AN ABSTRACT OF THE THESIS OF

THOMAS J. MUELLER for the DOCTOR OF PHILOSOPHY
(Name) (Degree)

in Mathematics presented on October 26, 1972
(Major department) (Date)

Title: TEEMAL, AN ADAPTIVE TRAINING PROCEDURE FOR A TWO LAYER SYSTEM OF LINEAR THRESHOLD ELEMENTS

Abstract approved: Signature redacted for privacy.
Emilio Gagliardo

In many practical applications of learning systems to problems of pattern recognition it has been realized and explicitly noted in the literature that linear discriminations are inadequate. On the other hand, it has also been noted that very little is known about the training of non-linear systems.

A reasonable compromise between linearity and high complexity is what is called a 'committee machine,' i.e. a collection of linear systems each performing a linear threshold function (subject to adaptation) with an overall element (as the majority rule) to express the final diagnosis.

In this paper we will present a system of algorithms which effectively locates a committee machine which uses majority or veto logic. The algorithms are error-correction techniques, which in general perform as many adjustments in training as known algorithms, but with the added feature

that in some cases the algorithms will allow the machine to misjudge some samples which are deemed to be noisy or otherwise abnormal without implementing, in relation to these samples, significant change in the committee members.

Experimental results are presented using artificially generated data in 2-space, hand-printed letters A and R (Munson), disconnected-connected 3 x 3 arrays, absence-presence of the code 1101, and 3 x 3 quasi-randomly generated arrays (Michalski).

TEEMAL, AN ADAPTIVE TRAINING PROCEDURE FOR A TWO LAYER SYSTEM OF LINEAR THRESHOLD ELEMENTS

by

THOMAS J. MUELLER

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

DOCTOR OF PHILOSOPHY

June 1973

APPROVED:

Signature redacted for privacy.

Professor of Mathematics in charge of major

Signature redacted for privacy.

Chairman of Department of Mathematics

Signature redacted for privacy.

Dean of Graduate School

Date thesis is presented October 26, 1972

Typed by Linda Knuth for Thomas J. Mueller

TABLE OF CONTENTS

Chapter		Page
I.	INTRODUCTION	1
II.	BASIC CONCEPTS	3
	1. Pattern Representation	3
	2. A Threshold Locigal Unit	4
	3. The Committee Machine	6
III.	COMMITTEE MACHINES AND CLASSIFICATION	9
	1. Existence of a Majority Solution	9
	2. Conjectures on the Size and Logic of Committee Solutions	10
IV.	EXISTING ALGORITHM FOR LOCATING COMMITTEE MACHINES	14
	1. Error Correction Techniques	14
	2. Algorithms for Committees of Size 1	15
	3. An Algorithm for Locating a Machine of Size 1	17
V.	TEEMAL - A NEW METHOD FOR LOCATING A COMMITTEE	22
	1. Preliminary Remarks	22
	2. CREATE - A Method of Creating a New Hyperplane	23
	3. ALCONONE - An Algorithm for Adjusting One Hyperplane	25
	4. Algorithm I, ALCONTY	30
	5. ALVARY - A Second Algorithm for Adjusting All Hyperplanes	34
	6. TEEMAL, The Method of Locating the Machine	39
VI.	EXPERIMENTAL RESULTS	43
	1. Artificially Generated Data in 2-Space	43
	2. Hand-printed Letters A and R	59
	3. Disconnected-connected 4 x 4 Arrays	64

Chapter		Page
VI.	EXPERIMENTAL RESULTS (continued)	
	4. Absence-presence of Code 1101	68
	5. 3 x 3 Arrays of Michalski	69
	BIBLIOGRAPHY	70
	APPENDICES	72
	APPENDIX I	73
	APPENDIX II	87
	APPENDIX III	97

TEEMAL, AN ADAPTIVE TRAINING PROCEDURE FOR A TWO LAYER SYSTEM OF LINEAR THRESHOLD ELEMENTS

I. INTRODUCTION

This paper concerns itself with a system of linear discriminant functions (hyperplanes) which have two states corresponding to the classes in the two-class problem. Each function is given equal weight and the state of the system is determined by a pre-set percentage of the states of the functions called the logic of the system, i.e. the system judges a pattern to belong to a category if at least 'p' percent of the functions judge it so.

In our development of an adaptive, error-correction technique for locating such a system, we are aware of the following observations of Nilsson [15].

> "A disadvantage results from the fact that error correction rules never allow an error in pattern classification without making some adjustment in the discriminant functions. In many pattern classification tasks, it may be necessary to tolerate some small number of classification errors in the training set in order to classify related patterns with small probability of error."

Another problem that has been noted is the difficulty of an algorithm running through cyclic states or remaining stationary in some configuration, a static state. This has been evident when the data is presented to the algorithm in cyclic form [16] and particularly when the data consists of a small training set [4].

To overcome these handicaps we have devised a system of algorithms which by constant transfer of control from one to the other avoids static and cyclic states. Further by use of a new and effective error-correction criterion we have obtained an algorithm which allows for some errors in training, i.e. samples which the algorithm considers to be noisy or otherwise abnormal. In such cases the algorithm will not perform any adjustment of the discriminant functions.

This system provides the committee a great freedom of movement in the training phase, and yet the process arrives at solutions which are quite stable and represent a near optimal configuration for the given parameters, i.e. the number of functions and logic. Although this paper is totally devoted to systems with hyperplanes as their discriminant functions and judged on the percentage basis, it is invisioned that some of the basic ideas of error-correction may be extended to train other systems of discriminant functions.

II. BASIC CONCEPTS

1. Pattern Representation.

In describing patterns for the pattern recognition problem, a set of (n-1)-measurements is taken to characterize each pattern to be classified. These measurements are represented as a (n-1)-dimensional vector with binary, integral, or real components depending on the manner in which the measurements are taken. In addition each of the (n-1)-dimensional pattern vectors is augmented by an additional dimension with the corresponding component equal to one. Thus a pattern vector X is written,

$$X = (x_1, x_2, \ldots, x_{n-1}, 1).$$

In this paper we will deal with sets of patterns which can be classified into two distinct categories. It is assumed therefore, that the measurements taken are so discriminatory that two patterns represented by the same vector belong to the same category. Thus let S be the set of patterns to be classified, let A be the set of patterns belonging to category 1, and let B those belonging to category 2. The sets A, B, and S are related as follows,

$$A \cup B = S \quad \text{and} \quad A \cap B = \emptyset.$$

2. A Threshold Locigal Unit.

We define a simple, automatic linear classifier. Let $W = (w_1, w_2, \ldots, w_n)$ be a vector in n-space. Then the set of all vectors $X = (x_1, x_2, \ldots, x_n)$ such that,

$$\sum_{i=1}^{n} x_i w_i = 0 ,$$

defines a hyperplane in n-space through the origin. This hyperplane divides the space into two parts, corresponding to the positive and negative sides of the hyperplane. The vectors belonging to the hyperplane are arbitrarily assigned to the negative side.

Then if for all vectors $X \in A$,

$$X \cdot W = \sum_{i=1}^{n} x_i w_i > 0 ,$$

and if for all vectors $X \in B$,

$$X \cdot W = \sum_{i=1}^{n} x_i w_i \leq 0 ,$$

the classes A and B are linearly separable and in fact are separated by the hyperplane defined by the vector W. We call this vector defining the hyperplane a <u>weight vector</u>.

Consider the schematic diagram of figure 1.1.

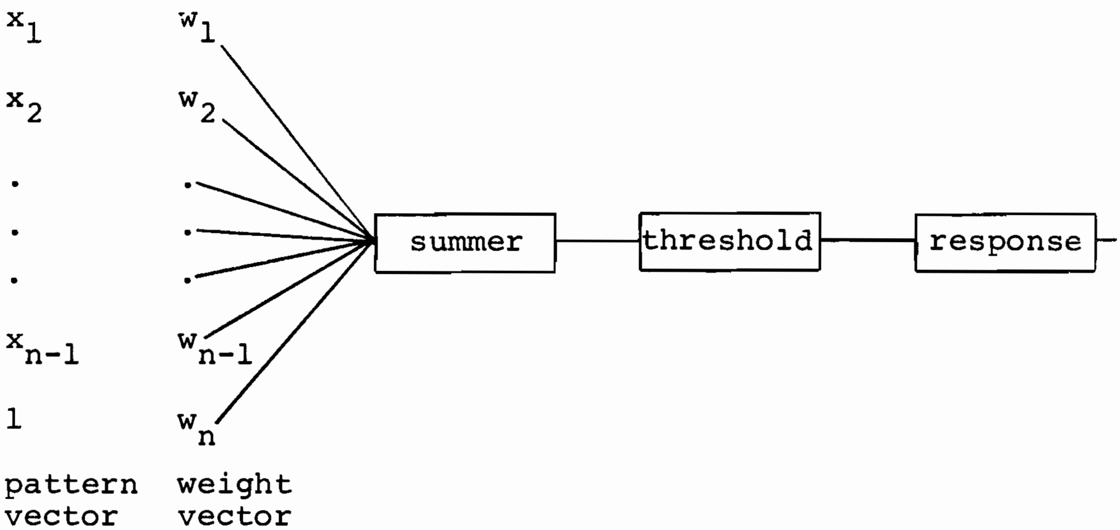


figure 1.1

The summer forms the vector inner product of the vectors X and W. The threshold compares this inner product with the real number called the threshold. The response is a binary output, +1 or -1, corresponding to the case of the inner product being greater than the threshold or less than or equal to the threshold. The representation of the data in n-space has been designed so that we can always use the threshold of zero. Thus we can express the response, r, of figure 1.1 as follows,

$$r = \mathrm{sgn}\left(\sum_{i=1}^{n} x_i w_i\right)$$

where,

$$\text{sgn}(y) = \begin{cases} +1, & \text{for } y > 0 \\ -1, & \text{for } y \leq 0 \end{cases} .$$

This type of linear classifier is commonly called a Threshold Logical Unit, TLU [15].

3. The Committee Machine.

Suppose there are K Threshold Logical Units which have corresponding weight vectors, W_k, and responses, r_k, for $k = 1, 2, \ldots, K$. We can consider the output responses r_k, as components of a $(K+1)$-dimensional vector with the $(K+1)$ component, r_{K+1}, equal 1. Thus we have the vector $R = (r_1, r_2, \ldots, r_K, 1)$. Using another weight vector $U = (u_1, u_2, \ldots, u_{K+1})$, we can build another TLU as shown in figure 1.2.

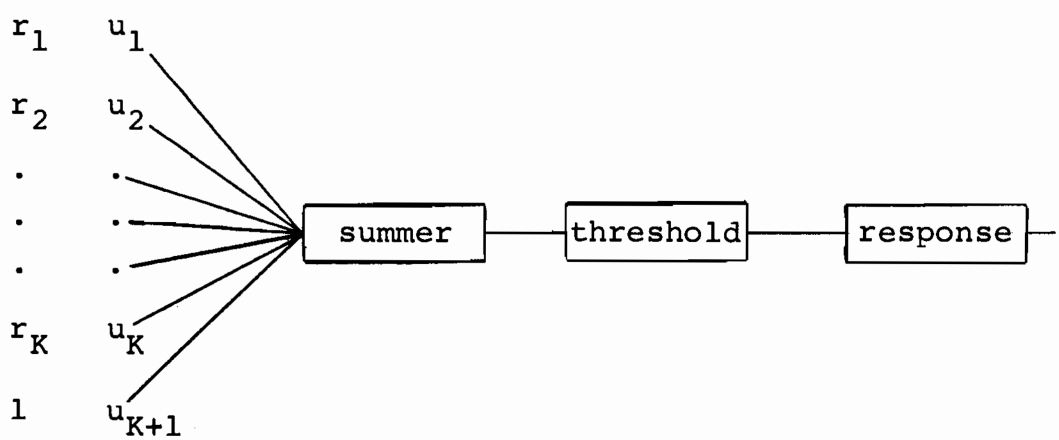


figure 1.2

This second layer of the two layer network again has a response of +1 or -1. A two layer system of this type is commonly called a <u>committee machine</u> [10], [15], [16]. The weight vectors W_k, are called the <u>committee members</u>. Each component, u_k of the weight vector U, represents the importance or influence of each committee member, W_k, in creating the final response of the machine for a pattern vector X.

In a committee machine there are two sets of parameters namely the components of the weight vectors W_k and U. In the machines we shall consider for practical use the weights $w_{i,k}$ of W_k's are allowed to range over the real numbers. The weights of U, u_k, will be restricted as follows,

$$u_k = 1, \quad \text{for} \quad k = 1, 2, \ldots, K \quad \text{and} \quad -K < u_{K+1} < K. \tag{2.1}$$

Thus each committee member in our machines will have equal voting power, with the integer, u_{K+1}, determining the logic of the machine.

In general committee machines of this type are said to use plurality logic, i.e. a committee votes correctly on a pattern if a certain percentage of the members vote correctly. There are two values for u_{K+1} that have received special attention in the past.

<u>Majority Logic</u>. K is odd and $u_{K+1} = 0$. The committee votes correctly on a given pattern if and only if a simple majority of the committee members do.

Veto Logic. $u_{K+1} = -(K-1)$. For $X \varepsilon A$ all the committee members vote correctly. For $X \varepsilon B$ at least one committee member votes correctly, i.e. any committee member has veto power.

III. COMMITTEE MACHINES AND CLASSIFICATION

1. Existence of a Majority Solution.

Besides the adjustable parameters of a committee machine, namely the components of the weight vectors W_k and U, there are two other characteristics of a given machine that greatly effects its capability to solve a two class problem. They are the value of K, the number of committee members in the structure of the machine, and u_{K+1}, the component of U that determines the logic of the machine. We would like to know the relationship between the size of a machine and its logic so that we could choose the most effective combination in solving a given practical problem. First we present an existence theorem.

Theorem 1.1. Given two sets of patterns corresponding to two categories with no pattern belonging to both categories. Then there exists a committee using the majority logic that partitions the space so that the categories are separated without error.

Proof. See [1] and [10].

The proof in [1] is a constructive one which also provides a method of locating the various committee members. But as usual a method resulting from a constructive proof is not a very efficient way of performing the task. In particular the size of the machine resulting from the use of this method may be as large as the number of original patterns to be classified.

2. Conjectures on the Size and Logic of Committee Solutions.

For a given logic obviously there can be many machines each of a different size which solve a given classification problem. Since K is a finite positive integer there is a machine of smallest size. If then we have a machine of minimum size using logic L_1 and a machine of minimum size using logic L_2, will the minimum be the same for both? Some conjectures have been made which provide a partial answer to this question. We present the following made by Kaylor [10] with some discussion.

Let the patterns be represented, as noted above, by n-dimensional vectors with binary components. Let K^* be minimum number of committee members required by any general solution committee, i.e. the components of U may be real. Let K' be the minimum number of committee members required by any general solution with $u_{K+1} = 0$.

<u>Conjecture 1.</u> There is a committee voting by majority logic of size K^*.

Conjecture 2. There is a committee machine voting by plurality logic of size K^*.

Conjecture 3. There is a committee machine voting by majority logic of size K'.

Conjecture 3 has been proven true for $N = 2$ and for $K' = 3$ and N arbitrary. In the proofs given in [4] by Kaylor there is no need for the assumption which requires the components of the pattern vector to be binary. Thus we may be led to assume that the conjectures may be true for real components as well as binary.

Conjecture 1 is false if the pattern vectors are allowed to have real components.

Proof. Consider the following counter example. Let set A be the points in two space just inside the inscribed circle of triangle Δ of figure 3.1. Let set B be the points just outside the circumscribed circle of the same triangle Δ. The lines formed by the three sides of the triangle form a boundary between the sets A and B. If we consider the sides of the lines where the inscribed circle is located as the positive side, then sets A and B are effectively separated by the lines, (hyperplanes in 2-space), using veto logic. Note that three is the minimum number of lines that will separate set A from B.

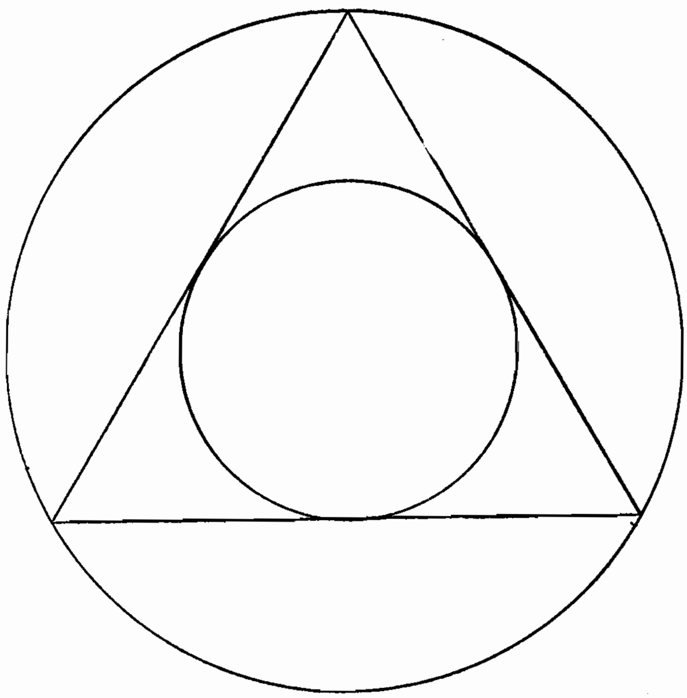


figure 3.1

But no three-line committee will produce a solution using the majority logic. Q.E.D.

In view of the above we highly suspect that Conjecture 1 is not true for binary components either. However, it is our belief that Conjectures 2 and 3 are true for pattern vectors having real or binary components.

In creating a system for finding the individual committee members we would prefer a system which lends itself to application for machines using any of the logic systems, majority, plurality, or veto, since we maintain that these machines perform the job of general machines of the same size.

IV. EXISTING ALGORITHM FOR LOCATING COMMITTEE MACHINES

1. Error Correction Techniques.

Most of the methods used for locating a solution committee use error correction techniques. This is an iterative procedure whereby the pattern vectors are presented to the committee one by one. As each pattern is considered by the committee a vote is taken. If the committee already votes correctly for the pattern under consideration no change in the committee is made. If, however, the committee is in error, i.e. it judges the pattern to belong to the wrong class, some correction (adjustment) is made in one or more of the committee members. The members chosen for this adjustment are usually those whose vote can be most easily changed to a correct vote. Corrections are made by adding a multiple of the pattern vector to the weight vector.

The main advantage of this technique is that the machine looks at one pattern at a time without using any information from the remaining pattern vectors, except as these have already exerted their influence on the machine when they in turn were under consideration by the committee.

2. Algorithms for Committees of Size 1.

The simplest committee machine that we can have is the one consisting of just one committee member. Much investigation and study has already been done concerning this case. We would like to present some of the results here as a goal for development of machines of larger size.

An Algorithm. Suppose we have pattern vectors $X_j \varepsilon A \cup B$, where A and B are linearly separable. Let S_x be a training sequence, i.e. the order in which the patterns are to be presented to the machine, with $S_x = (X_1, X_2, \ldots, \ldots)$ where $X_i \varepsilon S$. This sequence is arbitrary, the only requirement is that each pattern vector $X_j \varepsilon S$ occur infinitely many times in the sequence.

Let S_w be the sequence of weight vectors with $S_w = (W_1, W_2, \ldots, W_i, \ldots)$ where W_1 is arbitrary.

