A Branching Fuzzy-Logic Classifier for Building Optimization

By

Matthew A. Lehar

B.S. Mechanical Engineering
Stanford University, 1999
S.M. Mechanical Engineering
Massachusetts Institute of Technology, 2003

SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY IN MECHANICAL ENGINEERING
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUNE 2005

Signature of Author: _____
Department of Mechanical Engineering
May 26, 2005

Certified by: _____
Leon R. Glicksman
Professor of Architecture and Mechanical Engineering
Thesis Supervisor

Accepted by: _____
Lallit Anand
Professor of Mechanical Engineering
Chairman, Committee for Graduate Students

A Branching Fuzzy-Logic Classifier for Building Optimization

By

Matthew A. Lehar

Submitted to the Department of Mechanical Engineering
on May 19, 2005 in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy in
Mechanical Engineering

ABSTRACT

We present an input-output model that learns to emulate a complex building simulation of high dimensionality. Many multi-dimensional systems are dominated by the behavior of a small number of inputs over a limited range of input variation. Some also exhibit a tendency to respond relatively strongly to certain inputs over small ranges, and to other inputs over very large ranges of input variation. A branching linear discriminant can be used to isolate regions of local linearity in the input space, while also capturing the effects of scale. The quality of the classification may be improved by using a fuzzy preference relation to classify input configurations that are not well handled by the linear discriminant.

Thesis Supervisor: Leon Glicksman
Title: Professor of Architecture and Mechanical Engineering

Acknowledgments

I would like to thank the Permasteelisa Group for their sponsorship of the MIT Design Advisor generally, and my graduate studies in particular. I am also extremely grateful for the generous support of the Cambridge-MIT Institute and the Martin Family Society of Graduate Fellowships in Sustainability.

Jim Gouldstone's programming expertise has played an important role in the success of this project. The web interface would not have been possible without his guidance.

I am grateful to Les Norford and Dave Wallace for agreeing to join my thesis committee and for their help and suggestions.

Thank-you so much, Leon, for providing the inspiration for this project and for all your help in making it happen. The weekly meetings were very valuable to me.

Lots of love to Mum and Dad.

Cambridge, 5/19/05

TABLE OF CONTENTS

INTRODUCTION

Background: The MIT Design Advisor

The web-based design tool known as the "MIT Design Advisor"[1] has been in development in the Building Technology Lab since 1999. Online and freely available to the Internet community since 2001, the tool has been substantially expanded from its original role as a heat-transfer model for building facades. Presently, users can access daylighting and comfort visualizations, check their buildings against various code restrictions, simulate both mechanical and natural modes of ventilation, and receive total cost estimates for energy consumption over the building lifetime.

