOutput	Class
$(1,1)$	$(2 \times 1) + (3 \times 1) = 5$	$5 > 4 \Rightarrow +1$	X
$(0,0)$	$(2 \times 0) + (3 \times 0) = 0$	$0 < 4 \Rightarrow -1$	Y
$(1,0)$	$(2 \times 1) + (3 \times 0) = 2$	$2 < 4 \Rightarrow -1$	Y
$(0,1)$	$(2 \times 0) + (3 \times 1) = 3$	$3 < 4 \Rightarrow -1$	Y

j) Complexity: A complexity analysis of the perceptron algorithm can be found in [20]. The authors show that each iteration of the perceptron algorithm takes $O(\mathrm{md})$ time. Furthermore, under a Gaussian perturbation of variance σ^2 where $\sigma^2 \leq 1/2$ d, the perceptron algorithm converges in $O(-(d^3 m^2 \log^2(m-\delta))/\sigma^2\delta^2))$ iterations (if the problem is LS).

A geometry-based convergence upper bound for the perceptron learning algorithm is presented in [2]. This upper bound is based on the method described above for testing linear separability based on convex hulls.

D. Methods Based on Quadratic Programming

The training procedure for these methods consists in solving a quadratic programming optimization problem (QPOP). This is the case for the support vector machines (SVM). The SVM [21], [22] are linear learning machines that can be applied to classification problems involving both LS and NLS data sets [23], [24]. They can find a hyperplane that linearly separates a data set by solving a constrained QPOP.

In the case of NLS data, the data is mapped into some other Euclidean space; so that the SVM is still doing a linear separation but in a different space.

An on hands tutorial into classification using SVM can be found in [25]. The hyperplane separating two classes is defined as

$$\begin{cases} \vec{w}^T \vec{x}_i + b \geq 1 - \xi_i & y_i = 1 \\ \vec{w}^T \vec{x}_i + b \leq 1 + \xi_i & y_i = -1 \\ \xi_i \geq 0 & \forall_i \end{cases}$$

where ξ_j corresponds to slack variables in optimization theory and $\xi_i = 0$, if the there is no error for $\vec{x}_i$ (they will all be zero if the two classes are LS).

TABLE VII
PRODUCT OF THE DECISION REGION AND CLASS ASSIGNATION FOR EACH DATA POINT

Vector	Product	Decision	Class
(-2.0,-1.0)	(-1*(((1.3*(-2))+(0.6*(-1)))+4.3))-1=-2.0	-2.0 $\leq$0	1
(-1.0,-2.0)	(-1*(((1.3*(-1))+(0.6*(-2)))+4.3))-1=-2.6	-2.6 $\leq$0	1
(0.0,-2.0)	(-1*(((1.3*(0))+(0.6*(-2)))+4.3))-1=-4.0	-4.0 $\leq$0	1
(-1.0, 0.0)	(-1*(((1.3*(-1))+(0.6*(0)))+4.3))-1=-4.0	-4.0 $\leq$0	1
(-2.0, 1.0)	(-1*(((1.3*(-2))+(0.6*(1)))+4.3))-1=-3.3	-3.3 $\leq$0	1
(-2.0, 2.0)	(-1*(((1.3*(-2))+(0.6*(2)))+4.3))-1=-4.0	-4.0 $\leq$0	1
(1.0,-2.0)	(-1*(((1.3*(1))+(0.6*(-2)))+4.3))-1=-5.3	-5.3 $\leq$0	1
(0.5, 2.0)	(1*(((1.3*(0.5))+(0.6*(2)))+4.3))-1= 5.3	5.3 $\geq$0	2
(2.5,-2.0)	(1*(((1.3*(2.5))+(0.6*(-2)))+4.3))-1=5.3	5.3 $\geq$0	2
(2.0, 1.0)	(1*(((1.3*(2))+(0.6*(1)))+4.3))-1= 6.6	6.6 $\geq$0	2
(2.0, 2.0)	(1*(((1.3*(2))+(0.6*(2)))+4.3))-1= 7.3	7.3 $\geq$0	2
(2.5, 1.0)	(1*(((1.3*(2.5))+(0.6*(1)))+4.3))-1= 7.3	7.3 $\geq$0	2
(1.5, 1.0)	(1*(((1.3*(1.5))+(0.6*(1)))+4.3))-1= 6.0	6.0 $\geq$0	2
(2.5, 0.0)	(1*(((1.3*(2.5))+(0.6*(0)))+4.3))-1= 6.6	6.6 $\geq$0	2