If X_i is the i^{th} vector of the training sequence S_x then,

$$W_{i+1} = W_i \quad \text{if} \quad X_i \cdot W_i > 0 \quad \text{and} \quad X_i \varepsilon A,$$

or

$$\text{if} \quad X_i \cdot W_i \leq 0 \quad \text{and} \quad X_i \varepsilon B,$$

otherwise

$$W_{i+1} = W_i + c_i X_i \quad \text{if} \quad X_i \cdot W_i \leq 0 \quad \text{and} \quad X_i \varepsilon A,$$

or

$$W_{i+1} = W_i - c_i X_i \quad \text{if} \quad X_i \cdot W_i > 0 \quad \text{and} \quad X_i \in B.$$

Theorem 4.1. Let sets A and B be linearly separable categories of pattern vectors in n-space. Let S_w be the weight vector sequence generated by any valid training sequence S_x using the error correction procedure of the algorithm above with $c_i = 1$ for all i and W_1 arbitrary. Then for some finite index i_0,

$$W_{i_0} = W_{i_0+1} = W_{i_0+2} = \cdots$$

is a solution vector.

Proof. This theorem has been proved by various people in sundry ways [2], [3], [13], [14], [15], [17], and [19].

The hypothesis of the theorem stipulating that, $c_i = 1$ for all i, is not a necessary condition for convergence of the algorithm. In practical cases the following values have been used successfully and their convergence proven.

1. $c_i = 1$ for all i (Theorem 4.1)
2. $c_i = 1/\|X_i\|$, i.e. add or subtract a unit vector from W_i
3. $c_i = pW_i \cdot X_i / X_i \cdot X_i$ where $0 < p < 2$.

Theoretically the c_i's may range over a considerably larger set. Block [6] proved that the process will converge if the c_i's satisfy the following,

$$\sum_{i=1}^{\infty} c_i \text{ diverges and the } c_i\text{'s are bounded.}$$

In practical problems encountered in real life the hypothesis of linear separability is often not satisfied. The question is, if this is the case what can be said about the convergence of the algorithm above. Obviously no solution is possible, but a result conjectured by Nilsson [15] and proved by Efron [9] states that the weight vector will not grow indefinitely, more precisely it is bounded. This assurance is a great help in training machines of large size, but it also can be a limiting factor.

3. An Algorithm for Locating a Machine of Size >1.

The following method for training a committee machine of arbitrary size using the majority logic was first proposed by Ridgway [18] and then implemented into the hardware of a learning machine, MINOS II, by Brain et al. [4] and [5], at Stanford Research Institute. According to the authors the machine performed creditably well but suffered from two deficiencies, to wit, (1) for the most part there was a marked difference in the percentage error in the training and the test set, with the training percentage of error always significantly lower than in the test data, and (2) the algorithm had a tendency to get trapped in cyclic or static states. Both of these problems were attributed to the fact that the set was too small.

<u>The Algorithm</u>. Let S_x be a training sequence of the sets A and B, with $S_x = (X_1, X_2, \ldots, X_i, \ldots)$ where $X_i \varepsilon A \cup B$ and each X_i appears infinitely often in the sequence.

Let S_w^k be the weight vector sequences, $S_w^k = (W_1^k, W_2^k, \ldots)$ where W_1^k is arbitrary for $k = 1, 2, \ldots K$ and K is the number of weight vectors in the machine.

Let R_x^i be the response vector,

$$R_x^i = (r_1^i, r_2^i, \ldots, r_K^i, 1) \tag{4.1}$$

where $r_k^i = \begin{cases} 1 & \text{if } X_i \cdot W^k > 0 \\ -1 & \text{if } X_i \cdot W^k \leq 0 \end{cases}$ for $X_i \varepsilon S_x$ and $k = 1, \ldots, K.$

Let U be the logic vector, $U = (u_1, u_2, \ldots, u_K, 0)$, where $u_k = 1$, for $k = 1, 2, \ldots, K$.

Then if X_i is the i^{th} element of S_x and $X_i \varepsilon A$, and if

$$R_x^i \cdot U > 0, \quad \text{let} \quad W_{i+1}^k = W_i^k \quad \text{for} \quad k = 1, 2, \ldots, K,$$

but if

$$R_x^i \cdot U \leq 0,$$

then adjust the W_u^k's according to the following rule.

Let $N_i = |R_x^i \cdot U|$ and arrange the weight vectors such that $W_i^1 \cdot X_i \leq W_i^2 \cdot X_i \leq \ldots \leq W_i^K \cdot X_i$. Let $k_0 = (K - 1)/2$.

Now let

$$W_{i+1}^k = W_i^k + c_i X_i \quad \text{for} \quad k = k_0, k_0+1, \ldots k_0+(N_i+1)/2$$

and

$$W_{i+1}^k = W_i^k \quad \text{otherwise.}$$

Suppose X_i is the i^{th} element of S_x and $X_i \varepsilon B$.
If

$$R_x^i \cdot U \leq 0, \quad \text{let} \quad W_{i+1}^k = W_i^k \quad \text{for} \quad k = 1, 2, \ldots, K,$$

but if

$$R_x^i \cdot U > 0,$$

adjust the W_i^k's according to the following rule.

Let $N_i = R_x^i \cdot U$ and rearrange the weight vectors such that $W_i^1 \cdot X_i \leq W_i^2 \cdot X_i \leq \ldots \leq W_i^K \cdot X_i$ and let $k_0 = (K-1)/2$.

Now let,

$$W_{i+1}^k = W_i^k - c_i X_i \quad \text{for} \quad k = k_0, k_0-1, \ldots, k_0 - (N_i+1)/2$$

and

$$W_{i+1}^k = W_i^k \quad \text{otherwise.}$$

The adjustment criterion of the above algorithm can be stated very briefly. If the machine already votes correctly by the majority logic nothing is done. If, however, the machine votes incorrectly then a number of hyperplanes are chosen to be adjusted. The hyperplanes closest to the pattern vector under consideration are chosen such that the number of original correct votes plus the number of hyperplanes just adjusted would be just sufficient to achieve a simple majority.

Another algorithm for training committee machines having veto logic, which is very similar to the preceding algorithm and also due to Ridgway [18], adjusts hyperplanes according to the following rule. If $X_i \in A$ and the machine votes correctly nothing is done. If the machine votes incorrectly then all the hyperplanes which are wrong are adjusted. If $X_i \in B$ and the machine votes correctly nothing is done. If the machine votes incorrectly then the one hyperplane which is closest to the pattern vector under consideration is adjusted.

As indicated earlier both of these algorithms have been used with some success in training committee machines. The major difficulties they encounter will be lessened by our new process of training which we now describe.

V. TEEMAL - A NEW METHOD FOR LOCATING A COMMITTEE

1. Preliminary Remarks.

This method will consist of three main parts.

i. A process for replacing one of the hyperplanes.

ii. An algorithm for adjusting an individual hyperplane.

iii. An algorithm for adjusting all the hyperplanes.

In the design of our method all three processes are used successively with critera of time and success determining the transfer from one to the other.

As it has been pointed out the difficulties in training committee machines have been, (i) the tendency of the machines to get trapped in static or cyclic states, and (ii) the disparity in the percentage of errors in the training set versus that in the test set. These phenomena have been attributed to the use of too small a number of training samples. We believe, however, that the problems are not that simple, but that the nasty, complex, problems of prejudice and misplaced samples in training must be faced.

A limited, for linear machines only, attempt has been made in allowing misplaced samples in training by Duda and Singleton [8]. The adjustment criterion used never allowed errors without making some sort of correction, but rather attempted to minimize the effect of misplaced samples by an averaging technique of intermediate stages of learning.

Thus a major attempt has been made in the present method to allow for errors, i.e. misplaced or excessively noisy samples which may exert undue influence in the development of the machine by providing misleading and prejudicial information to the training algorithms. In addition, the error correction critera were designed in such a way that the machine could bail itself out of static or cyclic states, or better, never get trapped at all.

2. CREATE - A Method of Creating a New Hyperplane.

Suppose we have a committee machine, $C_{K,L}$, under training which consists of K-hyperplanes and these vote according to the logic, L. The process, CREATE, replaces one of the weight vectors defining a hyperplane, say W^k.

Let I_a be the number of elements in set A and I_b be the number of elements in set B. Let R_x^i be the response vector in 4.1 with $r_k^i = 0$. Let $U = (1, 1, \ldots, 1, u_{K+1})$ be the logic vector and without loss of generality we assume $(K + u_{K+1})$ is an odd integer. Let $S^a = S^b = 0$ be vectors in n-space.

Consider $X_i \in A$, for $i = 1, 2, \ldots, I_a$.

If $R_x^i \cdot U > 0$,

let $S^a = S^a$ and $n_i^a = 0$.

If $R_x^i \cdot U < 0$,

let $n_i^a = |R_x^i \cdot U| + 1$ and $S^a = S^a + n_i^a X_i$.

Consider $X_i \in B$, for $i = 1, 2, \ldots, I_b$.

If $R_x^i \cdot U < 0$,

let $S^b = S^b$ and $n_i^b = 0$.

If $R_x^i \cdot U > 0$,

let $n_i^b = R_x^i \cdot U + 1$ and $S^b = S^b + n_i^b X_i$.

Now let

$$N^a = \sum_{i=1}^{I_a} n_i^a \quad \text{and} \quad S^a = S^a/N^a$$

and

$$N^b = \sum_{i=1}^{I_b} n_i^b \quad \text{and} \quad S^b = S^b/N^b .$$

Thus we have formed two vectors which represent weighted centers of mass for some or all the elements of

A and B respectively. The weight factors, n_i^a and n_i^b, attribute to each vector in $A \cup B$ some measure of importance, the importance being the need for the new committee member to vote correctly on the particular pattern vector.

Given the vectors thus formed with,

$$S^a = (s_1^a, s_2^a, \ldots, s_n^a) \quad \text{and} \quad S^b = (s_1^b, s_2^b, \ldots, s_n^b)$$

we define the vector, $W^k = (w_1^k, w_2^k, \ldots, w_n^k)$, as follows.

Let $w_i^k = s_i^a - s_i^b$ for $i = 1, 2, \ldots, n-1,$

and

$$w_n^k = \frac{1}{2}\sum_{i=1}^{n-1} (s_i^b)^2 - \frac{1}{2}\sum_{i=1}^{n-1} (s_i^a)^2 .$$

Thus if we denote by X^-, W^{k-}, S^{a-}, and S^{b-} as vectors in the original non-augmented (n-1)-space, then the hyperplane defined by the equation $W^{k-} \cdot X^- = w_n^k$ is the perpendicular bisector of the line connecting the vectors S^{a-} and S^{b-}.

3. ALCONONE - An Algorithm for Adjusting One Hyperplane.

The hyperplane generated by CREATE is usually far from being the best hyperplane possible to cure the errors of the machine at time it is generated. It is, however, a good beginning, as an initial vector for our error-correction algorithm, in particular it is far superior to the random selection of an arbitrary vector.

In order to make maximum use of the discriminating power of the newly created hyperplane, then, it is useful to adjust it slightly so that pattern vectors just barely on the wrong side have a change of being accepted on the correct side. Yet, we do not want to adjust the hyper-plane excessively in an attempt to have this one hyper-plane assume too much responsibility in separating the two categories. If we consider a new hyperplane as a line in figure 5.1, a 'good' position this hyperplane to move, as a committee member, would be between clusters A and B_1 as line L_1. We call such areas, 'areas of local separability.' The adjusting algorithm should not concern itself too much with the patterns in the cluster B_2, which are at some distance from the hyperplane. Straight forward use of existing algorithms, whether fixed or variable increment, would result in locating the hyperplane in the position of line L_2 in figure 5.1, cutting through all the cluster areas. Mengert has suggested a way of giving weights to the pattern vectors to achieve a similar result [11].

The algorithm, ALCONONE, by its continuous function increment adjustment seeks out areas of local separability and is tolerant of errors, even a significant number.

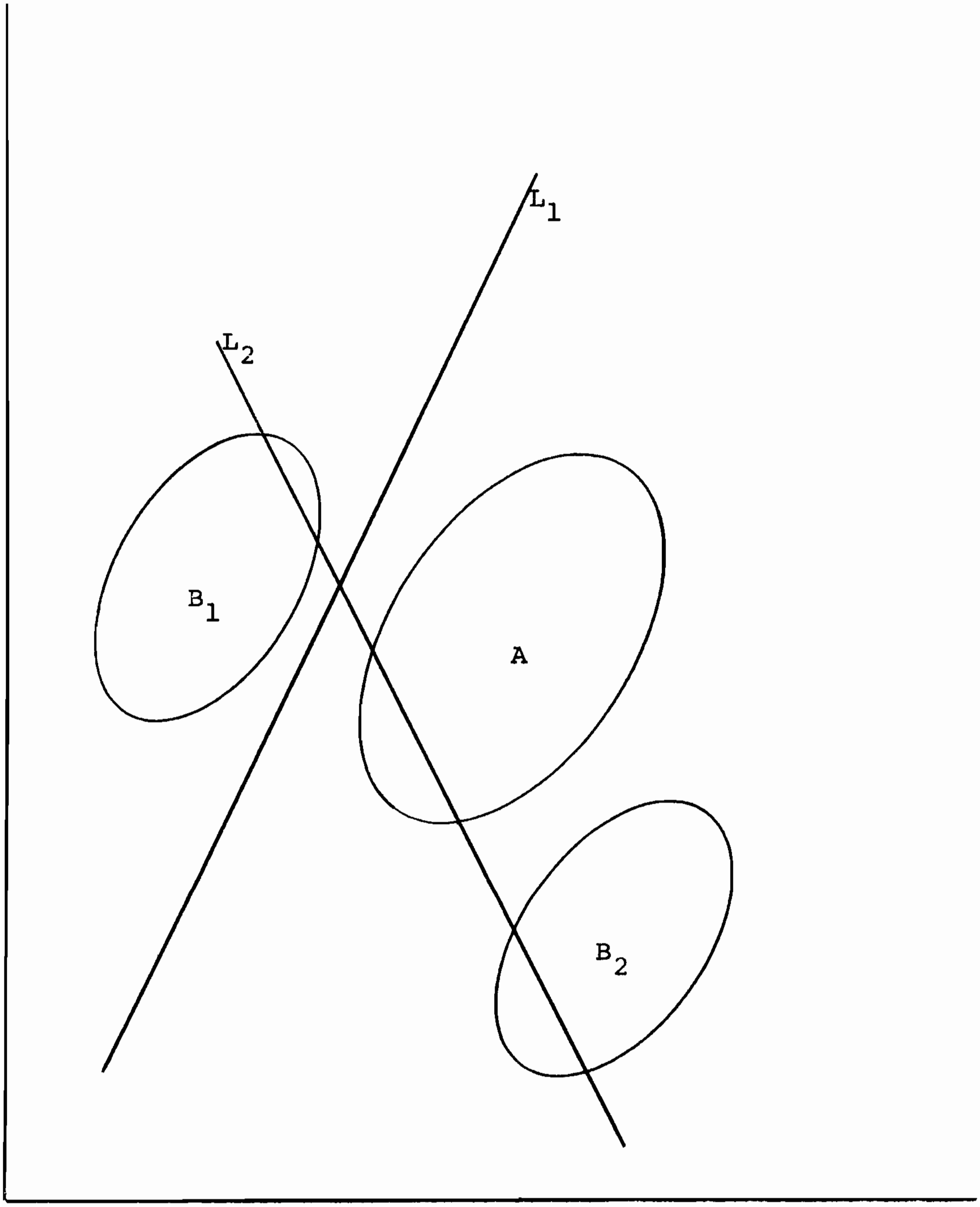


figure 5.1

<u>The Algorithm</u>. Let S_x be a training sequence for the sets A and B, with $S_x = (X_1, X_2, \ldots, X_i, \ldots)$ where $X_i \; \varepsilon \; A \quad B$ and each X_i appears infinitely often in the sequence.

Let S_w be a weight vector sequence, with $S_w = (W_1, W_2, \ldots)$ and $\|W_1\|^* = 1$, where $\|\cdot\|^*$ is the Euclidean norm of the first (n-1) elements of a vector in n-space.

Let R_x^i be the response vector in 4.1 and U be the logic vector described in 2.1.

Let X_i be the i^{th} element in S_x.

If $\quad X_i \; \varepsilon \; A \cup B$

and $\quad R_x^i \cdot U > 0$ or $W_i \cdot X_i > 0$,

let $\quad W_{i+1} = W_i$;

if $\quad X_i \; \varepsilon \; A$

and $\quad R_x^i \cdot U < 0$ and $W_i \cdot X_i \leq 0$,

let $\quad W_{i+1} = W_i + f(d_i) X_i / \| X_i \|$;

if $\quad X_i \; \varepsilon \; B$,

and $\quad R_x^i \cdot U < 0$ or $W_i \cdot X_i \leq 0$,

let $\quad W_{i+1} = W_i$;

if $\quad X_i \; \varepsilon \; B$,

and $\quad R_x^i \cdot U > 0$ and $W_i X_i > 0$,

let $\quad W_{i+1} = W_i \quad f(d_i) X_i / \| X_i \|$,

where $\quad d_i = W_i \cdot X_i$ and $f(d) = \beta + \dfrac{\alpha}{\alpha p + d^2}$.

The parameters α, β, and p of the function f play the following roles:

i. 'β' determines the infimum of the function.

$$\lim_{d\to\infty} f(d) = \beta .$$

ii. 'p' determines the maximum of the function.

$$f(0) = \beta + \frac{1}{p} .$$

iii. 'α' plays a very important and delicate role in determining the precise amount of adjustment to be implemented by the algorithm. If regulates the rate of decrease of the monotone, decreasing function f.

In practice since we are always working with finite sets, $\beta = 0$. It has been made part of the function for the theoretical considerations. 'p' on the otherhand had a value of around 10.

From experiments we found a good value for α should satisfy the following. Let $\bar{W}$ be the weight vector corresponding to the hyperplane which is the perpendicular bisector of the line joining the centers of mass of A and B, with $\|\bar{W}\| = 1$. Let I_a be the number of elements in A and I_b the number of elements in B. Let $d_0 = 5\sqrt{\alpha p}$. If $X \varepsilon A \cup B$, let N_{d_0} be the number of pattern vectors X such that,

$$d_0 < |X \cdot W| .$$

Then pick α such that

$$.02(I_a + I_b) < N_{d_0} < .05(I_a + I_b).$$

In other words N_{d_0} should be more than 2 % of the total number of pattern vectors but less than 5 %.

It should be noted that this algorithm satisfies the generalization of Theorem 4.1 [3]. Thus if the two classes are linearly separable this algorithm will converge to a solution. We feel this is an important additional feature since we may be concerned with subsets of A and B which may well be linearly separable. A graph of the function is given in figure 5.2.

Methods for Adjusting All Committee Members Simultaneously

We present two algorithms here for adjusting all the committee members simultaneously. The first uses the continuous function technique just described for adjusting a single committee member.

4. Algorithm I, ALCONTY.

Let S_x be a training sequence $S_x = (X_1, X_2, \ldots)$ and S_w^k be the weight vector sequences,
$S_w^k = (W_1^k, W_2^k, \ldots)$ where $k = 1, 2, 3, \ldots, K$ and $\|W_1^k\|^* = 1$.