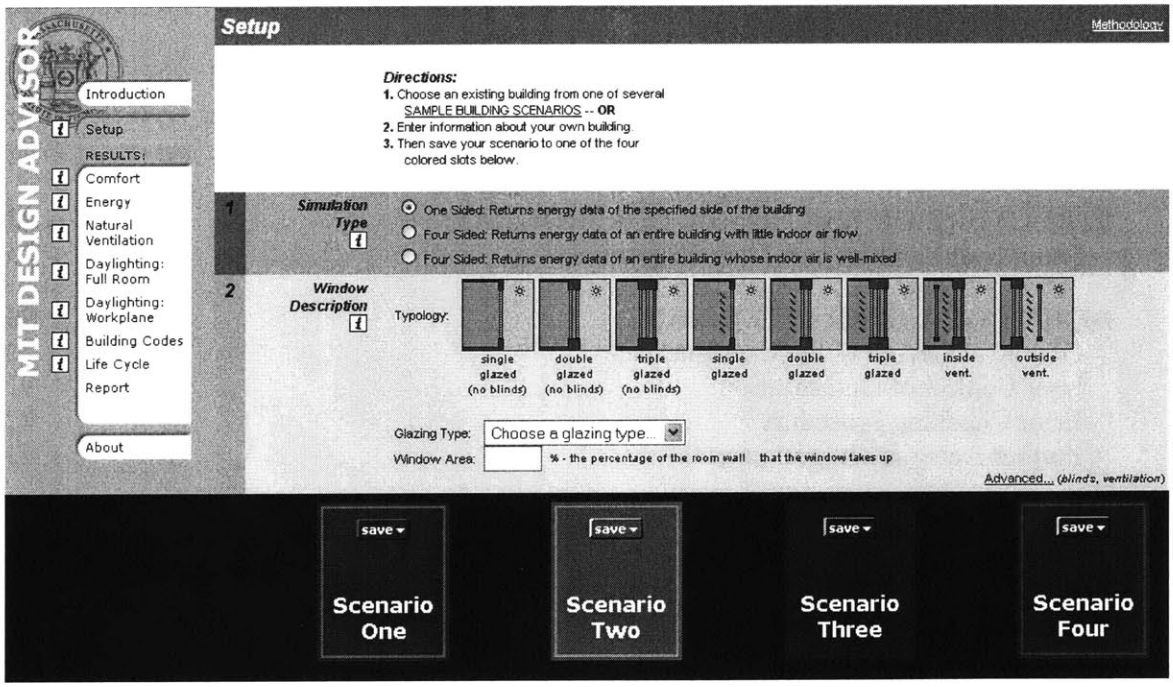


Fig. 1. The Design Advisor input panel.

The tool is designed to be used in the early stages of a building project, at a point when architects are still entertaining a range of different ideas about siting, general plan outline, window coverage, and other basic design issues. Accordingly, the design specifications that the user is asked to enter (Fig. 1), and which are then used in simulation programs to produce estimates of monthly energy usage by the building, are general in nature (although users wishing to use more detailed constraints can open submenus where they may be entered).

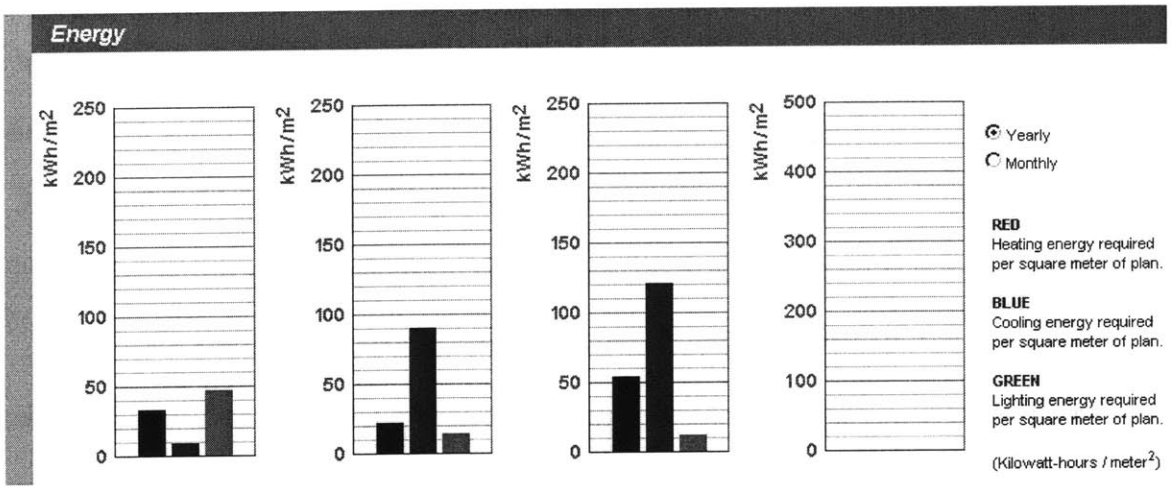


Fig. 2. View of the Design Advisor website showing energy consumptions calculated for 3 buildings.