For a training data set with input vector $\vec{x}_i \in R^n, i = 1, \ldots, l$ and output value $y_i, y \in \{1, -1\}$ the SVM needs to find a solution to the optimization problem

$$\text{Minimize: } \frac{1}{2}||\vec{w}||^2 = C\sum_{i=1}^{l}\xi i$$
$$\text{subject to: } y_i(\vec{w}^T\vec{x}_i + b) \geq 1 - \xi i, \xi i \geq 0$$

where $C > 0$ represents the penalty parameter between the error term and the margin of the hyperplane.

We can represent this problem in its dual form as

$$\text{Maximize: } W(\alpha) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1,j=1}^{n}\alpha_i\alpha_j y_i y_j \vec{x}_i^T\vec{x}i$$
$$\text{subject to: } \alpha_i \geq 0, \quad \sum_{i=1}^{n}\alpha_i y_i = 0.$$

A kernel function must be used on the SVM. In its simplest form, a kernel function calculates the dot product of two training vectors. This helps with the evaluation of the correct classification of each training vector. Some of the most common kernels used include

- linear: $K(\vec{x}_i, \vec{x}_j) = \vec{x}_i^T\vec{x}_j$;
- polynomial: $K(\vec{x}_i, \vec{x}_j) = (\gamma\vec{x}_i^T\vec{x}_j + r)^d, \gamma > 0$;
- radial basis function: $K(\vec{x}_i, \vec{y}_j) = \exp(-\gamma \; || \vec{x}_i - \vec{x}_j)||^2), \gamma > 0$;
- sigmoid: $K(\vec{x}_i, \vec{x}_j) = \tanh(\gamma\vec{x}_i^T\vec{x}_j + r)$;

where γ, r, and d are kernel parameters.

To illustrate how the SVM works, we will apply it to the following 2-D LS set of input vectors:

$$\text{Class 1} = \{(-2,-1),(-1,-2),(0,-2),(-1,0),(-2,1),(-2,2),(1,2)\}$$
$$\text{Class 2} = \{(0.5,2),(2.5,-2),(2,1),(2,2),(2.5,1),(1.5,1),(2.5,0)\}$$

which represent the input vector for the two LS classes. Let

$$y = \{-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1\}$$

represent the output value (-1 Class 1, and 1 Class 2). We want to find out if Class 1||Class 2.

To solve this problem we need to solve the following optimization problem:

$$\text{Maximize: } W(\alpha) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1,j=1}^{n}\alpha_i\alpha_j y_i y_j \vec{x}_i^T\vec{x}i$$
$$\text{subject to: } \alpha_i \geq 0, \quad \sum_{i=1}^{n}\alpha_i y_i = 0.$$

This allows to find the Lagrange multipliers α_i associated with the classification constrains.

Table VI shows the resulting values of α_i obtained by using a QP solver. The support vectors are the vectors with $\alpha_i \neq 0$.

We can now use the Lagrange multipliers to obtain the value of the hyperplane that linearly separates the two classes

$$\vec{w} = \sum_{i=1}^{14}\alpha_i y_i \vec{x}_i = -\alpha_7 x_7 + \alpha_8 x_8 + \alpha_9 x_9 = (1.3333, 0.6667).$$

The value of w_0 (the bias) can be obtained by

$$w_0 = 1 - (\vec{w}^T\vec{x}_i).$$

If we choose x_1, then we obtain $w_0 = 4.33$. Using these two values, our decision boundary becomes

$$y_i(\vec{w}^T\vec{x}_i + w_0) - 1 \leq 0 \text{ for class 1}$$
$$y_i(\vec{w}^T\vec{x}_i + w_0) - 1 \geq 0 \text{ for class 2.}$$

Table VII shows the output values obtained by applying the decision boundary to the classification points.