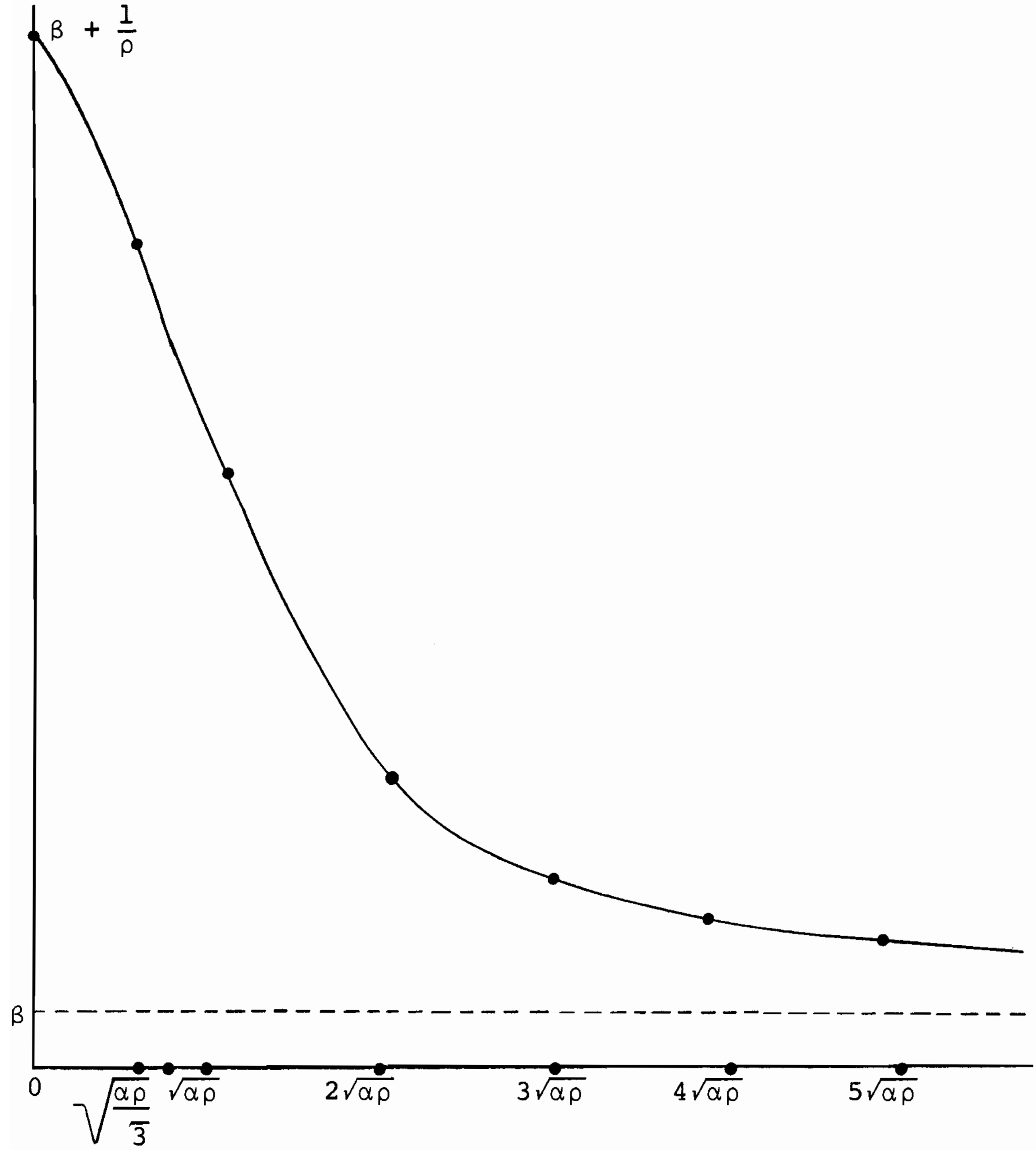


figure 5.2 $\quad f(d) = \beta + \frac{\alpha}{\alpha\rho + d}$

Let R_x^i be the response vector

$$R_x^i = (r_1^i, r_2^i, \ldots, r_K^i, 1)$$

where $r_k^i = \begin{cases} 1 & \text{if } X_i \cdot W^k > 0 \\ -1 & \text{if } X_i \cdot W^k \leq 0 \end{cases}$ for $X_i \in S_x$.

Let U be the logic vector

$$U = (u_1, u_2, \ldots, u_K, u_{K+1})$$

where $u_k = 1$, for $k = 1, 2, 3, \ldots, K$ and $-K < u_{K+1} < K$.

Let X_i be the i^{th} element in the training sequence S_x, and $X_i \in A$.

Then if:

$$R_x^i \cdot U > 0$$

let $W_{i+1}^k = W_i^k$, $k = 1, 2, \ldots, K$.

If:

$$R_x^i \cdot U \leq 0$$

let $W_{i+1}^k = W_i^k + f(d_i^k) X_i$ for all $k = 1, 2, \ldots, K$ and $d_i^k < 0$,

where,

$$d_i^k = W_i^k \cdot X_i \quad \text{and} \quad f(d) = \beta + \frac{\alpha}{\alpha p + d^2} \quad .$$

If X_i is the i^{th} element of S_x and $X_i \varepsilon B$, then:

if $R_x \cdot U \leq 0$

let $W^k_{i+1} = W^k_i$, $k = 1, 2, \ldots K$.

But if $R_x \cdot U > 0$

let $W^k_{i+1} = W^k_i - f(d^k_i)X_i$ for $k = 1, 2, \ldots, K$

and $d^k_i > 0$,

where the function f and d^k_i are as above.

Evaluation of Algorithm I

Algorithm I, ALCONTY, has all the desired characteristics described for the continuous function adjusting algorithm, ALCONONE. This algorithm has proved effective in locating a committee and performs as well as the algorithm due to Ridgway described in Chapter IV. But to achieve these results some delicate variance was needed on the parameters α, β, and P of the function f. Also it was necessary to maintain the norm of the weight vectors W^k near equality, in our case $\|W^k\|* = 1$. To overcome these handicaps we have devised Algorithm II, ALVARY, which has all the good characteristics of ALCONTY without the described sensitivity.

5. ALVARY - A Second Algorithm for Adjusting All Hyperplanes.

Our second algorithm for adjusting all the committee members simultaneously uses a 'closeness' criterion, but in a very different fashion.

Let S_x and S_w^k be the pattern sequence and the weight vector sequence respectively. Without loss of generality we restrict the component, U_{K+1}, of the vector U such that $(K + u_{K+1})$ is an odd integer. Let P_a and P_b be rational numbers of the form $1/N$.

Suppose X_i is the i^{th} element of S_x and $X_i \varepsilon A$.

If $R_x^i \cdot U > 0$

let $W_{i+1}^k = W_i^k$ for $k = 1, 2, \ldots, K$,

if $R_x^i \cdot U \leq 0$ and $0 < d_i^k/d < P_a$

let $W_{i+1}^k = W_i^k + cX_i/\|X_i\|$

and $W_{i+1}^k = W_i^k$ otherwise

for $k = 1, 2, \ldots, K$,

where $0 < c < 2$, $d_i^k = W_i^k \cdot X_i$,

and $d = \sum_{d_i^k \leq 0} d_i^k$, $k = 1, 2, \ldots, K$.

Suppose on the other hand X_i is the i^{th} element of S_x and $X_i \varepsilon B$.

If $R_x^i \cdot U \leq 0$

let $W_{i+1}^k = W_i^k$ for $k = 1, 2, \ldots, K$,

if $R_x^i \cdot U > 0$ and $0 < d_i^k/d < P_b$

let $W_{i+1}^k = W_i^k - cX_i/\|X_i\|$

and $W_{i+1}^k = W_i^k$ otherwise for $k = 1, 2, \ldots, K$,

where $0 < c < 2$, $d_i^k = W_i^k \cdot X_i$,

and $d = \sum_{d_i^k > 0} d_i^k$, $k = 1, 2, \ldots, K$.

Qualitative Analysis of ALVARY

The criterion for adjusting used in the algorithm ALVARY depends on the closeness of the pattern vectors to hyperplanes. The dependence, however, is relative to the distances of the pattern vectors from the hyperplanes. This allows for great flexibility in the number of committee members to be adjusted at a particular step in an iteration, as well as in the choice of the particular members to be adjusted. This is in contrast to previous rules which possess fixed and rigid critera for adjusting.

We will summarize here what we feel are effective and original properties for an error-correction technique.

Property 1. There may be no adjustment of any hyperplane even though the machine votes incorrectly on a given pattern vector.

Example. Let $C_{K,L}$ be a committee machine with $K = 5$ and L is the majority logic, i.e. the component u_{K+1} of the vector U is 0. Let $P_a = P_b = \frac{1}{4}$. Suppose $C_{K,L}$ votes incorrectly on the pattern X and d_1, d_2, and d_3 are the distances of X from the exactly three incorrect voting hyperplanes. For simplicity we assume that $d_1 < d_2 < d_3 < 0$. Let $d = d_1 + d_2 + d_3$.

Then $d_1/d < \frac{1}{4}$

implies $4d_1 < d_1 + d_2 + d_3$

implies $d_1 < (d_2 - d_1) + (d_3 - d_1)$. (5.1)

Thus there will be an adjustment only if d_1 is less than the sum of the differences of (d_1, d_2) and (d_1, d_3). If the d_i's are nearly equal there will be no adjustment.

Property 2. If all the hyperplanes are close to the pattern vector, X, there is greater probability that there will be some adjustment than if they were all far away.

We note equation (5.1) above. If d_1 is small it is more likely that the right hand sum will be greater than d_1. The degree of variance possible in the d_i's without adjustment occurring decreases as the d_i's decrease.

Property 3. If there is any adjustment, one or more of the hyperplanes closest to the pattern vector will be adjusted.

Property 4. If more than one hyperplane needs to have its vote changed for the machine to vote correctly, then usually more than one hyperplane will be adjusted.

Property 5. If one of the hyperplanes is at a great distance from the pattern vector in relation to the others then all the closer hyperplanes will be adjusted.

We note again (5.1). If $d_3 > 3d_2 > 3d_1$ then both hyperplanes corresponding to d_1 and d_2 will be adjusted.

In evaluating the properties just described, Property 1 proves to be very advantageous and is unique for error correcting techniques. Until now no error correcting method allowed for errors in the machine without making some adjustment of the hyperplanes [15]. As noted in Property 2 the probability of no adjustment occurs when the distances from the pattern vector to the hyperplanes which vote incorrectly are large. These large distances would indicate, especially in the more advanced stages of learning, that the pattern vector in question is perhaps isolated or a rather degenerate and noisy case of an element in the given class. In either case, we do not wish for such a pattern vector to have any influence in locating the committee machine.

Property 3 assures us that the algorithm still corrects hyperplanes which are closest to being correct and easiest to change to a proper voting posture.

Property 4 follows the accepted practice that the more hyperplanes there are that vote wrong the greater the need for adjusting more hyperplanes.

In Property 5 we have a very interesting and profitable characteristic. For example if we have a pattern vector close to two hyperplanes and relatively far away from the other wrong hyperplanes, and we are to adjust a single committee member, to choose the closest one is a quite arbitrary discision since there is a very small difference between the two close hyperplanes. ALVARY will adjust both of them in this case. Precisely which one would have been better to adjust will be determined by future adjustments needed for other pattern vectors. The algorithm by its action implicitly chooses which hyperplane should have been adjusted. This added degree of freedom allows the algorithm to adapt itself to various classification problems.

6. TEEMAL, The Method of Locating the Machine.

Using the algorithms above we will now put them together in a process for finding a committee machine to solve a general two class problem.

INITIALIZE

Create k^{th} weight vector,

Use ALCONONE on k^{th} weight vector for I_s iterations,

for $k = 1, 2, \ldots K$.

$k = 0$.

Step I

Call ALVARY.

If no adjusting on iteration go to Step II.

If no errors Stop.

After I_g iterations go to Step II.

Step II

$k = k + 1$.

If $k > K$, $k = 1$.

If $k < K$, $k = k$.

Create k^{th} weight vector.

Use ALCONONE on k^{th} weight vector for I_s iterations.

Go to Step I.

Further Elaboration.

Initialization. As stated each weight vector is created and then adjusted successively until the proper number is obtained. As each weight vector is created and adjusted a record is kept of how each one of them votes on each particular pattern vector.

As the algorithm proceeds in making the decisions whether to adjust or not, a check is made of the voting record of the existing weight vectors. If this record for a particular pattern is already sufficient to achieve a correct vote according to the logic, L, of the machine $C_{K,L}$, then no adjustment is performed, even though the weight vector under consideration may be voting incorrectly on the pattern vector. Thus only the pattern vectors needing a correct vote from this weight vector will be considered for possible adjusting.

The single adjusting algorithm is controlled by an integer parameter, I_s, which determines the number of iterations the algorithm shall perform each time it is used.

Step I. In general the adjusting algorithm runs until there is no longer any improvement in the machine. To implement this criterion an integer parameter, I_g, is given the machine. It is at the end of a cycle of

I_g-iterations that a decision is made whether there is any improvement decrease in errors. If there is a decrease in the errors made by the machine another cycle of I_g-iterations is performed.

If, however, there has been no decrease in the number of errors after a particular cycle of I_g-iterations then the algorithm stops and returns to Step II in the state that has been best so far. This state is either the condition of the weight vectors before starting the latest cycle of iterations, or the state before iterations of this cycle that has had the fewest adjustments. The state chosen is the one that makes the fewer errors.

The algorithm also stops if there is no adjustment during some particular iteration.

Step II. In Step II each weight vector is replaced successively from 1 to K. A number of critera have been tested to determine which of the weight vectors to replace. These were such as, the weight vector adjusted most during the previous cycle of Step I, the vector adjusted the least, or the vector making the most errors as a linear machine, committee of one. For the most part these critera led to cyclic or static states in which the same individual vectors were being replaced. The parameters and critera used for creating and adjusting a weight vector are the same as those used in the initialization stage.

A number of check points exists in the process which terminate the algorithm as soon as no errors are made by the machine $C_{K,L}$ being located.

VI. EXPERIMENTAL RESULTS

The above method, TEEMAL, of alternating algorithms and procedures has been applied to various kinds of two-class problems. We present a number here which involve different facets and difficulties in classification.

1. Artificially Generated Data in 2-Space.

First we present an example of the performance of the algorithm, ALCONONE, for adjusting a single hyper-plane.

Suppose we have the set of letters "a" and "b" located in 2-space according to figure 6.1, with the a's belonging to class A and the b's belonging to class B. The line, hyperplane H_p, in the figure 6.1 is the perpendicular bisector of the line connecting the centers of mass of the classes A and B.

The parameters of the function f of ALCONONE are set as follows, $\beta = 0$, $\alpha = 0.01$, and $p = 8$. The position of the hyperplane, H_p, after each iteration (one iteration equals one pass through the data) is indicated by the lines $H_1, H_2, \ldots, H_9$ in figure 6.1. ALCONONE was allowed to run for a total of fifty iterations, but there was no significant change in the position of the lines after the second iteration. This position is considered to be a very effective one for a member of some committee machine.

As a means of comparison the fixed increment error-correction algorithm of page (16) was also run with $c = 0.1$. The hyperplane wandered from the initial position of H_p figure 6.2, through the positions H_1, H_2, ... H_9. These are the positions after each iteration as in ALCONONE. The algorithm ran through another 50 iterations with the same scattered positioning of the hyperplane, never reaching anything resembling a stable or 'good' position.

A Committee Solution

Consider now the set of a's in set A and the set of b's in set B in figure 6.3. Obviously there is no linear separation possible, and in addition, there is no committee solution for a committee machine of size less than five. Our experiment attempts to separate these two sets, A and B, with a committee machine $C_{K,L}$, where $K = 5$ and L = the majority logic.

We will compare the performance of Stage I, the algorithm ALVARY with the algorithm due to Ridgway described in Chapter IV using the majority logic. In both algorithms the coefficient of adjustment, c, was equal to $.1/\|X\|$. For ALVARY, $P_a = P_b = 1/5$.