Fig. 11 shows a plot of the input vectors together with the hyperplane that linearly separates them in two classes. The support vectors α_7, α_8, and α_9 are also displayed.

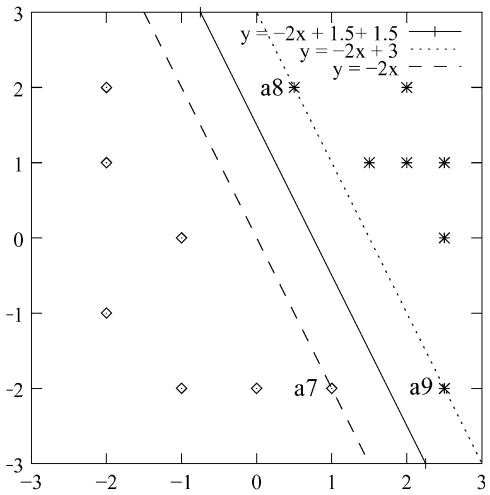

Fig. 11. SVM for a 2-D classification problem.

k) Complexity: Training an SVM requires the solution of a very large QPOP. This can be very slow, especially for large data sets. The complexity of a QPOP depends on the training data, its size, and its dimensions. There is no known method to define data complexity analytically. A fast method for training SVMs called sequential minimal optimization is described in [26]. The algorithm uses a divide and conquer approach by splitting the QPOP into smaller problems and solving these problems analytically.

E. Fisher Linear Discriminant

Let X_1 and X_2 represent a two-class classification problem, where $X_1 = \{x_1^1, x_2^1, \ldots, x_{l_1}^1\}$ is a set of patterns belonging to class C_1 and $X_2 = \{x_1^2, x_2^2, \ldots, x_{l_2}^2\} x_i^j \in \mathbb{R}^d$ is a set of patterns belonging to class C_1. The Fisher linear discriminant (FLD) [27] tries to find a linear combination of input variables, $w \times x$ which maximizes the average separation of the projections of the points belonging to the two classes C_1 and C_2 while minimizing the within class variance of the projections of those points. The Fisher discriminant is given by the vector w that maximizes

$$J(\vec{w}) = \vec{w}^T S_B \vec{w} / \vec{w}^T S W \vec{w}'$$

where S_B is the between class scatter matrix $S_B = (m_1 - m_2)(m_1 - m_2)^T$, and S_W the within class scatter matrix

$$S_W = \sum_{i \in \{1,2\}} \sum_{j=1}^{l_i} \left(x_j^i - m_i\right)\left(x_j^i - m_i\right)^T.$$

with $m_i = l{-}_i{}^1 \sum_{j=0}^{l_i} x_j^i$.

The parameter vector $\vec{W}$ of the linear classifier

$$q(x) = 0, \quad \text{for } \vec{W}' \times \vec{x} + b \geq 0$$
$$q(x) = 1, \quad \text{for } \vec{W}' \times \vec{x} + b < 0$$

is computed to maximize class separability criterion. The bias b is determined to lie between means of training data projected onto direction W.

To illustrate how the FLD works, we will apply it to the following 2-D LS set of points. Let Class 1 $x_1 = (-1.0, -0.2)$, $x_2 = (2.3, 0.8), x_3 = (-8.7, -3.6), x_4 = (-5.1, 2.5), x_5 = (-9.3, -2.9), x_6 = (-8.2, -3.5)$ and Class 2 $x_7 = (-1.4, -12.2), x_8 = (2.0, -4.6), x_9 = (-3.4, -11.3), x_{10} = (-2.1, -8.1)$ represent the input vector for the two LS classes. Let $y = \{-1, -1, -1, -1, -1, -1, 1, 1, 1, 1\}$ represent the output values –1 Class 1, and 1 Class 2). We want to find out if Class 1||Class 2.

To solve this problem we need to compute means and scatter matrix for each class:
mean Class 1:

$$m_1 = [-5.0259, -1.1707]$$

mean Class 2:

$$m_2 = [-1.2486, -9.1099]$$

scatter matrix Class 1:

$$S_1 = [113.0600, 42.8267, 42.8267, 33.7694]$$

scatter matrix Class 2:

$$S_2 = [16.2444, 19.2303, 19.2303, 35.2001].$$

The final scatter matrix is the sum of the two scatter matrixes

$$S_w = S_1 + S_2$$
$$S_w = [129.3043, 62.0569, 62.0569, 68.9695].$$

Next, we compute the weight vector of the hyperplane that separates the two classes using the inverse of the square matrix S_w and the means for the two classes

$$\vec{W} = \text{inv}(S_w) * (m_1 - m_2)$$
$$\vec{W} = [-0.1486, 0.2489].$$

Finally we compute the bias

$$\text{proj}_{m_1} = \vec{W}' * m_1 = 0.4558$$
$$\text{proj}_{m_2} = \vec{W}' * m_2 = -2.0815$$
$$\text{bias} = -0.5 * (\text{proj}_{m_1} + \text{proj}_{m_2}) = 0.8129.$$

Fig. 12 shows a plot of the input vectors together with the hyperplane that linearly separates them in two classes.

Table VIII shows the output values obtained by applying the decision boundary to the classification points.

IV. Quantification of the Complexity of Classification Problems

As well as studying the computational complexity of methods for testing linear separability, it is also interesting to look at ways of measuring the difficulty of a classification problem. Being able to quantify this complexity provides ways into better understanding of classification problems and classifier behavior.

A study presenting several measures to characterize the difficulty of a classification problem is presented in [28]. Their