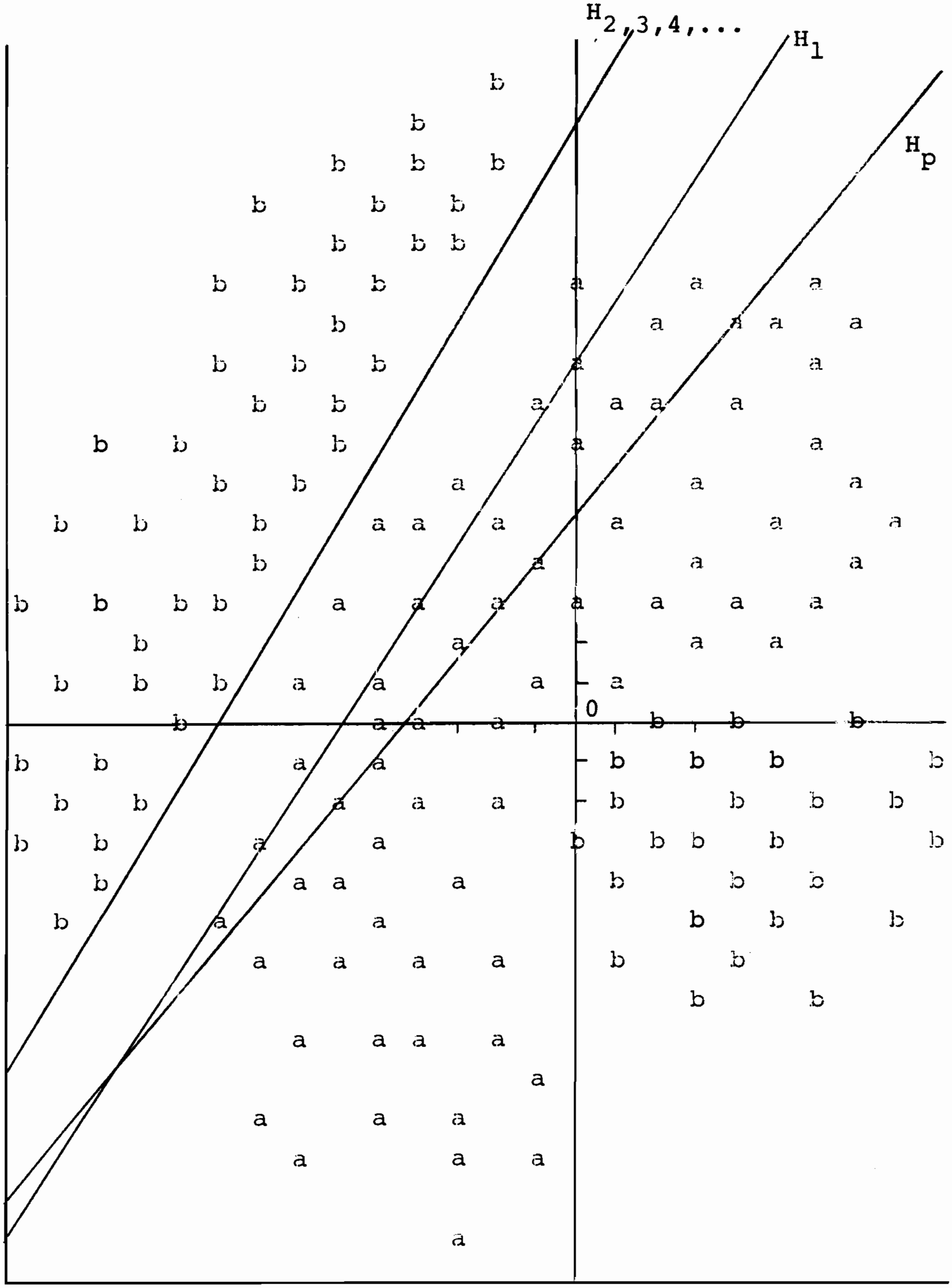


figure 6.1   Performance of ALONONE

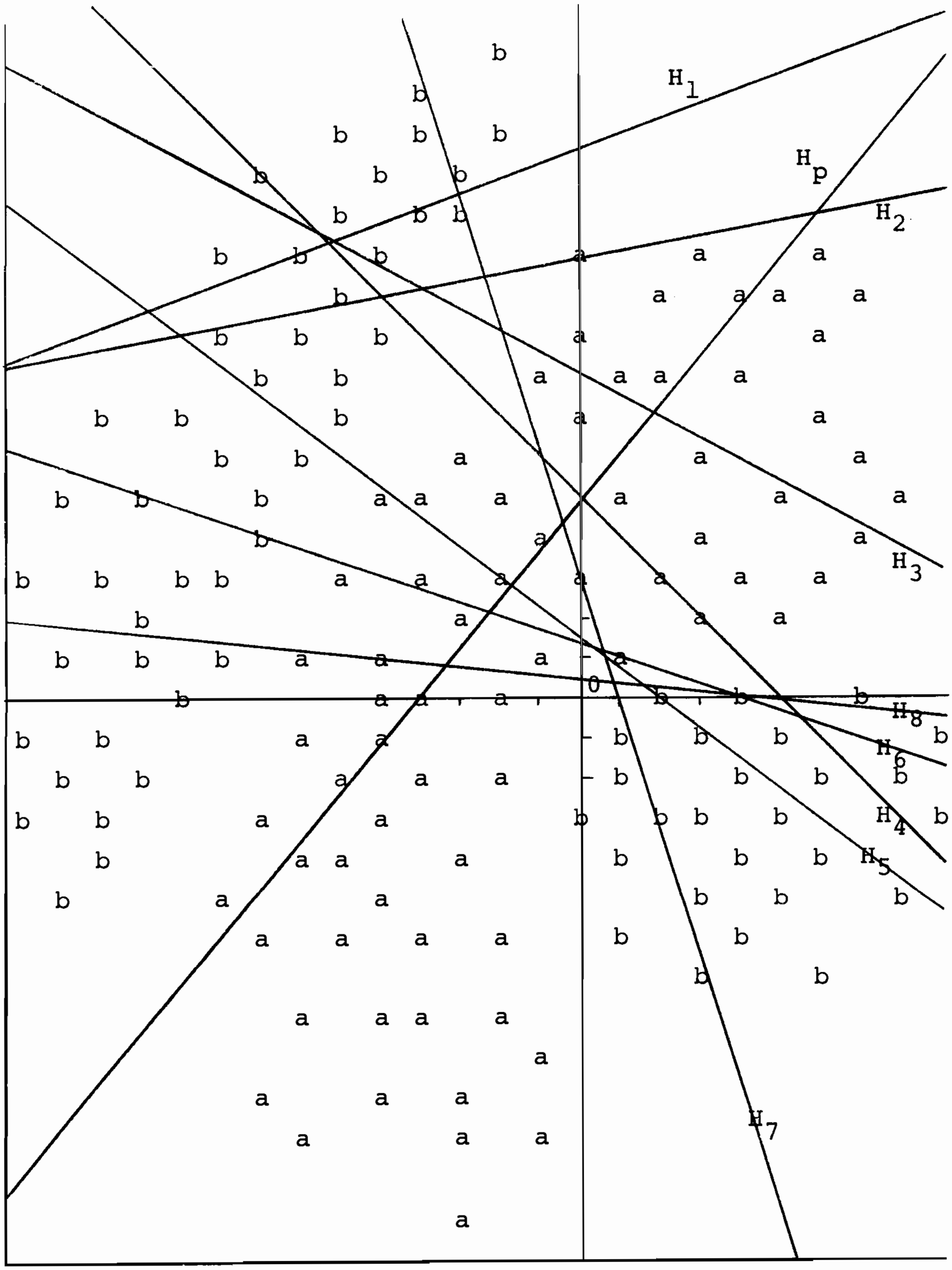


figure 6.2 Fix Increment Algorithm

CASE I:

Both algorithms were given the following vectors generated by the initializing stage.

$W_1 = (-0.8250, -0.5650, .3005)$

$W_2 = (.7861, .6181, 1.1804)$

$W_3 = (-0.7597, 0.6503, -1.8173)$

$W_4 = (.4602, -0.8878, 4.0328)$

$W_5 = (-0.0411, .9992, -0.0646)$

The intersection of the hyperplanes, determined by these weight vectors, with the plane, $z = 1$, is pictured in figure 6.3. These hyperplanes form a partial solution making 37 errors.

The algorithm ALVARY after 13 seconds, 22 iterations, and 471 adjustments located a set of hyperplanes making no errors defined by the following weight vectors.

$W_1 = (-0.9951, -0.0989, -0.3379)$

$W_2 = (.8693, -0.4943, 7.4013)$

$W_3 = (-0.1402, -0.9901, -7.3710)$

$W_4 = (.5721, -0.8202, 3.7841)$

$W_5 = (.0249, .9997, -0.7574)$

Ridgway's algorithm given the same starting position for the weight vectors after 24 seconds, 32 iterations,

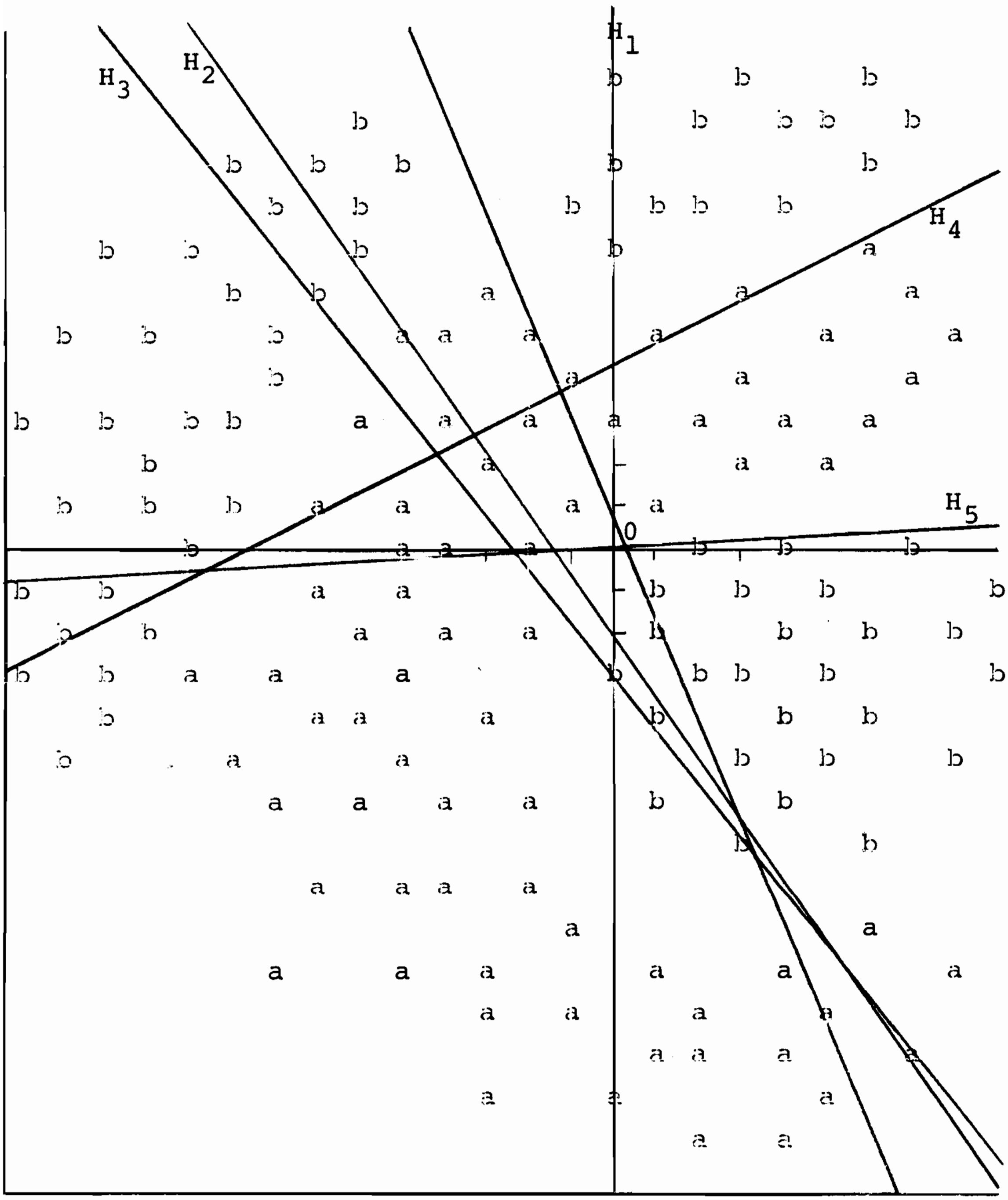


figure 6.3 Initial Stage

and 379 adjustments located the following weight vectors which vote correctly on all the elements of the two sets.

$W_1 = (-0.9176, -0.1133, -0.3488)$

$W_2 = (.2011, -0.0625, 1,5045)$

$W_3 = (-0.1111, -0.3216, -1.9526)$

$W_4 = (.6477, -0.8826, 4.0033)$

$W_5 = (.0114, .6163, -0.5318).$

Both of these solutions have the hyperplanes located in relatively the same positions as seen from figures 6.4 and 6.5.

CASE II:

The two algorithms were given another set of weight vectors generated by the initializing stage as follows:

$W_1 = (-0.9994, -0.0344, -0.6268)$

$W_2 = (.8941, -0.4478, 2.5590)$

$W_3 = (-0.8319, .5549, 2.3090)$

$W_4 = (.8632, -0.5049, 3.5920)$

$W_5 = (-0.9687, -0.2481, -1.4796)$

The hyperplanes determined by these weight vectors form a partial solution making 61 errors.

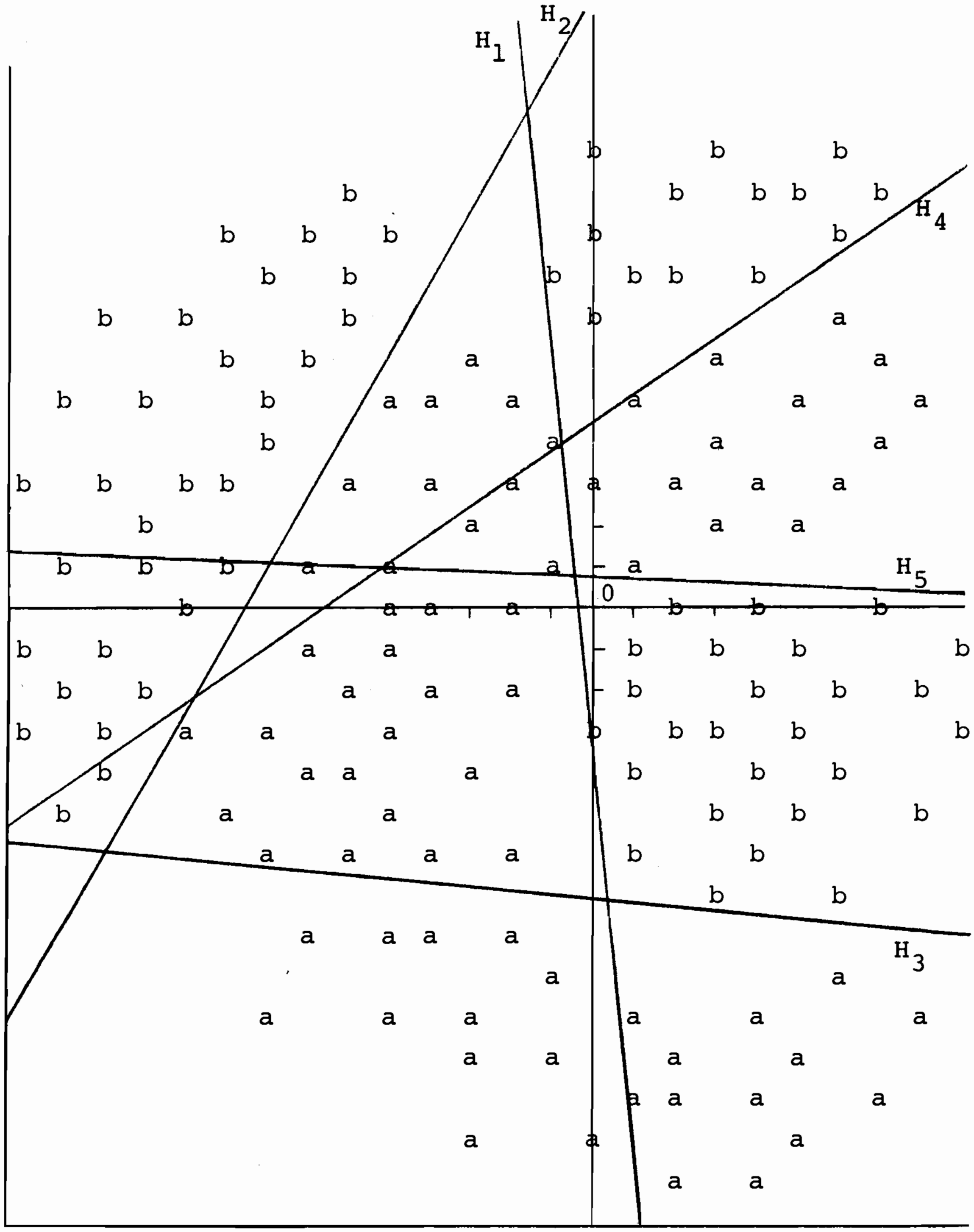


figure 6.4 ALVARY Solution

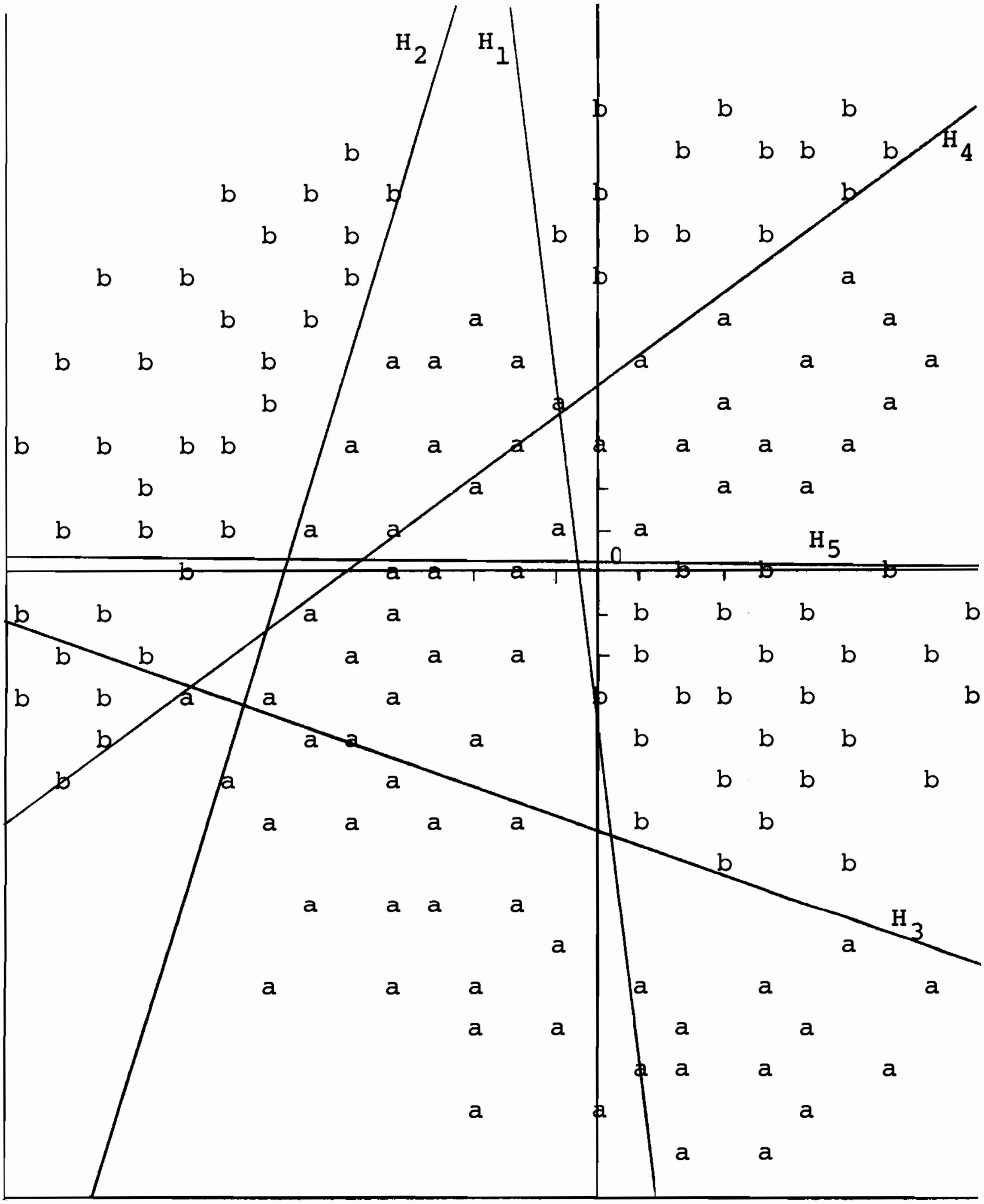


figure 6.5   Ridgway Solution

The algorithm ALVARY after 14 seconds, 31 iterations, and 364 adjustments located a complete solution as follows:

$$W_1 = (-0.9899, -0.1418, -0.4509)$$

$$W_2 = (\ .5633, -0.8262, \ 4.0524)$$

$$W_3 = (-0.1214, \ .9926, \ .1975)$$

$$W_4 = (\ .8503, -0.5263, \ 7.0637)$$

$$W_5 = (-0.0927, -0.9957, -7.5338).$$

Ridgway's algorithm found a complete solution after 23 seconds, 30 iterations, and 282 adjustments. The weight vectors are as follows:

$$W_1 = (-1.2218, -0.0836, -0.5711)$$

$$W_2 = (\ .4598, -0.6101, \ 2.6213)$$

$$W_3 = (-0.8731, \ 1.4919, \ 1.7126)$$

$$W_4 = (\ .3918, -0.2472, \ 3.6811)$$

$$W_5 = (\ .0417, -0.1619, -1.6506).$$

As noted by these results ALVARY performs as well as Ridgway's algorithm with the advantage of being faster since the decision making process of ALVARY in determining precisely which weight vectors to adjust is simpler. In addition, the fact that ALVARY makes more adjustments in locating the complete solution is an advantage since this is part of the decision making process, as noted in property 4 of ALVARY.

A Partial Solution

Given the sets A and B in figure 6.6, which are the same as in figure 6.3 except that one of the elements of set A has been removed, indicated by @ , and has been replaced by an element of set B, indicated by b'. Thus there is no committee machine possible which will separate the two classes. There is, however, the partial solution which makes just the one error on the element belonging to both classes. We now investigate the behavior of TEEMAL and Ridgway's algorithm in their attempts to locate a solution.

The initial stage located the following weight vectors.

$$W_1 = (-0.8059,\ -0.5921,\ -0.5213)$$

$$W_2 = (\ .9670,\ \ .2548,\ \ 2.9075)$$

$$W_3 = (-0.6464,\ -0.7630,\ -1.0230)$$

$$W_4 = (\ .8732,\ -0.4873,\ \ 5.8161)$$

$$W_5 = (\ .4388,\ \ .8986,\ -1.5830)$$

The committee defined by these weight vectors makes 34 errors as indicated in figure 6.6.