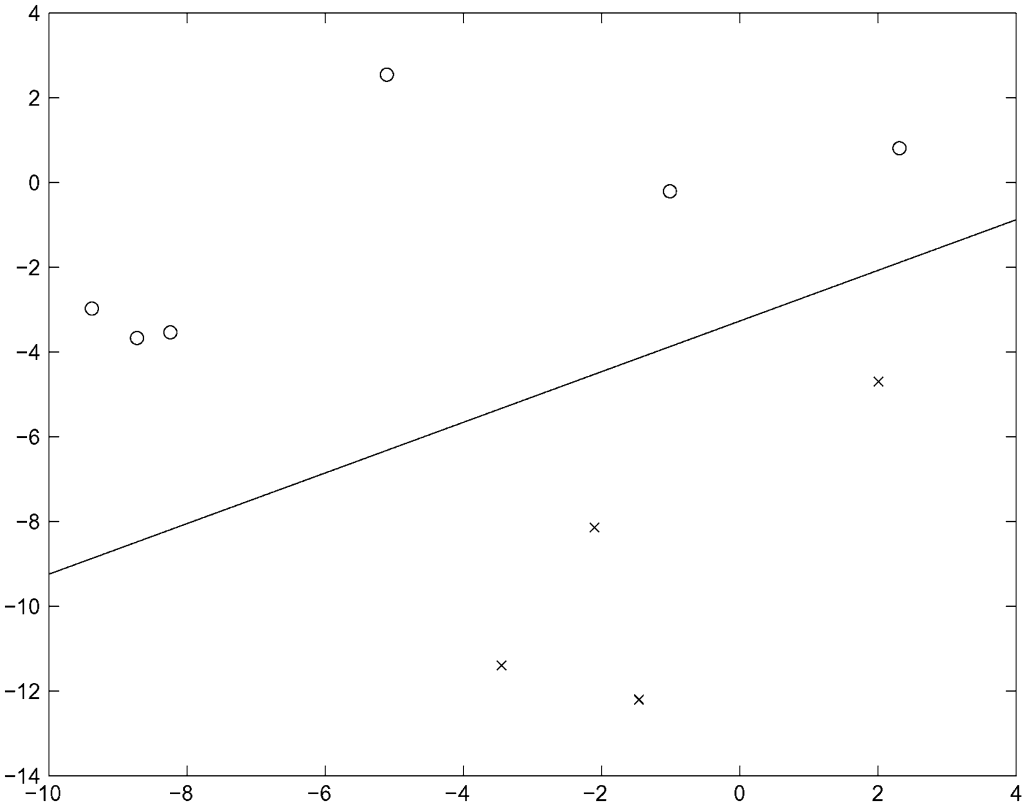

Fig. 12. Hyperplane for a 2-D classification problem using Fisher linear discriminant.

TABLE VIII
PRODUCT OF THE DECISION REGION AND CLASS ASSIGNATION FOR EACH DATA POINT

Vector	Product	Decision	Class
(-1.0,-0.2)	(-0.1486*-1.0) + (0.2489*-0.2) + 0.8129 = 0.9118	0.9118 $\geq$0	2
(2.3, 0.8)	(-0.1486*2.3) + (0.2489*0.8) + 0.8129 = 0.6705	0.6705 $\geq$0	2
(-8.7,-3.6)	(-0.1486*-8.7) + (0.2489*-3.6) + 0.8129 = 1.1977	1.1977 $\geq$0	2
(-1.4,-12.2)	(-0.1486*-1.4) + (0.2489*-12.2) + 0.8129 = -2.0079	-2.0079 $\leq$0	1
(2.0,-4.7)	(-0.1486*2.0) + (0.2489*-4.7) + 0.8129 = -0.6558	-0.6558 $\leq$0	1
(-3.4,-11.4)	(-0.1486*-3.4) + (0.2489*-11.4) + 0.8129 = -1.5105	-1.5105 $\leq$0	1
(-5.1, 2.5)	(-0.1486*-5.1) + (0.2489*2.5) + 0.8129 = 2.2063	2.2063 $\geq$0	2
(-9.4,-3.0)	(-0.1486*-9.4) + (0.2489*-3.0) + 0.8129 = 1.4673	1.4673 $\geq$0	2
(-8.2,-3.5)	(-0.1486*-8.2) + (0.2489*-3.5) + 0.8129 = 1.1583	1.1583 $\geq$0	2
(-2.1,-8.1)	(-0.1486*-2.1) + (0.2489*-8.1) + 0.8129 = -0.9004	-0.9004 $\leq$0	1

study focuses in the geometrical characteristics of the class distributions. The authors analyze measures that can emphasize the way in which classes are separated or interleaved, which play a role in the level of accuracy of classification. These measures include: overlap of individual feature values; separability of classes; and geometry, topology, and density of manifolds.

Another study on the measurement of the classifiability of instances of classification problems is presented in [29]. For this purpose, the authors propose a nonparametric method based on the evaluation of the texture of the class label surface. When the instances of a class are interlaced with another class, the surface is rough. The surface is smoother when the class regions are compact and disjoint. The texture of the class label surface is characterized by the use of a co-ocurrence matrix. They apply this approach to a look-ahead-based fuzzy decision tree induction that splits the instances of a particular node in such a way as to maximize the number of correct classifications at that node.

These characterization approaches can be used as another criteria from which to select the most adequate classifier for a specific problem.

V. DISCUSSION AND CONCLUDING REMARKS

Several of the existing methods for testing linear separability between two sets of points and the complexities associated with

TABLE IX
SUMMARY OF THE COMPUTATIONAL COMPLEXITIES OF SOME OF THE METHODS FOR TESTING LINEAR SEPARABILITY

Method	Complexity
Fourier Kuhn	$O(m^{2n})$
Simplex	$O(n^{d/2})$
Quick Hull	$O(n^{d-1}log(n))$
Class of Linear Separability	$O(n^d)$
Perceptron Algorithm	$O(\frac{d^3m^2log^2(m-\delta)}{\sigma^2\delta^2})$

some of the algorithms have been presented. The methods presented have been divided into four groups.

- The methods based on solving systems of linear equations. These methods include: the Fourier–Kuhn elimination algorithm, and the Simplex algorithm. The original classification problem is represented as a set of constrained linear equations. If the two classes are LS, the two algorithms provide a solution to these equations.
- The methods based on computational geometry techniques. We focused on the convex hull algorithm and the class of linear separability method. If two classes are LS, the intersection of the convex hulls of the set of points that represent the two classes is empty. The class of linear separability method consists in characterizing the set of points P of $\mathbb{R}^d$ by which it passes a hyperplane that linearly separates two sets of points X and Y.
- The methods based on neural networks. The method described in this section is the perceptron learning algorithm. If the two classes are LS, the perceptron algorithm is guaranteed to converge, after a finite number of steps, and will find a hyperplane that separates them.
- The methods based on quadratic programming. These methods can find a hyperplane that linearly separates two classes by solving a quadratic optimization problem. This is the case for the SVM.
- The Fisher linear discriminant method. This method tries to find a linear combination of input variables, $w \times x$, which maximizes the average separation of the projections of the points belonging to the two classes C_1 and C_2 while minimizing the within class variance of the projections of those points.

Table IX presents a summary of the complexities of some of the algorithms for testing linear separability described in this paper.

Other methods for testing linear separability include the Tarski elimination algorithm [30] which verifies the validity of a special first order formula in an algebraic closed field

$$\exists \vec{w} \in I\!R^d \exists t \in I\!R(\wedge_{\vec{x}\in X}\vec{w}^T\vec{x}+t>0)\wedge(\wedge_{\vec{y}\in Y}\vec{w}^T\vec{y}+t<0).$$

It is well known that the first-order logic for an algebraic closed field is decidable.

Several aspects should be considered when choosing a method for testing linear separability for a given problem. Some of these aspects include: the complexity level, the difficulty of the classification problem, the easiness of implementation, and the degree of linear separability that can be obtained when dealing with nonlinearly separable sets.

The Fourier-Khun elimination method is computationally impractical for large problems due to the large build-up in inequalities (or variables) as variables (or constraints) are eliminated. Its computational complexity is exponential and can result in the worst case in $O(m^{2n})$ constraints (for m inequalities involving n variables).

From a complexity point of view, the simplex method is remarkably efficient in practice and is guaranteed to find the global optimum. However, in some cases, this method has markedly varying response times. Most of the times it is extremely fast, but when a succession of pivots are required, it slows down considerably, given rise to a less stable iteration.

The linear separability algorithm based on the convex hull of the data sets is simple to implement in three or less dimensions. It becomes more difficult as the dimension of the problem augments. It is important to use effective storage techniques and take into account possible imprecision in measurement. One must decide how to store a convex hull, once found, which is especially tricky to do in higher dimensions. Hulls are generally stored as a list of the highest dimensional facets (facets in two dimensions), the neighboring facets for each facet, as well as the vertices associated with the facets. This allows for a way to add extra points to a computed hull.

The class of linear separability method can have a high computational complexity. However, this method cannot only be used for testing linear separability, but also for finding linearly separable subsets of maximum cardinality from within a nonlinearly separable set of points [31]. The concept of linearly separable subsets of maximum cardinality is used as the basis for constructing RDP multilayer linear networks.

From an implementation point of view, the perceptron neural network is probably one of the simplest algorithm to program. However, this algorithm is not very stable. There is also no way to know after how many weight updates one can conclude that if the algorithm has not converged, the problem at hand is not linearly separable. A convergence upper bound for this algorithm has been developed by Elizondo [2]. This bound remains hard to compute, and further work needs to be done before it could be of practical use.