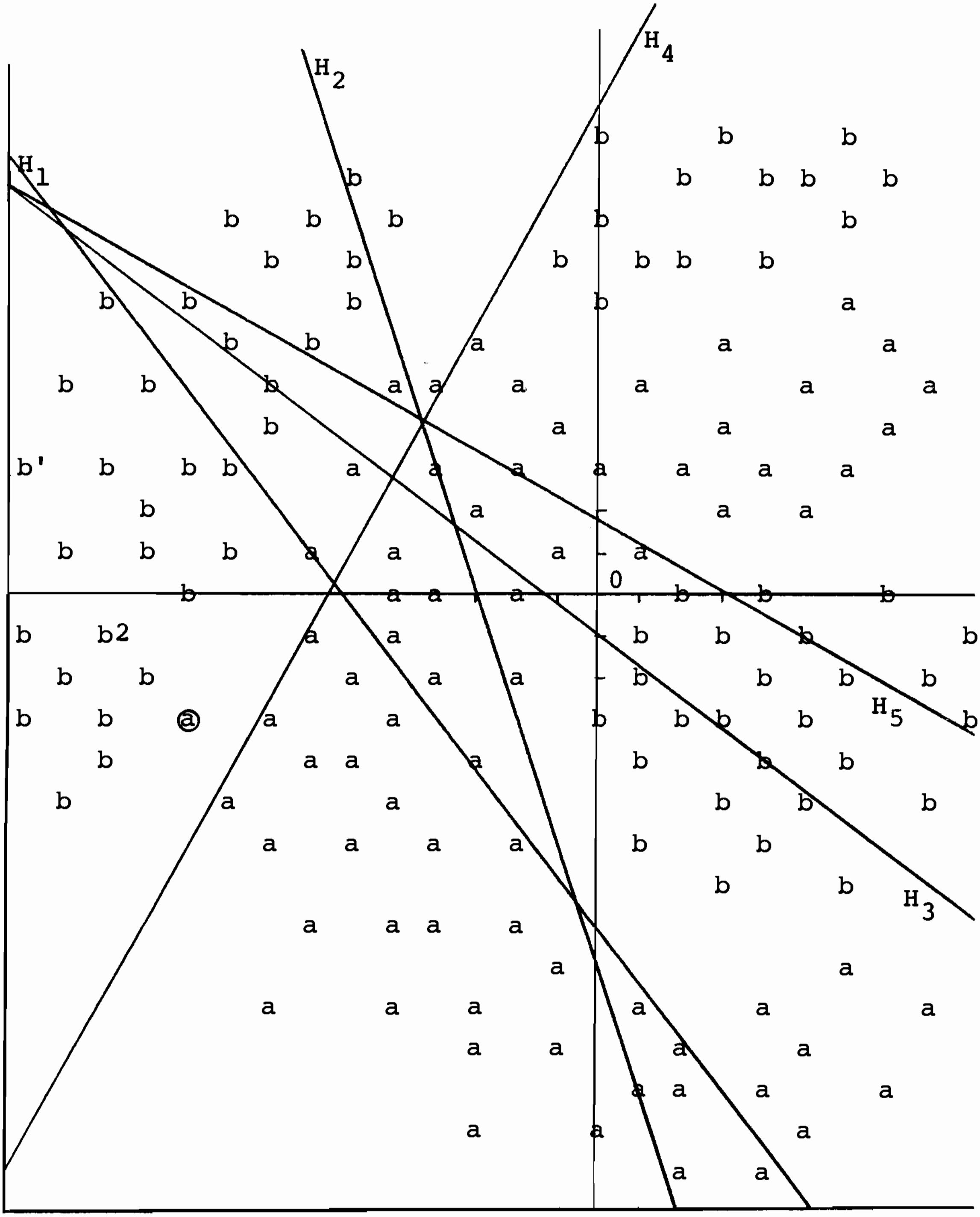


figure 6.6 Initial Stage

For Stage I we set $I_g = 10$ and $c = .1/\|X_i\|$. $P_a = P_b = 1/5$. For Stage II the algorithm ALCONONE has parameters $\beta = 0$, $\alpha = .01$, and $p = 8$, and $I_s = 3$.

After obtaining control in Stage I, ALVARY located the committee in the position of the lines indicated in figure 6.7 which makes one error. The weight vectors are,

$$W_1 = (-0.9895, -0.1443, -0.5481)$$

$$W_2 = (\ \ .9253, -0.3793, \ \ 6.9520)$$

$$W_3 = (-0.0567, -0.9984, -6.9201)$$

$$W_4 = (\ \ .4038, -0.9148, \ \ 5.3044)$$

$$W_5 = (\ \ .0733, \ \ \ .9973, -1.0009)$$

It took 20 iterations for ALVARY to reach this state and since there was no more improvement in the voting record it transferred control to Stage II. This process was repeated four times, i.e. weight vectors W_1, W_2, W_3, and W_4 were all successively recreated and adjusted. Each time ALVARY gave control to Stage II the machine $C_{K,L}$ was in a position that never made more than two errors, and on transfer from Stage II to Stage I the committee made an average of four errors. On receiving control the fifth time ALVARY reached a state where it no longer made any adjustments on the committee. The weight vectors of this state are as follows,

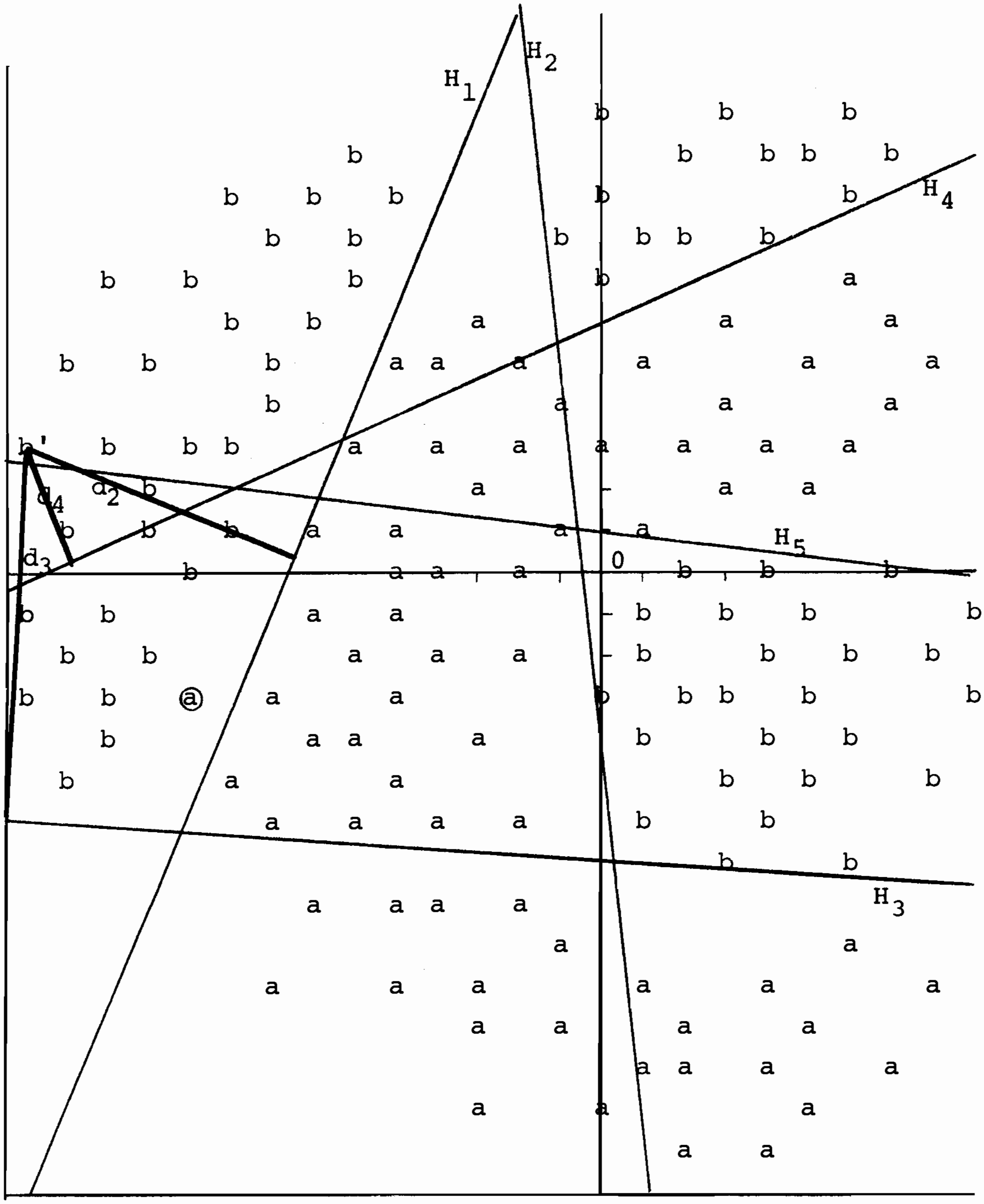


figure 6.7   1 ERROR Using ALVARY

$$W_1 = (-0.9986, -0.0534, -0.8200)$$

$$W_2 = (\ .8735, -0.4968, \ 7.4098)$$

$$W_3 = (-0.1483, -0.4868, -7.0432)$$

$$W_4 = (\ .5092, -0.8606, \ 4.1149)$$

$$W_5 = (-0.0587, \ .9983, -0.8871).$$

The only error made by this committee is the ambiguous one. We note that the distances d_1, d_2, and d_3 in figure 6.8 are relatively equal and relatively large according to Property 2 of ALVARY.

In contrast Ridgway's algorithm with the same starting position reached the state of one error after 60 iterations. In further iterations the weight vectors migrated through positions which made from one to five errors.

We thus arrive at the following conclusions:

i. TEEMAL is stable and is not effected significantly by noisy or abnormal samples.

ii. The process CREATE and the algorithm ALCONONE choose a very good weight vector.

iii. Since in most practical cases we would not expect to have a possible complete solution, the process TEEMAL will prove more effective than previous known algorithms.

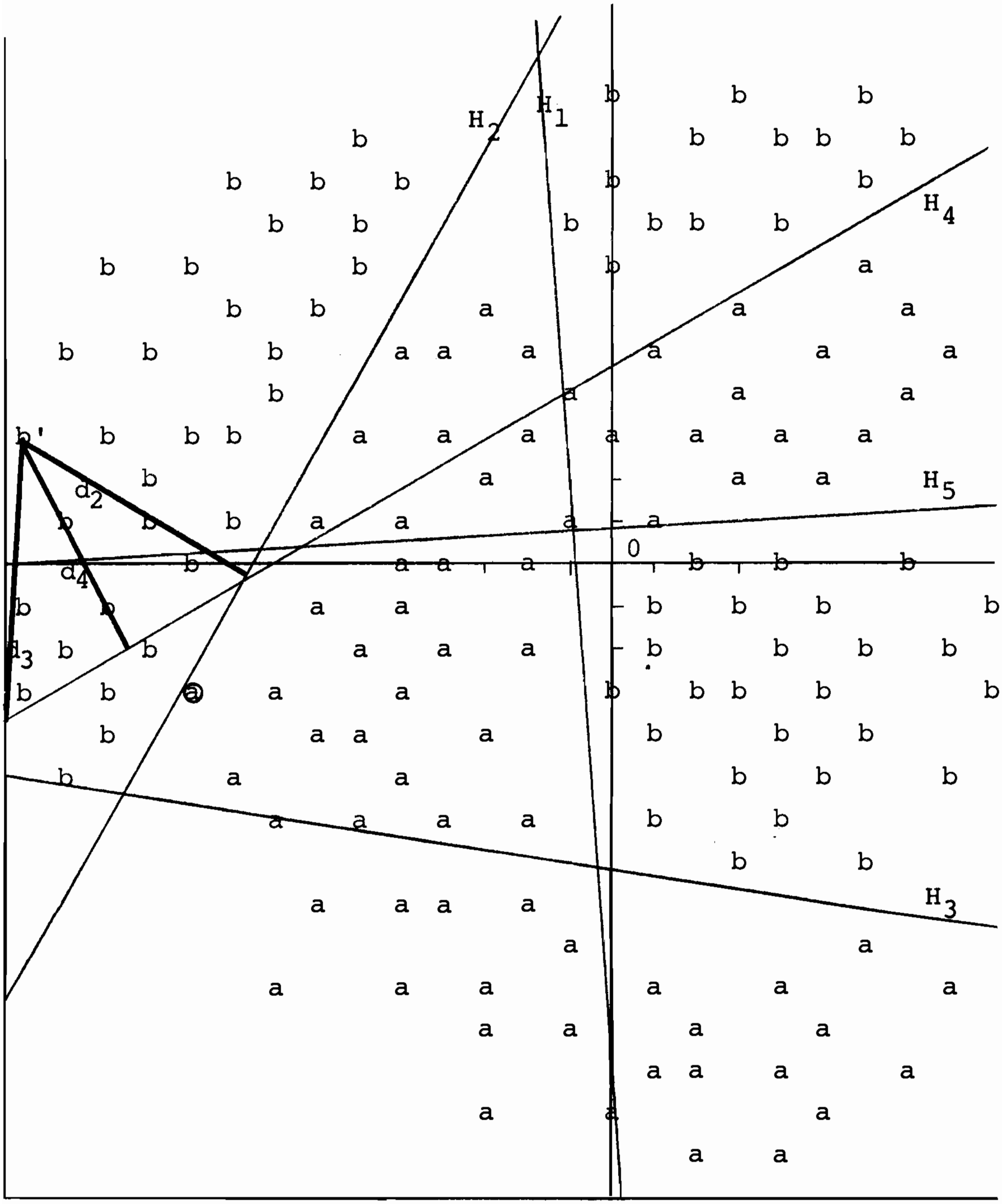


figure 6.8   1 ERROR AFTER 4 calls to Stage II

2. Hand-printed Letters A and R.

Our next example uses real data which consists of two hand-printed letters of the alphabet, A and R. The data used was generated by Munson, Stanford University, and consists of the hand-printed characters on a 24 x 24 grid. Thus each sample is represented by a 576-dimensional vector with binary components. Since use of all the components would envolve a considerable amount of computer time, which was not available for us, a reduction of the number of variables was deemed necessary.

The program ABIOSOFOS, a general learning program of E. Gagliardo, learned to discriminate A from R on this same data. In so doing it created a formula using 27 of the original variables. The ones chosen are indicated on a 24 x 24 grid in figure 6.9 and are as follows: variables 36, 82, 89, 151, 157, 180, 184, 207, 223, 224, 231, 247, 283, 299, 300, 302, 304, 305, 307, 325, 326, 328, 329, 330, 352, 353, 354. These variables were used in the same order to form vectors in 28-dimensional space with the 28th component always equal one as usual. The 240 samples used for training are listed in Appendix I, Samples Set I, plus 27 samples of each A and R used for testing.

With the data in this form TEEMAL was directed to locate a committee, $C_{K,L}$, with $K = 5$ and L equal

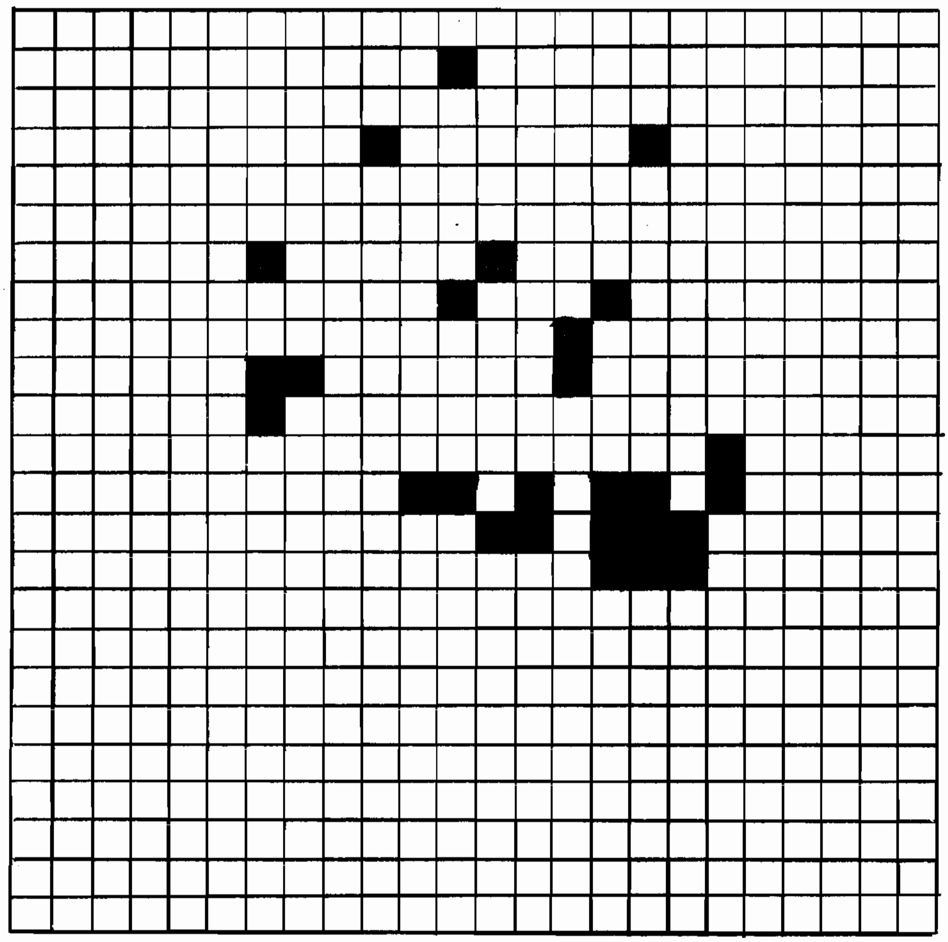


figure 6.9 27-Variables Selected by ABIOSOFOS

to the majority logic. After the initial stage, $C_{K,L}$ was made up of the weight vectors listed in Appendix I, Vector-Set I, which made 15 errors on the training data.

On transferring control to Stage I, ALVARY with parameters $P_a = P_b = 1/5$ and $c = .01/\|X_i\|$ trained for 34 iterations after which $C_{K,L}$ was made up of the weight vectors listed in Appendix I, Vector Set II, which made 0 errors on the training samples and 7 errors on the 54 test samples.

The same configuration of $C_{K,L}$ after the initial stage was used as a starting position for Ridgway's algorithm. With the coefficient of adjustment, $c = .01/\|X_i\|$, the same as for ALVARY, after 36 iterations the algorithm located a machine which made 0 errors on the training set and 8 errors on the test set. The weight vectors for this $C_{K,L}$ are listed in Appendix I, Vector Set III.

To further illustrate TEEMAL's performance on this data another selection of variables, again due to ABIOSOFOS of E. Gagliardo, was used. In this case the number chosen was 19 as indicated on the grid of figure 6.10 and are enumerated as follows: variables 33, 83, 127, 177, 223, 245, 275, 283, 285, 304, 306, 327, 329, 339, 354, 363, 401, 410, 568. A list of the 240 training samples and the 54 test samples are listed in Appendix I, Sample Set II.