The SVM method is an efficient learning algorithm. Some of its advantages include its ability to handle nonlinear classification problems that are nonseparable using linear methods; the handling of arbitrary complex classifications, and the flexibility and avoidance of overfitting. A key issue with this method is finding the right kernel that will map a nonlinearly separable data set into a higher dimension making it linearly separable. Training an SVM involves finding a solution to a very large QPOP. This can prove very slow for large data sets. The sequential minimal optimization method is a fast method for training SVMs. It uses a divide and conquer approach to minimize the training time.

The Fisher linear discriminant is a well-known method for testing linear separability. The method has proven very powerful and popular among users of discriminant analysis. Some reasons for this are its simplicity and unnecessity of strict assumptions. However, this method has optimality properties only if the underlying distributions of the groups are multivariate normal. The discriminant rule obtained can be seriously harmed by only

a small number of outlying observations. Outliers are hard to detect in multivariate data sets. The method is not very robust because it uses the sample means and variances, which can be affected by one sufficiently large point. Kernel variations of this method have been proposed that can be used for nonlinear pattern recognition.

All the methods above will provide a hyperplane only if the two classes are linear separable. Both the SVM and the Fisher linear discriminant will provide a hyperplane even if the two classes are nonlinearly separable.

Future directions in the area of linear separability include the optimization of the hyperplanes that linearly separate two classes in order to maximize the generalization level. In other words, once we know that two classes are linearly separable, there exists an infinite number of hyperplanes that can separate them linearly. How does one select the hyperplane that provides the highest level of generalization? This will involve a deep comparison study of the levels of generalization obtained with the different methods for testing linear separability, involving several real world data sets and benchmarks. Another interesting aspect to study is to include linear separability probabilities. This will make a set of points linearly separable within a certain probability making the decision regions less rigid. Most research involving linear separability has been conducted using tasks that involve learning two classes with a small number of data samples. It could be useful to do more research involving linear separability for more than two classes and larger data sets.

ACKNOWLEDGMENT

The author would like to thank the Associate Editor and two anonymous referees for all their detailed comments and suggestions that have clearly improved the final presentation of this paper. He would also like to thank especially to Dr. R. John, Dr. F. Chiclana, and R. Birkenhead from the CCI group at the School of Computing of De Montfort University for their feedback on this paper.

REFERENCES

[1] M. Tajine and D. Elizondo, "Enhancing the Perceptron Neural Network by Using Functional Composition," Comp. Sci. Dept., Univ. Louis Pasteur, Strasbourg, France, Tech. Rep. 96-07, 1996.

[2] D. A. Elizondo, "The Recursive Determinist Perceptron (rdp) and Topology Reduction Strategies for Neural Networks," Ph.D. dissertation, Univ. Louis Pasteur, Strasbourg, France, Jan. 1997.

[3] M. Blair and D. Homa, "Expanding the search for a linear separability constraint on category learning," *Memory and Cognition*, vol. 29, no. 8, pp. 1153–1164, 2001.

[4] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*. Cambridge, U.K.: Cambridge Univ. Press, 2003, vol. I.

[5] A. Atiya, "Learning with kernels: Support vector machines, regularization, optimization, and beyond," *IEEE Trans. Neural Netw.*, vol. 16, no. 3, pp. 780–781, May 2005.

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

[7] J. Fourier, Memoire de l'Academie Royale des Sciences de l'Institute de France, 7 (1824), xlvij-lv., 1827. Chez Firmin Didot Pere et Fils.

[8] H. W. Kuhn, "Solvability and consistency for linear equations and inequalities," *Amer. Math. Monthly*, vol. 63, pp. 217–232, 1956.

[9] M. Sakarovitch, *Optimization Combinatoire Graphe et Programmation Lineaire*. Paris, France: Hermann, 1984. Editeurs de Sciences et des Arts.

[10] M. S. Bazaraa and J. J. Jarvis, *Linear Programming and Network Flow*. London, U.K.: Wiley, 1977.

[11] R. Sedgewick, *Algorithms.*. Reading, MA: Addison-Wesley, 1983, pt. 38, p. 508.

[12] R. P. B. Chazelle and J. E. Goodman, *Deformed Products and Maximal Shadows.*: American Mathematical Soc., Mar. 1996.

[13] M. Tajine and D. Elizondo, "New methods for testing linear separability," *Neurocomput.*, vol. 47, no. 1–4, pp. 295–322, Aug. 2002.

[14] J. Stoer and C. Witzgall, *Convexity and Optimization Infinite Dimensions I*. Berlin, Germany: Springer-Verlag, 1970.

[15] D. S. Johnson and F. P. Preparata, "The densest hemisphere problem," *Theor. Comput. Sci.*, vol. 6, pp. 93–107, 1978.

[16] *Solving Geometric Problems With the Rotating Calipers*, May 1983.

[17] D. Avis and D. Bremner, "How good are convex hull algorithms," in *Proc. IEEE Symp. Computational Geometry* , 1995, pp. 20–28.

[18] W. McCulloch and W. Pitts, "A logical calculus of the ideas imminent in nervous activity," *Bull. Math. Biophys.*, vol. 5, pp. 115–133, 1943.

[19] F. Rosenblatt, *Principles of Neurodynamics*. Washington, D.C.: Spartan, 1962.

[20] A. Blum and J. Dunagan, "Smooth analysis of the perception algorithm," *Proc. 13th Annu. ACM-SIAM Symp. Discrete Algorithms*, pp. 905–914, 2002.

[21] *A Traning Algorithm for Optimal Margin Classifiers*, 1992.

[22] C. Cortes and V. Vapnik, "Support-vector network," *Mach. Learn.*, vol. 20, pp. 273–297, 1995.

[23] L. Ferreira, E. Kaszkurewicz, and A. Bhaya, "Solving systems of linear equations via gradient systems with discontinuous righthand sides: Application to ls-svm," *IEEE Trans. Neural Netw.*, vol. 16, no. 2, pp. 501–505, Mar. 2005.

[24] S. Pang, D. Kim, and S. Y. Bang, "Membership authentication using svm classification tree generated by membership-based lle data partition," *IEEE Trans. Neural Netw.*, vol. 16, no. 2, pp. 436–446, Mar. 2005.

[25] C. Hsu, C. Chang, and C. Lin, "Practical Guide to Support Vector Classification," National Taiwan Univ., Taipei 106, Taiwan, Tech. Rep., 2003.

[26] J. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods—Support Vector Learning*, B. Schlkopf, C. Burges, and A. Smola, Eds. Cambridge, MA: MIT Press, 1998.

[27] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annu. Eugenics*, vol. 7, no. II, pp. 179–188, Apr. 1936.

[28] T. K. Ho and M. Basu, "Complexity measures of supervised classifi—cation problems," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 3, pp. 289–300, Mar. 2002.

[29] M. Dong and R. Kothari, "Look-ahead based fuzzy decision tree induction," *IEEE Trans. Fuzzy Syst.*, vol. 9, no. 3, pp. 461–468, Jun. 2001.

[30] A. Tarski, "A decision method for elementary algebra and geometry," Univ. California Press, Berkeley and Los Angeles, Tech. Rep., 1954.

[31] D. Elizondo, "Searching for linearly separable subsets using the class of linear separability method," in *Proc. IEEE 2004 Int. Joint Conf. Neural Networks (IJCNN 04)* , vol. 2, Jul. 2004, pp. 955–959.

D. Elizondo received the B.Sc. degree in computer science from Knox College, Galesbourg, IL, in 1986, the M.Sc. degree in artificial intelligence from the University of Georgia, Athens, in 1992, and the Ph.D. degree in computer science from the Universite Louis Pasteur, Strasbourg, France, and the Institut Dalle Molle d'Intelligence Artificielle Perceptive (IDIAP), Martigny, Switzerland, in 1996.

He is currently a Senior Lecturer at the Centre for Computational Intelligence of the School of Computing at De Montfort University, Leicester, U.K. His research interests include applied neural network research, computational geometry approaches towards neural networks, and knowledge extraction from neural networks.