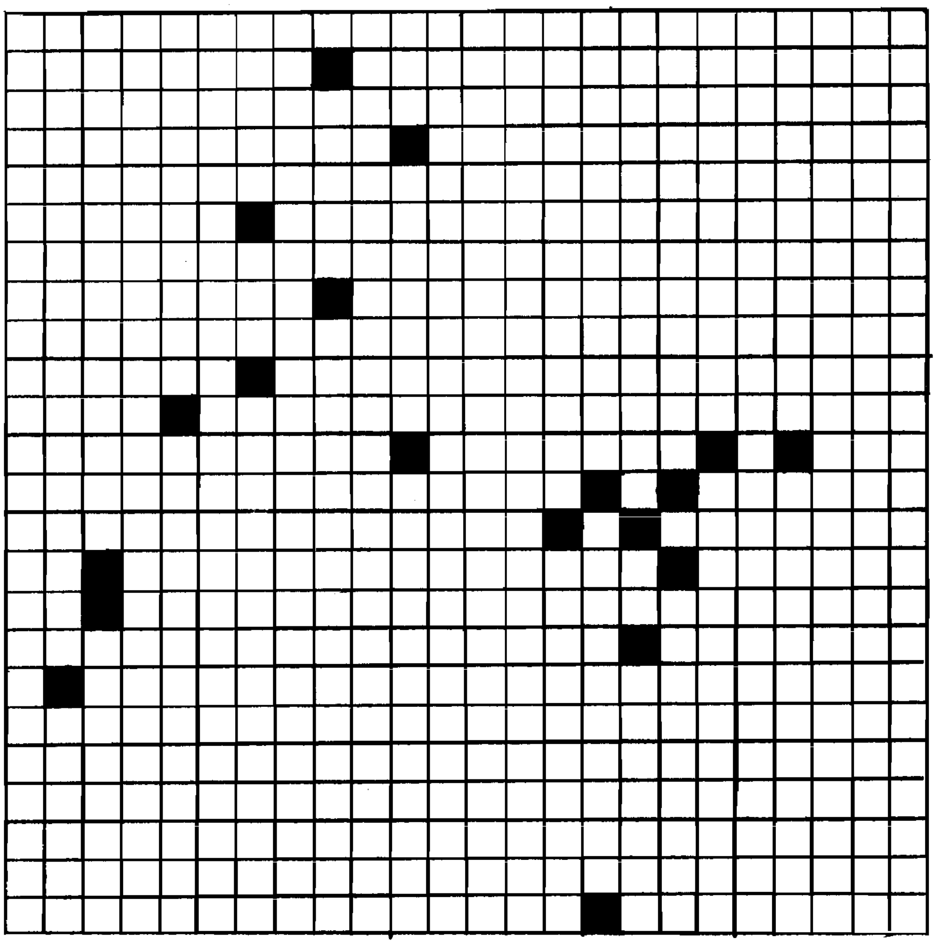


figure 6.10 The 19 Variables Selected by ABIOSOFOS

After the initial stage of TEEMAL $C_{K,L}$ made 17 errors on the training data. A list of the weight vectors of $C_{K,L}$ at this stage is given in Appendix I, Vector-Set IV.

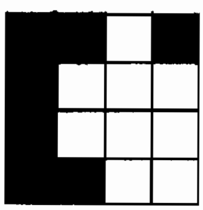
Then with parameters $\beta = 0$, $\alpha = .001$, $p = 8$, $I_s = 1$, $I_g = 10$, $P_a = P_b = 1/5$, and $c = .01/\|X_i\|$ TEEMAL transferred control between Stage I and Stage II five times. On two occasions TEEMAL transferred control to Stage II because no adjustments were made on a particular iteration of ALVARY. This was the case even though $C_{K,L}$ made from 13 to 15 errors on the training samples. At the end of these five transfers $C_{K,L}$ made 15 errors on the training data and 8 errors on the test data. The weight vectors of this $C_{K,L}$ are listed in Appendix I, Vector-Set V.

Again the same initial position of $C_{K,L}$ was presented to Ridgway's algorithm with $c = .01/\|X_i\|$. After 30 iterations $C_{K,L}$ made 8 errors on the training data and 8 errors on the test data. It's weight vectors are listed in Appendix I, Vector-Set VI.

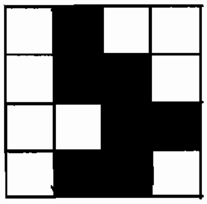
On changing the parameters P_a and P_b of ALVARY such that both were larger, ALVARY performed more adjustments and located a machine which made as few as 7 errors on the training data. When these same machines were used on the test data they made at least 8 and some times more errors. We thus conclude that this further learning, as in the case of Ridgway's algorithm, is not discovering new, general characteristics but rather singular, prejudicial ones.

3. Disconnected-connected 4 x 4 Arrays.

Given a 4 x 4 array of squares with each array containing seven black squares. An array will be called disconnected if the seven black squares are neither face-wise nor corner-wise connected. A disconnected array will be represented by a 17-dimensional vector with binary components, a 1 standing for a black square, with the 17^{th} component equal to 1 for all samples. Thus a pattern vector representing a disconnected array would be the following.

X =  = (1,1,0,1,1,0,0,0,1,0,0,0,1,1,0,0,1)

An array will be called connected if the seven black squares are all face-wise connected. Thus a pattern vector representing a connected array would be the following.

$X =$  $= (0,1,0,0,0,1,1,0,0,0,1,1,0,1,1,0,1)$

This data was randomly generated by E. Gagliardo. In the experiment we have used 240 samples for the training set and an equal number for a test set. A listing of the pattern vectors is given in Appendix II, Sample-Set I.

The initial stage of TEEMAL located a committee machine $C_{K,L}$ with $K = 5$ and L equal to the majority logic, which made 35 errors on the training set. The weight vectors for this committee are listed in Appendix II, Vector-Set I.

With parameters $\beta = 0$, $\alpha = .001$, $p = 8$, $P_a = P_b = 1/5$, $c = .01/\|X_i\|$, $I_s = 1$, and $I_g = 10$ TEEMAL was allowed to run until it had gone from Stage I to Stage II five times. i.e. each weight vector was hired and rehired once. At this $C_{K,L}$ made 23 errors on the training data and 47 errors on the test data. That is approximately 10 and 20 percent respectively. The weight vectors of this machine are listed in Appendix II, Vector-Set II.

Ridgway's algorithm was given the configuration of weight vectors listed in Appendix II, Vector-Set III which make 33 errors on the training set. After 80 iterations the algorithm was trapped in a static position which make 24 errors on the training set and 70 on the test set, or 10 and 30 per cent respectively. The vectors involved in this machine are listed in Appendix II, Vector-Set IV.

From the above then we can see an improvement in TEEMAL in locating a committee machine. (i) TEEMAL did not get involved in a trapped state. (ii) The discrepancy in the percentage of errors in training and test sets has been significantly reduced.

4. Absence-presence of Code 1101.

In this experiment the pattern vectors will consist of strings of ten binary digits. An element will belong to Class A if it does not contain the code 1101 in sequence. If it possesses the code somewhere in the string then the element will belong to Class B. We have 120 samples of Class A and B for training and an equal number for testing. A listing of both sets is given in Appendix III, Sample-Set . As usual the 11^{th} component is always equal to 1.

The initial stage of TEEMAL created a machine $C_{K,L}$ with $K = 5$ and L equal to the majority logic which made 79 errors on the training data. The weight vectors of this committee are listed in Appendix II, Vector-Set I.

Then with the parameters $\beta = 0$, $\alpha = .001$, $p = 8$, $P_a = P_b = 1/5$, $c = .02/\|X_i\|$, $I_s = 10$, and $I_g = 10$ TEEMAL proceeded to use Stage I and Stage II five times, again each weight vector being regenerated once. At this point $C_{K,L}$ made 18 errors on the training data and 69 on the test data, or 8 and 29 percent respectively. The weight vectors of this machine are listed in Appendix III, Vector-Set II.

The same initial configuration was also given as a starting position for Ridgway's algorithm with $c = .02/\|X_i\|$. After 150 iterations the machine settled in a trapped state which consisted of the weight vectors listed in Appendix III, Vector-Set III. This state made 22 errors on the training samples and 97 on the test samples, or 9 and 40 percent respectively.

Again we note the improvement in the performance of TEEMAL in the two areas of major concern.

5. 3 x 3 Arrays of Michalski.

Our next experiment involved the data due to Michalski [12]. The data consists of 3 x 3 arrays with entries belonging to the integers mod 4. There is a slight change in the data that we are using and that found in the paper of Michalski. The 5^{th} element of F^1 is (2,2,2,2,2,2,2,2,2,1) and the 9^{th} element of F^0 is (0,0,0,1,0,1,3,0,3,1) written as 10-dimensional vectors with the 10^{th} component equal to 1. Thus we have the following sets of vectors:

F^1	F^0
(0,2,0,3,0,3,1,3,1,1)	(2,1,1,1,1,0,3,1,2,1)
(2,1,3,0,3,0,2,1,2,1)	(1,0,1,0,0,1,0,0,1,1)
(1,2,1,2,2,2,0,2,0,1)	(2,3,3,2,3,3,3,3,2,1)
(1,3,1,2,3,2,1,2,0,1)	(3,0,3,0,3,0,3,2,3,1)
(2,2,2,2,2,2,2,2,2,1)	(2,0,3,1,2,1,2,1,3,1)
(1,3,3,1,2,2,0,2,3,1)	(0,3,0,3,3,3,0,3,0,1)
(1,0,2,0,2,3,2,3,3,1)	(1,2,0,2,2,2,0,2,1,1)
(1,1,2,1,2,2,2,3,2,1)	(1,1,1,1,1,1,1,1,1,1)
(0,1,3,1,2,3,2,2,2,1)	(0,0,0,1,0,1,3,0,3,1)
(1,0,2,0,2,3,2,1,3,1)	(0,0,0,0,0,0,0,0,0,1)
(3,2,3,0,3,2,1,0,3,1)	(1,0,0,0,3,0,0,0,1,1)
(2,2,3,1,3,1,1,0,3,1)	(1,1,0,1,0,0,0,3,3,1)
(1,3,2,0,3,2,1,1,3,1)	(0,1,1,3,0,3,0,0,0,1)
(2,2,2,0,2,3,1,0,2,1)	(1,1,1,0,0,0,3,0,3,1)
(3,2,3,2,3,2,0,1,0,1)	(0,0,3,0,1,1,0,0,3,1)
(2,2,2,1,2,2,0,1,0,1)	(3,2,1,2,2,3,3,3,2,1)
(3,2,1,2,3,0,1,1,1,1)	(2,3,2,3,0,3,0,3,2,1)
(1,2,3,1,1,3,1,0,1,1)	(3,1,1,0,3,1,0,3,1,1)
(2,3,2,1,2,2,1,1,0,1)	(3,1,3,1,0,0,3,1,3,1)
(3,2,3,2,1,0,1,0,1,1)	(2,1,1,1,2,0,2,0,2,1)

TEEMAL was now directed to locate a committee machine $C_{K,L}$ with $K = 3$ and L the veto logic. It discovered the following weight vectors which serve as a complete solution to the problem.

$$W_1 = (.0912, .5826, .3854, .0126, .4216, .5325, -0.0762, .1672, -0.0926, -2.6298)$$

$$W_2 = (-0.2417, .2321, .5447, -0.2471, -0.3392, .0323, .1207, -0.6274, .0754, 1,8668)$$

$$W_3 = (-0.5446, .2322, .2269, -0.4616, -0.0196, .0851, -0.1846, -0.5823, -0.0642, 2.7856).$$

Finally using the five variables selected by the algorithm of Michalski, (shown in figure 9 of [12]) namely the variables 3, 4, 5, 6, 7, of the vectors described above, TEEMAL located another machine of the same size and logic. Its weight vectors are as follows,

$$W_1 = (.3484, .5156, .3684, .5360, .4357, -2.5125)$$

$$W_2 = (.3559, -0.4058, .1056, .1210, .8264, .1162)$$

$$W_3 = (-0.1561, -0.1754, .1243, .2166, -0.9394, 2.0255).$$

BIBLIOGRAPHY

1. Ablow, C.M. and D.J. Kaylor. "A Committee Solution of the Pattern Recognition Problem." IEEE Trans. on Theory, Vol. IT-11, No. 3. July, 1965. pp. 453-455.

2. Agmon, Samuel. "The Relaxation Method for Linear Inequalities." Canadian Journal of Mathematics IV. 1954. pp. 382-392.

3. Block, H.D. "The Perceptron: A Model for Brain Functioning." Reviews of Modern Physics, 34. 1962. pp. 123-135.

4. Brain, A.E. and J.H. Munson. Graphical Data Processing Research Study and Experimental Investigation. Stanford Research Institute Report, No. 22, Stanford Research Institute, Menlo Park, California. 1966.

5. Brain, A.E., et al. "A Large, Self-contained Learning Machine." 1963 WESCON Paper 6.1. August, 1963.

6. Casey, R.G. and G. Nagy. "Advances in Pattern Recognition." Scientific American, Vol. 224, No. 4. April, 1971.

7. Duda, R.O. and H. Fossum. "Pattern Classification by Iteratively Determined Linear and Piecewise Linear Discriminant Functions." IEEE Trans. on Electronic Computers, Vol. EC-13, No. 2. April, 1966. pp. 220-232.

8. Duda, R.O. and R.C. Singleton. "Training a Threshold Logic Unit with Imperfectly Classified Patterns." 1964 WESCON, Los Angeles, California. August, 1964. pp. 25-28.

9. Efron, R. The Perceptron Correction Procedure in Non-separable Situations. Rome Air Development Technical Docmentary Report RADC-TDR-63-533. February, 1964.

10. Kaylor, D.J. A Mathematical Model of a Two-layer Network of Threshold Elements. Rome Air Development Center Technical Documentary Report RADC-TDR-63-534. March, 1964.

11. Mengert, P.H. "Solution of Linear Inequalities." IEEE Trans. on Computers, Vol. C-19, No. 2. February, 1970. pp. 124-151.

12. Michalski, R.S. "A Variable-valued Logic System as Applied to Picture Description and Recognition." IFIP Conference on Graphic Languages, Vancouver, Canada (May 22-26, 1972).

13. Minsky, M. and S. Papert. Perceptrons - An Introduction to Computational Geometry. The MIT Press, Cambridge, Massachusetts. 1969.

14. Motzkin, T.S. and I.J. Schoenberg. "The Relaxation Method for Linear Inequalities." Canadian Journal of Mathematics VI. 1954. pp. 393-404.

15. Nilsson, N.J. Learning Machines: Foundations of Trainable Pattern-Classifying Systems. McGraw-Hill Book Co., New York. 1965.

16. Nilsson, N.J. "Adaptive Pattern Recognition - A Survey." Cybernetic Problems in Bionics, Oestreicher et Moore eds., Gordon and Breach Science Publishers, Inc., New York. 1966.

17. Novikoff, A.B.J. On Convergence Proofs for Perceptrons. Stanford Research Institute Report 36-5, Contract No. 348(00), Stanford Research Institute, Menlo Park, California. January, 1963.

18. Ridgway, W.C. An Adaptive Logic System with Generalizing Properties. Stanford Electronic Laboratories Technical Report 1556-1, Stanford University, Stanford, California. April, 1962.

19. Rosenblatt, F. Principles of Neurodynamics. Spartan Books, Washington, D.C. 1961.

APPENDICES

APPENDICES

These appendices contain the computer output consisting of the data and weight vectors used in the learning experiments described in this paper. All the samples are written as row vectors with binary components. The weight vectors are written as row vectors with real components.

APPENDIX I

Sample-Set I Training Data

Letters A	Letters B
0010111000001111111111111111	0100000011110100000000000001
0010111100001111111001111111	0101001111110000000000000001
0010110000001100011000110111	0101001111110000000000000001
0000001101100111110111000001	0001000000000111000110001001
1110001011011111110101101101	0000000011110110000100000001
1101001011011111111111111101	1011000010110111000110000001
0010111001000111110011111111	0101101111110110000100000001
0000111100100111110111101101	0101001110110100000000000001
0000110100100111110111001001	0100001110110000000000000001
0010001100100011110111101101	1110001101110000000000000001
0010001100100110010101111111	1111000001110100000000000001
0010001100101011000100111111	1110001101010000000000000001
0100101101100111110111101111	0010111101100111000110000001
0000111100100100100111111111	1111001111110111000110000001
0000111100110000110111111101	1111000010010101100111000001
0000111100100111110111001001	1110001101100011000110000001
0010111100100100110001101001	1110001111110011000110000001
0010111100100111110011101101	1111011111010000000000000001
0000111100000111110111111111	0100001101100111000110000001
1000111100100000110011111111	0101110011010110000000000001
1000111100100100110111111111	1101100111110110001100000001
0010110000100111110001000001	0010111100100110000100000101
0000111100100111110111101001	0010110000100110000000000001
0010110000000111110111111101	0110110000100111000110000001
0000111000000110110111101101	0010011101100111000100000001
0010110000001111111111111111	0010111100100111000110000001
0010110000001111110111111111	0110111101100110000100000001
0010101100100111111111111101	0001100101100110000100000001
0000110000000111100111111111	0100101110110110000110001101
0000110100100111110111111111	0101001101110110000000000001
0000111100100111110111111101	0101001111110110000110000001
0100110001000111111111111001	0101001111110111000110000001
0000110100100111110111111111	0101001111110110000000000001
0000111100100001100011001001	0100000010011110000000000001
1000110000000101000100000001	0101001101110000000000000001
0100110011010001000110000001	0100000100110000000000000001
1000110100100111100111111111	0001000011110011100111001101
0000110100100110110011101101	0110000011110111110111001001
0000111100100110110001101101	0010001111110111000110000001
0000110100100101110011101101	0100001110010110000100000001

1100110101100001000111011111
0100000011010111000111001001
0000111000000111110111101101
0000111100101111110111110001
0010110100100000110001101101
1101110011110111000110000001
1100110000100001000010001001
1101110011010111000111001001
0010001100101110001011110011
0000111100101111110111101101
0000011000000000110001111111
0000111100100100100111001001
0010001100100110100101001001
1000111100100000100001001001
0010011000000111110101101001
0010111100100101100010000001
0010000000000100110111111001
0000111100101111111111001001
0000111100101111111111111111
0000011000011111111111111111
0000010000000010000110000001
0000010000000001000110000001
0000110000000111100011001001
0010111100101100111000110111
1000111100100111110101111111
0000111000000110010000110111
0010111000000111110111111111
0000111000000111110001101101
0100110011110111000100000001
0000111100100111100001001001
1100100101100111111111101001
0100110001000001100011000001
1010001101100000110001111111
0010100000000111111111111111
0010001011011111111111111101
0110000001000100010001111101
1101001111111001110001001001
0001111011011111111000111111
0000111100000101111111111111
0000110100100111100011001001

1010001111110110000000000001
1010011111010100000000000001
1110001101100101000111001001
0110001001110111000110000001
1111001111110111000110001001
1111011011110111100011101101
0101001111110100000000000001
0101001111110110000100000001
1101001111110100000000000001
1111000111110100000000000001
0111000011010111000000000001
0010001100110100000000000001
0010111100100110000100000001
0010011111010000000000000001
0011001111110010000100000001
0010011100100110000100000001
0000110100000110000100000001
0110010100100110000110000001
0010110100100011000110000001
0110111101100001100011101101
0000100111010110000100000001
0101110010010000000000000001
0001001111110110000110000001
0110111101000111000110000001
0101110011010110000110000001
1100110000000111000110000001
0101000011010111000110001001
0101001011010110000100000001
0100000111110111000100000001
0101110010010100000000000001
0101001110110111000100000001
0101100111010110000000000001
0110011101100100000000000001
0010000101100111000110000001
0010000111110111000100000001
1111000110100100001100000001
1001110011010001000010001001
1110000000000111100111001101
1101010111010110000100000001
0000110100100111000110000001

```
0000110100100111110011001001
0000110000001000111111111111
0000110000001111111111110111
0010110000001111111110010111
0010001100100110110001101101
0010001100101111111111111111
0010111000000111110001101101
0010111001000111110001101001
0010111000000111110001111101
0001111011010000110001111111
1010000000001111111110110111
0010111000000111110111111111
0010110000001111111000110111
0000111100101110100001001001
0010111000000111110111111111
0000111100100100110001101101
1000110100100100110001101111
1100001001110111110001101101
1000111100000100110110101111
1000110100101111110111001101
1000110100100101110111110101
0010111100101111111111101101
0010110000001000011000111111
0010110000001010111111110111
0000001000001111111111111111
1100000011011111001000110111
1010000000001110110001111111
1110000011011111110101101101
0000100101110000110001101111
0100010011010001100111001101
0100010011010011000110000001
0010111000001110110111111111
0010111000000110110001111111
0010111100101111111111110011
0010100000000111111111111101
0010000000001111111111110111
0010100000001011111111111111
0000110000100111000111001101
0001000001000100000000000001
1100110111110001110011101111
```

```
0101011111110011000111001001
1111000011010110000100000001
0100010011110011000110000001
0011000011110111000110000001
0000001100100100000000000001
1110110001000110000100000001
1010001100000100000000000001
0000001111110011000110000001
1110000011010111110110000001
1111000011110111100111001001
1000000011100111000110000001
1011000011010111100000000001
0100011111110110000100000001
1111001111110011000111000001
0101001111010110000110000001
1111001111110011000111001101
1001100111010000000000000001
1101101111110110000000000001
1100100111110111000100000001
1111001111110110000100000001
1111001111110110000100000001
1110000011110110000100000001
0010110000001110000000000001
0010100011011111110100000001
0010000001010111110100000001
1100000101100110000110001001
1100010011110111000110000001
1111001111110111000110000001
1101001111110110000100000001
0101101111110000000000000001
0100110011110000000000000001
0000110000010111100111001101
0010111101110111000100000001
1010000001000110110111000001
1010001011010100000000000001
1000100100010111000110000001
1000001111000110000110000001
1010000001000111110111101111
1000001001000111110111001001
1101001111110111000111001001
```

Sample-Set I Test Data

0000111100100110110101101101	1110011101100001000010000001
0000111100100100110001101101	1110011100100101000011001001
0000111100100111110111101101	1110011100100111000100000001
1111000011011000001000010111	0010110000100111000100000001
1100001111110100111111111111	0000000001110111000000000001
0000011100100111110111101001	0010110101100111000110000001
0010011000000111110111111101	0010111101110111110111101111
0010111100100111111111111111	0010110000101111110111100001
0010110001000100100011111111	0010111100101111110111101101
0100010001000111110111110001	0011000011110110000100000001
1100110101110001110011101101	0110011101100111000110000001
1100110001000001000111101111	0100111100100110000000000001
0000111000000110110111111111	1100001011011011111111111001
0000111100100111110111000001	1111001111110110000000000001
0010001000000110110001111111	1011001111110000000000000001
1000111101100101110011111111	0010111100100101000110000001
0010111100100100110001111111	0110111101100111000110000001
1100110101100101110011111111	0010111100100011000110000001
0010111100100111110111101101	0001001111110001000110001001
0010110000000111101101111111	0000001101100110000100000001
0010110000000111111111111111	0010011101100101000110000001
0010011100101111111111111111	1111001111110110000110000001
0010111100100110110101111111	1111111111010111000110000001
0010111100101111111111111111	1110001100100111100111001001
0010101000000111110111001001	0111000010110111110001101111
0010001000100111110111000001	1110000011011111110111100001
0010110000000111110111101101	1111001111110111100111101101

Vector-Set I

-0.114364	-0.212768	-0.077964	-0.193705	.226075
.213564	.000644	-0.040461	-0.238746	-0.273991
-0.064982	-0.265034	.133066	-0.010629	-0.051260
.122314	.270779	.275653	.120991	-0.023067
.075122	.264295	.295892	.224930	.281628
.249647	.194920	-0.645126		

-0.123397	-0.210077	-0.107162	-0.203253	.195364
.213342	-0.040384	-0.088076	-0.213067	-0.232645
-0.074319	-0.255513	.157733	-0.068738	-0.071008
.107917	.275264	.305669	.125542	-0.027723
.071960	.283610	.289403	.227128	.291092
.226481	.188246	-0.460120		

-0.095816	-0.158260	-0.130859	-0.167526	.217651
.266997	-0.030035	-0.097841	-0.158390	-0.194780
-0.075809	-0.217109	.183199	-0.017244	-0.026913
.168830	.286676	.314425	.133434	.009594
.093419	.311953	.300737	.237216	.282195
.215465	.186264	-0.282013		

-0.093028	.199842	-0.359194	.094011	-0.070349
.068955	-0.109716	-0.073640	.320564	.170656
-0.002578	.236533	.059378	-0.185787	-0.187165
-0.110248	-0.384019	-0.141499	.032235	-0.089082
-0.191116	-0.270889	-0.130555	.032235	-0.320666
-0.321485	-0.051796	.738181		

-0.182426	.162689	-0.214701	.154903	-0.069868
.026946	-0.037616	.013916	.407455	.227377
.168188	.294695	.097649	-0.032969	-0.013606
-0.056114	-0.340107	-0.066483	.019363	.044482
-0.192489	-0.262656	-0.147234	.019363	-0.343407
-0.372755	-0.093397	1.006164		

Vector-Set II

-0.208293	-0.292317	-0.137134	-0.207587	.199411
.186003	.000690	-0.011210	-0.255857	-0.347211
-0.026772	-0.305462	.142602	-0.043541	-0.087084
.034630	.215168	.273975	.129662	-0.056870
-0.015944	.218936	.263514	.241050	.205362
.171088	.187456	-0.755661		

-0.153231	-0.209842	-0.117018	-0.193004	.195079
.223022	-0.000372	-0.038029	-0.192819	-0.202385
-0.014335	-0.235212	.167540	-0.078654	-0.080921
.107751	.294914	.325282	.125388	-0.027701
.061838	.303250	.289049	.226850	.290714
.206229	.188016	-0.439615		

-0.161884	-0.040756	-0.158298	-0.065799	.117745
.209923	-0.104454	-0.206495	-0.223744	.010651
-0.058143	-0.255495	.192233	.189104	.015328
.319199	.300813	.329930	.140014	.074621
.019496	.264891	.315567	.248914	.049786
.226091	.195449	-0.101871		

-0.072111	.187313	-0.373094	.103022	-0.028552
.009176	-0.157248	-0.161370	.295206	.117932
-0.132532	.242829	.078262	-0.203158	-0.283229
-0.108734	-0.358002	-0.119384	.031716	-0.186725
-0.198042	-0.195933	-0.128455	.031716	-0.244907
-0.316313	-0.050963	.676850		

-0.186105	.165971	-0.208830	.158028	-0.061076
.037691	-0.028173	.024399	.415674	.231964
.191984	.300639	.099619	-0.013231	.006523
-0.047045	-0.336766	-0.067824	.019753	.065783
-0.186170	-0.247550	-0.150204	.019753	-0.329931
-0.370072	-0.095281	1.046864		

Vector Set III

-0.144364	-0.242768	-0.127964	-0.193705	.186075
.173564	-0.019355	-0.060461	-0.238746	-0.313991
-0.084982	-0.285034	.133066	-0.060629	-0.101260
.052314	.200779	.255653	.120991	-0.073067
.005122	.194295	.255892	.224930	.211628
.179647	.174920	-0.715126		

-0.113397	-0.210077	-0.107162	-0.203253	.205364
.223342	-0.030384	-0.078076	-0.213067	-0.222645
-0.064319	-0.255513	.157733	-0.048738	-0.061008
.127917	.285264	.315669	.125542	-0.017723
.091960	.293610	.289403	.227128	.291092
.226481	.188246	-0.440120		

-0.147449	-0.033862	-0.130859	-0.088065	.107112
.265365	-0.100035	-0.167841	-0.240022	.013588
-0.091411	-0.218741	.183199	.177154	.007485
.267198	.286676	.314425	.133434	.053992
.001787	.251953	.300737	.237216	.086165
.215465	.186264	-0.073646		

-0.103028	.189842	-0.359194	.094011	-0.110349
.028955	-0.109716	-0.103640	.320564	.170656
-0.002578	.236533	.059378	-0.225787	-0.227165
-0.120248	-0.384019	-0.141499	.032235	-0.129082
-0.201116	-0.270889	-0.130555	.032235	-0.320666
-0.321485	-0.051796	.698181		

-0.182426	.162689	-0.204701	.154903	-0.059866
.036946	-0.027616	.023916	.407455	.227377
.188188	.294695	.097649	-0.012969	.006394
-0.046114	-0.330107	-0.066483	.019363	.064482
-0.182489	-0.242656	-0.147234	.019363	-0.323407
-0.362755	-0.093397	1.026164		

Sample-Set II Training Data

Letters A

00000011011110101001
00000011011010101001
00000011001010101001
01010000010100000001
01011011011110001001
00101011011110001001
00010010011010101001
01010010010110001001
00000010010100000001
00000000010110001001
00000010001010101001
00000001001010101001
01010010011110101001
00000010010110101001
00000000010110001001
00000010011100000001
00000010010110000011
00000010011110000001
00000010011110101001
00000000010110101001
00000010011110101001
00000010010100000001
00000010011110000001
00000010011110000001
00000010010010001001
01000011011110101001
00000011011110101011
00000010011110001001
00000010010110100001
00000010010110101001
00010010011110001001
01010010011110000001
00000010010110101001
00000000010100001011
00000010000100011001
11011000000101011011
01010010010110101001
01000010010110001001
00000010010010001001
00000010010110001001

Letters R

01011010000000000001
00111100000001010001
01111110000001010001
00110010000100001001
10111010000000000001
10101010000000000001
00111010000000000001
01111010000000000001
01001010000000001001
01010000000000000001
10110010000000000011
10010010000000000001
00010010000100000001
00111010000000000001
01001100010100000001
01010010000000000001
11011010000000000001
11111000000000000001
01010010000000001001
01111010000000000001
11111010000000000001
01000010000000001001
01000010000000000001
01000010000000000001
00000010000000000001
00000010000000000001
01010010000000000001
00110010000000001001
01101110000100001001
01000010000000001001
01111010000000000001
01111010000100001001
01101010000000000001
01001111000000001001
11110010000000001001
01000110000000000001
00111010010100001001
01011010010100001001
00011010000000000001
01001110000000000001

```
01010000000110101001
10011010000100000001
00000010011110000001
00000011011110000001
00000010011110001001
11111110000101010101
01010000000100000011
11011000000100000001
00000011101010100001
00000011010110001001
00000000011110101001
00000010010100000001
00000000010100001001
00000000010100000001
00000010010110000001
00000010010100000001
00000010010110000001
00000011011100000001
00000011011110100001
00000011111110101001
00000000000100000001
00000000000100000001
00000010010100001001
00000011011010101001
01000010011010101001
00000010001010101001
00000010011110101011
00000010010110000001
01011010000001010001
00000010010100000001
01010000011110000001
01010010000000000001
01010000011110101001
00000011111110101001
01011011111110001001
01010010001010001001
01111011010100000001
00101011011010101001
00000010011110101001
00000010010100000001
```

```
10011010000000000001
10111000000000000001
01010010000100000001
01010010000100000001
10111010000100001001
11111010010110001001
11111010000000001001
11101010000000000001
10111010000000000001
10111010000000000001
01101000000000000001
00000010000000000001
00000010000000000001
01011010000000000001
01111010000000000001
00010010000000000011
00000010000000000001
01000010000000000001
00010010000000000001
01010010010110001001
00011010000000000001
00101010000000000001
00001010000000000001
01010010000000000001
01111010000000000001
01010010000100000001
01011010000100001001
00001000000000000001
01001000000000000001
11101110000000000001
01101100000000001001
11101010000000000001
01000010000000000001
01010010000100001001
00011010000000000001
10101010000000000001
10101010000100001001
10000010010100001001
01111010000000000001
00010010000000000001
```

```
00000010010100000001
00000011011110101001
00000011011110101001
00000011111100101001
00000010010010001001
00000001011110100001
00000010010110001001
00010000010110000001
00000010011110000001
00011000011010101011
01000011011010100001
00000010011110101011
00000011011010101001
00000011010100000001
00000010011110101011
01000010010010001001
01000010010110101001
01010010010010001001
01000010010110101001
01000000101010001001
01000010011110001001
00000011011110001001
00000011001010101001
00000011111110101001
00000011111110101001
01011001101010101001
01010011111010101001
01011011011110000001
01010000010110101001
01011000010100001001
01011010000000000001
00000011011110101011
00000010011010101001
00000011011110100001
00000010011110001001
00000011111110101001
00000010011110100001
00000010000100000001
00110110000001011011
11011000010110101001
```

```
01111010000100001001
01111010000000000001
01011010000101000001
00111010000000000001
01000010000000000001
00110010000000001001
00010010000000000001
10011010000000001001
11111010010000000001
11101010010100001001
10101110000001011001
11101110010000000001
01001110000000000001
01111010000100001001
01111110000000001001
10111010000100001001
00101110000000000001
10111110000000000001
01001100000000000001
10111010000000000001
01111010000000000001
01011010000000000011
00000010000000000001
00011011011000010001
00010000010000000001
01010010000100001001
01011010000000000011
01111010000000001001
11101010000000001001
01101100000000000001
11001110000000000001
00000010010100001001
00010010000000000001
01010000010100000001
11011000000000000001
10000110000000000001
10011010000000001001
10010110010110100001
11000110010100000001
11101110000100001001
```

Sample-Set II Test Data

```
00000010010010001001
00000010010110001001
00000010010110001001
10111001001000101001
01011000011110101011
00000010010110000011
01010010010110001011
00000010011110101001
00000010010110100001
01010010011110000001
01010010010110001001
01010000001101000001
00000010011110100001
00000000011100000001
00000010011110100001
01010010011110101001
00000010011010101001
01010010011110101001
00000010010110000011
00000011001110101001
00000011011110101001
00000011011110101001
00000010011110101001
00000011011110101001
00000010010100000001
00000010011100000001
00000010010110000001
```

```
00000010000000000001
00010010000000000001
00010010000100000001
01010010010110101001
00000011011110000001
00000011011110001001
01011110000000000001
01010010000000000001
01000010000000000011
01011001011110000001
11011010000000000011
10101010000000000001
00000010000100000001
01010010000000000001
00010000000000000001
01111000000100001001
00110010000000000001
00010010000100000001
11101010000000000001
11101010000000000001
11010010010100001001
01101100010110101001
01011001011110000011
10111010010110001001
10111000000000001001
01010010000100000011
11010010000000000001
```

Vector-Set IV

-0.166808	-0.197974	-0.237638	-0.173708	-0.307347
-0.152477	-0.043399	.169875	.048080	.404734
.298095	.429115	.407582	.002123	.249586
.006931	.169144	.004969	.039530	-0.225362

-0.180310	-0.164593	-0.266235	-0.238339	-0.303125
-0.112471	-0.069549	.179288	.057160	.317615
.319767	.359118	.450122	.049025	.284756
.020035	.205046	.017373	.083574	-0.308205

-0.190365	-0.108158	-0.279235	-0.161072	-0.266422
-0.123690	-0.125702	.217068	.059339	.319091
.348734	.377085	.435046	.103121	.280621
.090517	.166473	.030045	.119758	-0.246081

-0.006830	.333025	.153243	.233697	.326352
-0.003415	-0.153681	-0.029750	0	-0.610633
-0.095913	-0.139224	-0.248860	.264653	-0.113118
.168740	-0.247906	.066163	.198490	.076100

-0.041759	.322937	.175388	.250768	.377437
-0.020880	-0.161992	.007173	0	-0.551007
-0.051290	-0.176125	-0.275610	.233851	-0.137805
.182561	-0.277002	.058463	.175388	.169654

Vector-Set V

-0.164128	-0.180484	-0.258174	-0.266977	-0.365058
-0.098848	-0.049258	.118833	.043076	.398476
.247006	.394777	.403248	-0.007402	.242275
-0.046664	.203653	.004962	.020831	-0.271594

-0.162670	-0.177134	-0.255929	-0.239425	-0.309395
-0.097700	-0.068871	.146340	.043973	.431067
.277293	.376008	.411992	-0.006785	.247577
-0.021872	.240464	.005059	.022056	-0.281879

-0.209074	-0.219354	-0.256154	-0.253203	-0.355237
-0.149920	-0.060947	.122041	.043684	.339134
.261870	.368682	.419110	-0.001374	.245820
-0.039835	.225358	.005030	.035553	-0.291164

-0.016146	.296418	.190030	.302181	.458677
-0.027325	-0.101051	.019426	0	-0.537431
-0.043918	-0.176796	-0.231017	.253374	-0.115509
.209456	-0.160649	.063343	.190030	.040462

-0.057342	.278550	.175936	.312434	.368340
-0.049523	-0.193351	.058645	0	-0.507295
0	-0.214069	-0.250220	.234581	-0.125110
.234581	-0.300216	.058645	.175936	.206347

Vector-Set VI

-0.158643	-0.191307	-0.231171	-0.188625	-0.317264
-0.152477	-0.058316	.151793	.048080	.386652
.273346	.438947	.407582	.002123	.249586
-0.017817	.169144	.004969	.039530	-0.240279

-0.232604	-0.137603	-0.266235	-0.278016	-0.268340
-0.179138	-0.121054	.179288	.057160	.218362
.319767	.244128	.410445	.049025	.252875
.073700	.131125	.017373	.083574	-0.324925

.021789	-0.039501	-0.218374	.005153	-0.056314
-0.066163	-0.064702	.220402	.059339	.318341
.348734	.182148	.435046	.118456	.280621
.204122	.079117	.067573	.142920	-0.083939

-0.006830	.312613	.153243	.213285	.326352
-0.003415	-0.174093	-0.029750	0	-0.610633
-0.095913	-0.159636	-0.248860	.264653	-0.113118
.168740	-0.268318	.066163	.198490	.057687

-0.041759	.285775	.175388	.213605	.377437
-0.020880	-0.079378	.007173	0	-0.595729
-0.043731	-0.131649	-0.268051	.233851	-0.137805
.182561	-0.269443	.058463	.175388	.221689

APPENDIX II

Sample-Set I Training Data

Disconnected	Connected
10001101000001111	01001110011001001
00011011000011101	00100111011000101
11000101000101011	01001111011000001
01010001010111001	00000110111101001
11100000101100011	00100110011100101
01110000110110001	01000110111001001
10101000101000111	00000110111100101
00111010100010101	00101111011000001
11001101000110001	00000010101011111
00111011100000011	00000100010111111
11011100000001101	00110001011100011
01100000110011011	00010111000100111
00011000101100111	11110101010000001
10000001110111001	11111010001000001
01100000001110111	10001110100011001
10110011000001101	11001000111010001
11001010100000111	00110110010011001
00110101000111001	11000110001000111
11101000010100011	00010111110010001
00010101100011101	10001100011100011
11000001010100111	00110010011011001
00111000101011001	11000100011000111
10001010000101111	00010011111010001
01110001101010001	10001110001100011
00111111000000101	01100111110000001
11001111000001001	01101110001100001
01000100110111001	00101110110001001
11001101010001001	01001100111000101
01000000111111001	00000011111001101
00100000111100111	00001100011101101
00111011001000101	00100011011101001
00100010101100111	01000111001100101
01010001001011011	01110011001100001
10101000010010111	11101100110000001
00011001001011011	00001000111011101
11010010100100011	11101110100000001
10110100100010101	00001100110011101
11010010000101011	00000011001101111
10110100100110001	01110111000100001
10001001010010111	00000001011101111

00010000101101111
10000000110111101
00100001001110111
10110011000100101
11101101000010001
01111011000000011
11011100100001001
01001000110011011
11010000101110001
10110000110100011
10111000001010101
10100010100010111
00011101000010111
10001011000011011
01010100000111011
11010001010001011
11101011000000101
01111101000001001
11001000110101001
01001101100011001
01000000110101111
00100000101111101
00101011000100111
00110001101100101
11011100101000001
10110011010100001
11101100001010001
10000010110011101
00000101001110111
00001010110011011
00010100001101111
01110011010000011
11010000001001111
10110000010011101
10001001001110011
10010011100110001
11100100000010111
01110010000011011
10011100100100011
00011001110010011

00010101011100101
10001010111001001
01000110001111101
11100011011000001
01001110101010001
00100111010100011
01111100011000001
00000110110001111
00111111010000001
11001111001000001
01000110110011001
11001100011001001
00000010111111001
00000100111100111
00110011011000101
00100110001100111
00100111001001101
01001110010001101
00000101111101001
01001111010100001
01100100111001001
01100010011100101
00101111101000001
00001010111100101
01001100011001101
00100011011001101
01001111001100001
00000011111101001
01100110001100101
01100110110001001
00001100111100101
00101111110000001
00000010111001111
00000100011111101
00100011011100011
00010111001100101
11100111010000001
01111110001000001
10001110110001001
01001100111010001

00110011101000101	00001000111011101
11001100010101001	00000001011101111
00100000111111001	01110011001100001
11001111000000101	00000011001101111
01000101110011001	01110111000100001
00101010001100111	11101110100000001
00111111000001001	00001100110011101
01000000111100111	11101100110000001
01000001010101111	00100110011001101
00101000101011101	01000110011001101
00001011000101111	00000111111100001
01110001101100001	00001111011100001
11101010100000101	01100110011001001
01110101000101001	01100110011000101
11101000110100001	00001111111000001
00001101100011101	00001110111100001
10011110001010001	00001000111111001
10010111010000011	00000001111100111
11010100011010001	01110011001000101
10000110010011011	00100010001101111
00010100011110011	00111111000100001
10000010111010011	11001111100000001
00010110001010111	01000100110011101
10110010011000011	11101100010001001
10110000010100111	01000100010101111
11010000101011001	00100010101011101
10000010100110111	00001111000100111
10111001001010001	00110001111100001
11001010000011011	11101010001000101
00110101000010111	01110101010001001
11011001010000011	11001000111100001
00010100100111011	00001111100011001
01100111010000011	00101111101000001
01101110001010001	01001111010100001
00001110110001011	01100100111001001
01011100111000001	01001110010001101
10000010111001101	00000101111101001
00010100011101101	00001010111100101
10100011011100001	00100111001001101
00000111001110101	01100010011100101

Sample-Set I Test Data

11011000100011001	01000110001101101
10110001000100111	00100110110001101
11111001000010001	00001101011100101
10000000100111111	00100111110100001
00110001000110111	01101100011000101
11001000100011011	01100011011001001
00010000100111111	01001110101100001
11111001000000011	00001011111001001
11100001100010101	01000101011100011
01111000000101011	00101010111010001
10111000100101001	00001110001001111
01001001100010111	01110010111000001
01010001100001111	10001110101000101
10101000000111101	00010111010101001
00101001000111011	11100100111000001
11010001100100101	00000111010011101
11110000001001101	01110101000100011
11110000010001101	11101010100010001
10001001101110001	00001100100011111
10001011100110001	11111000110000001
01100100000011111	10001000101011101
01100010000011111	00010001010101111
00011101100100011	11110001001100001
00011001110100011	00000011000111111
10100100110100011	00101010111000101
01010010101110001	01000101011101001
10100110100000111	01100010111100001
00111000011010101	00001111001001101
10001011001001011	01000111010101001
00011101010010101	00101110101000101
11000001011001011	00001111010001101
01010110000111001	01100100111100001
10110010110001001	01000111001100101
11010100001100101	00101110110001001
10100011110010001	00001100011101101
10001100001110101	01100111110000001
00100011010011011	01001100111000101
01001100001010111	00100011011101001
00010011110001011	01101110001100001
01011100001100011	00000011111001101

10100001010011011
01011000001010111
10010011100001011
01011000001110011
10110010100001011
11010100000110101
10100001110010011
10011100000110101
11001000110001011
00110001001110101
11101011000000011
00010000101111101
10100011000100111
01011100100011001
10000000110101111
01111101000010001
01010100100001111
10100010000111101
00101101000110011
10010001110100101
11100001001010101
01111000010001011
10011000101101001
01001011100010011
00011011001010101
10001101010001011
01010000011111001
11000111000001011
01010100110110001
10100010101100011
00111110000010101
10100000111000111
00001111110000011
00001111001110001
01100110010001011
01010100011001101
10000011111100001
00011100111100001
10100010011001101
01100110001010101

00001111110010001
00001111001100011
01110110010001001
01000100011001111
00010011111100001
10001100111100001
00100010011011101
11100110001000101
01001110011001001
00100111011000101
01001111011000001
00000110111101001
00100110011100101
01000110111001001
00000110111100101
00101111011000001
00001111000100111
00001111100011001
01000100010101111
01110101010001001
11001000111100001
00110001111100001
11101010001000101
00100010101011101
00010111110001001
10001110001100101
00100111010011001
11000100011100101
00100011111010001
01001100011100011
00110010111001001
01001110001000111
00000000101111111
00000001101111111
00110001001100111
00110011000100111
11111101000000001
11111011000000001
11001100100011001
11001000110011001

00111100100101001	01000110001101101
11000011100100101	00100110110001101
01100101100010101	00001101011100101
10101000010101101	00100111110100001
00101001001111001	01101100011000101
01001001110000111	01100011011001001
01010001101001101	01001110101100001
01101010000101011	00001011111001001
10001101000010111	01100010001011101
00011011000011011	01100100010001111
11010100000101011	00011001111100001
01010001010011011	00001111100100011
11010000101100011	01110100010001101
10110000110110001	11100010001001101
10101000001010111	00001111100110001
10110010100010101	10001001111100001
00111101000110001	01110100110010001
11001011100000011	11100010001100011
01010100100011101	00111110100010001
11101000010001011	10001000111000111
00011000101111001	00010011001011101
10000001110100111	10001100010001111
01110001001010101	00010001011111001
10100010000101111	11000111000100011
10000101100010111	10001110001100101
00011010000111011	00010111110001001
10110100000101011	11000100011100101
01010001010010111	00100111010011001
11010001101000011	01001100011100011
10111000010110001	00100011111010001
10101000001011011	01001110001000111
11010010100010101	00110010111001001
01110100100100101	00000010001111111
11100010100101001	00000100110011111
00101100100110101	00010001011100111
10101001110000101	00110111000100011
01001001001011101	11111100010000001
00101001010001111	11110011001000001
01011001001101001	11001110100010001
01000011100101011	10001000111011001

Vector-Set I

.374027	-0.013730	.018387	.288847	.038066
-0.389689	-0.352917	.007350	-0.036448	-0.356885
-0.410359	.001628	.287783	.058849	.004422
.338880	.248314			

.320880	.034258	-0.000403	.288012	-0.049790
-0.397226	-0.390729	-0.004416	-0.036737	-0.375354
-0.404958	.009445	.308985	.066263	-0.070901
.297903	.255897			

.329013	.000049	.113414	.374598	.022915
-0.319013	-0.295984	.135272	.046933	-0.325419
-0.320077	.142856	.305503	.022194	.123813
.436179	.473780			

-0.539157	.200174	-0.035165	-0.443328	.275260
.200860	.127187	.003122	.222566	-0.034035
.149131	-0.026986	-0.326745	.051554	.031192
-0.393839	-0.335122			

-0.405139	-0.113861	.102741	-0.115445	-0.329400
-0.032466	.186231	.268053	-0.204398	.108507
.277211	.487615	-0.422619	-0.059152	.179886
-0.017084	-0.012582			

Vector-Set II

.390025	.143378	-0.006384	.401488	.087516
-0.168970	-0.288317	-0.128063	-0.040602	-0.281238
-0.422239	-0.210084	.392253	-0.120156	-0.134376
.202057	.225314			

.283476	.061909	-0.141313	.256531	-0.014604
-0.366758	-0.421932	-0.005099	-0.047876	-0.379079
-0.352199	.028426	.367501	.112425	-0.085112
.302435	.246342			

.523112	-0.032965	-0.005444	.360333	.076967
-0.095632	-0.233285	-0.170346	.008481	-0.185656
-0.330779	-0.062064	.521607	-0.039449	-0.084135
.261484	.486133			

-0.400504	-0.073580	-0.042569	-0.277122	.332284
-0.346907	-0.130302	-0.148840	.319612	-0.000523
.437515	-0.123465	-0.181629	.006313	.374386
.060627	-0.020070			

-0.136424	-0.213663	.137120	.079887	-0.209187
-0.103197	.257599	.406900	-0.455630	-0.277333
-0.036339	.235102	-0.354458	-0.355253	.059255
.166603	.032636			

Vector-Set III

.304419	-0.016535	-0.002238	.324440	.030740
-0.418942	-0.352695	-0.085322	-0.044845	-0.400533
-0.327183	-0.017093	.309477	.019590	.042597
.354832	.226581			

.362289	.013924	.079627	.313266	-0.036142
-0.381335	-0.371867	.010564	-0.006304	-0.363339
-0.356151	-0.003653	.354769	.011780	-0.053380
.301988	.308145			

.392995	.188676	.090603	.340053	.144786
-0.336846	-0.379276	.122711	.124182	-0.296817
-0.369037	.055179	.276478	.089273	.052274
.250753	.469751			

-0.476499	.002018	.104990	-0.373731	-0.065898
.141448	.143363	.234038	-0.000911	.024573
.000112	.358331	-0.493508	.069885	.272585
-0.266134	-0.286202			

.090886	.243395	-0.090086	-0.370124	.269998
.343554	-0.061438	-0.112865	.264262	.306013
-0.119848	-0.244106	.064835	.281031	-0.242655
-0.447724	-0.011085			

Vector-Set IV

.905459	.018821	.139183	.713348	-0.004615
-0.171455	-0.246629	-0.262098	.202642	-0.294467
-0.433249	-0.087804	.769097	-0.157187	-0.028114
.637674	.509423			

.539066	-0.233563	-0.167861	.348621	-0.071497
-0.239913	-0.442578	-0.166213	-0.183080	-0.469405
-0.285440	-0.180430	.425480	-0.271063	-0.088736
.372699	.165724			

.251574	.047255	.055248	.233987	.074075
-0.725755	-0.414632	.122711	.018116	-0.367526
-0.651879	.019824	.276478	.018563	.052274
.250753	.257619			

-0.865407	.249505	.069635	-0.373731	-0.136608
.176804	.390850	.375459	-0.106977	.342771
.318311	.358331	-0.011706	.175951	.060453
-0.301489	-0.250847			

.196952	.101973	-0.090886	-0.334769	.340709
.272844	-0.061438	-0.183576	.441039	.447434
.162994	-0.279461	.206257	.316386	.075543
-0.200237	.165691			

APPENDIX III

Sample-Set I Training Data

Code Absent	Code Present
00111110011	11101000101
00111001001	01101101101
01111000111	01111010101
11000101001	00011011001
10111001111	00011101101
00001111001	10000110101
00101111001	10100110101
00001001111	00001110111
00000011111	00010011011
10100010001	11010101001
11000010011	11011001111
00111001111	00110100101
10010001101	01101000101
00111100001	11101000011
01111100011	01110100001
00011111101	00111010001
00011111101	00111101111
00111001001	00111101001
01001010101	10100110101
00111001111	00110110101
00101111101	01011011011
11111000001	11010101001
11001001011	11010001101
11100001001	01110100001
11000101101	01101000101
00000111111	00110100001
11111000011	01111010101
01110010011	10001101011
11000100111	00011101101
01100010001	00101110111
01111100101	10100110101
00110001011	10011011011
01111100001	11010010001
11100001001	11010001101
11111000001	00110100001
10101000011	11101010001
10110001011	11101010111
00011111101	00111010001
01111100011	00111010001
00101110001	00011101111

10100011001	00101101001
00000010111	10100110101
00001000111	00000110111
00100011001	00000011011
00110000001	11011001001
10011111001	11010000011
10010010001	00110101001
10010101111	00110100001
10101001101	11010001101
10011111001	00110111011
10101111001	01101000101
10100001111	01110100001
00000111111	00111010101
10101001101	10111011001
10110000101	00001101111
10010111101	00111110101
00000111111	00011110101
10101001101	00000011011
00000010001	11010001001
11111000001	11011001001
01100000011	11110100001
11000101101	01101010011
00000010111	01101000101
01111000101	00111010101
00101111001	00111010111
00111100011	00001101111
00000101111	00111101101
10010101101	00100110111
00000111101	00000011011
11111000001	11011000001
10111001011	01110100101
00101001111	01101000101
01110010011	11011000001
01100010011	00110100101
01110000011	01101101101
01110000111	11101000001
11110000111	01110100001
11111000111	00011010101
11001111101	00101101111
01001000011	00111101001

11111100001	10010110101
10111110001	00111011011
10111111101	00110111011
10111000111	11010001111
10111000111	11011000111
10100010111	00110100101
10110001101	00110111001
01000101101	11101001001
01001111101	00110100101
00111001111	10011011101
10111001101	00001101101
00000010111	01101101111
11111100001	10100110101
01111110011	10110110101
11111000111	00100011011
11111000101	11011000001
01111100001	00000110111
11111000001	11111111011
11001001011	11101000001
10100001101	01101010101
01111000001	00111010001
11111000001	00111010001
00101111001	00001101111
10100011111	11011001001
00001001111	00110100101
00110001001	11101011101
10100000111	01101000101
10100010001	00111010011
10001111001	00111011101
11000001001	00101101101
00000111101	01001101001
00011000101	00100110111
00010111101	00000110111
11100000001	00100110101
00111100001	10100011011
00000010011	11001101001
00011111101	11011001011
00001111001	11011100001
01111001011	11011000011
00101000001	01101101011

Sample-Set I Test Data

10011111001	01110100001
10100100001	01101011011
00111110011	00111110101
00001111001	00101110111
00111001111	10110110111
01111000011	10110011011
01111100001	11011010011
10000010101	11010011001
10101100001	11011000101
11100001101	00111010001
00010111101	11111011011
10111111101	11110110001
01111100001	00101101111
00001111001	00110101001
00110000001	01101101011
00000111101	01101011011
01111000101	00111010001
00001010111	01111010101
01110000001	00101101001
10000010111	00101101001
11100101001	00111110111
01111100101	00000110111
01110010011	01110011011
00100101101	11011001001
10100010011	10001101011
10010001101	00101101001
01000111001	00000110111
00110001101	01011011001
10101001011	10101101011
11001111101	01110110001
10101000101	01101000001
00101000101	11101000001
00111000111	00110110001
00101001111	01101001101
00111001111	11111011011
01100101011	00110100101
10100101011	01110011011
10101010011	11010011001
10101010111	00100110111
10100101111	00100111011

01000100111	10111101011
01000001111	01111010101
10000001101	01011011001
01010001001	00101101101
01100101111	11000110111
00001110001	00000111011
10011111001	00000111011
01001011101	10100011011
01010000001	00101101111
11100001001	10111011011
10000111111	11101011011
00000111111	10011010101
10100101011	01101010111
01010100001	00110100011
01010001011	11101000001
00011001111	11011111011
10001010111	10110100001
00001111111	11111011111
10110001011	11011001001
10111111001	00001111011
10010100111	00000011011
01000000011	11011110111
01000000011	10100011011
01100000011	10111101111
11111111111	11010001111
10111111111	00101110101
10110011111	00000111011
00010101001	01110101001
00110000001	11010010011
00100100001	11000110101
00001000111	11111011111
00110011101	00011111011
11110000001	00011101011
11111100001	10110101011
11111110001	00111010001
10111111001	11010110011
10010111101	11101001011
11001001011	00011011001
10100011111	01101000101
11111100101	00000111011

10100010111	10110100001
10101010011	00001110111
00001011111	11010001111
01010111001	00001101111
01100010011	11011001011
10001110011	01101001111
00011111001	10111011011
00011111001	00110100011
00111000111	00000111011
10111001101	11010101001
00000101101	00100110101
10000100101	00111010111
10100110011	00101101101
00100011111	00110011011
00100001111	00101101001
01000001001	11011110111
00101100101	10111011011
11001110011	00111011111
10101010111	10011011101
11111111111	10000111011
11001011111	01101100011
01100101101	11001110111
00100011111	01101000011
00100001111	10001101001
01000001001	00110100001
01010000011	11101000001
00100111001	11011000011
00000111111	10110101001
00000111111	11011000001
00100101111	11010001001
10111001011	00100011011
11100101011	01011011001
01111000011	10010111011
11110000011	00000011011
00100010011	00000110101
00100010011	00111110111
00101010001	00101101101
11001011101	10001101111
10101001011	00111010001
11000100101	01101011011

Vector-Set I

.452095	-0.292291	.093116	-0.270343	-0.181122
-0.558492	-0.219643	.275071	-0.226395	.329455
.285391				

-0.362931	-0.494602	.128056	-0.166767	-0.092267
.675679	.078687	.070027	.317234	-0.051506
-0.045566				

.518519	.508945	.011417	.034004	.095700
-0.481948	-0.320692	-0.139285	-0.297867	-0.135744
-0.010107				

-0.478075	-0.557544	.231548	.182850	.230909
.356726	.428285	.062253	.044503	.060700
.138178				

.497592	.411504	-0.385666	.295567	.372192
-0.039259	-0.143631	-0.049692	-0.216877	-0.369819
-0.123756				

Vector-Set II

-0.082364	-0.515430	-0.025477	-0.381493	-0.303685
-0.641063	-0.241045	.106588	-0.008944	.092906
.777529				
.188323	-0.118418	-0.193166	-0.335313	-0.238065
.562631	.538710	-0.321256	.156264	-0.098503
-0.095093				
.098267	.028417	.053089	.196383	.139240
-0.554083	-0.698417	-0.253151	-0.045885	.260342
.208280				
-0.410873	-0.532941	.400939	.112856	.199854
.299585	.245929	-0.009793	.421970	-0.073085
.237681				
.034750	.742614	.035322	.134559	-0.069105
.429828	.064805	.303893	-0.369368	-0.073841
-0.420416				

Vector-Set III

-0.147905	-0.212291	.033116	-0.350343	-0.381122
-0.378492	.020357	.395071	-0.046395	.029455
.365391				

-0.282931	-0.474602	.328056	.013233	-0.092267
.695679	-0.121313	.130027	.477235	-0.011506
-0.325566				

.578519	.548945	-0.108583	-0.025996	-0.084300
-0.481948	-0.320692	-0.139285	-0.077867	-0.075744
.029893				

-0.618075	-0.377544	.231548	.382850	.170909
.076726	.168285	.022253	.084503	.020700
.278178				

.397592	.191584	-0.285666	.155567	.352192
-0.079259	.056369	-0.149692	-0.276877	-0.029819
-0.003756				