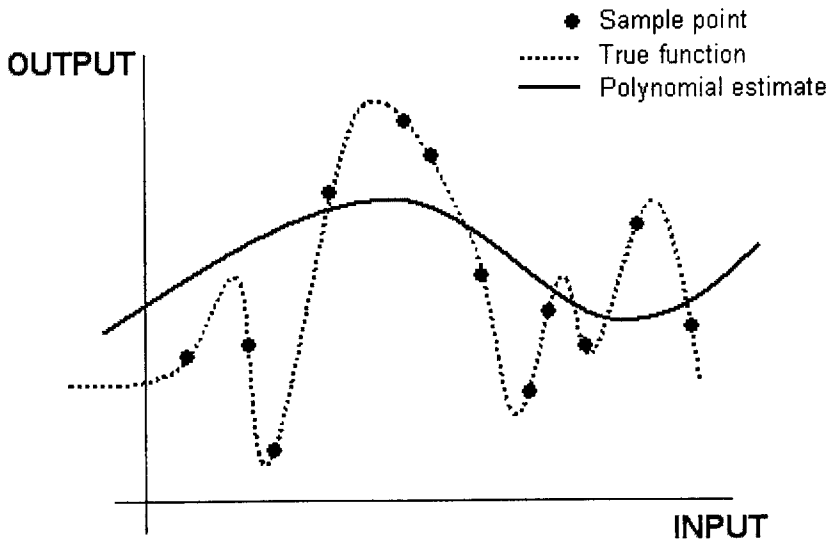
The benefit of using a tool like the Design Advisor derives from the ability to compare many different configurations of building on the basis of the amount of energy consumed. The present version of the software includes the facility to save four different configurations at once, so that their heating, cooling and electrical lighting consumption may be compared graphically on the screen (Fig. 2). While this facility was considered useful in determining how the building performance is affected when individual

parameters are varied, it has still been difficult for users to discover how the more than 40 different inputs defining the building can be conjointly varied in such a way as to bring energy savings. This would require an automatic feature for optimizing multiple parameters simultaneously in the building model. This thesis presents a possible algorithm for such an optimization tool. Finding a robust methodology for incrementally choosing and testing increasingly efficient combinations of input parameters is a problem that has substantially been solved in other arenas. The principal part of the work presented here concerns the problem of developing an allegory-function to substitute for the physical building simulations used by the Design Advisor. Those models, which have been verified at an accuracy of more than 85% by third-party tools, rely on physical descriptions of the conductive, convective, and radiative behavior of building facades and interior cavities. Although they rely on substantial simplifications of the building model, they still require about 5 seconds of processing time to produce energy estimates for each new building configuration, which is impractical for a large-scale optimization incorporating hundreds of thousands of trials. Instead of running these models directly in the optimization process, we have constructed a non-physical input-output model that can be trained to reproduce the behavior of the physical models approximately, and that runs in a tiny fraction of the time.

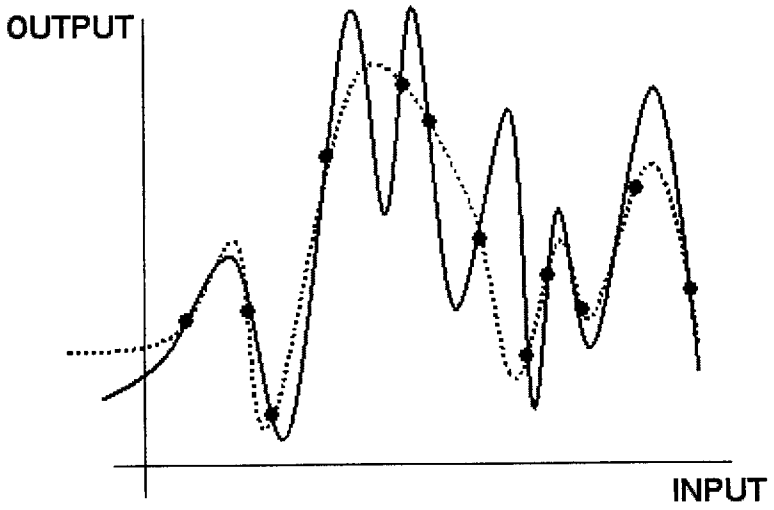
Strategies for Pattern Recognition

At the heart of the problem of creating an input-output model is the task of pattern recognition. Given a set of sample input values and the outputs they produce, the model must reconstruct the function that relates them. This is often accomplished with a

regression, in which a general form is assumed for the function and its coefficients are adjusted so that expression predicts the sample output values with minimum error. To match the function to the sample data so that it will correctly predict any new output, the equation should be of the same *order* as the data. Pattern recognition techniques traditionally treat the modeled phenomena as though they can be classified as fundamentally linear, quadratic, cubic, or of some higher order. If a low-order model is used to estimate data in a high-order problem, the presumed function will not estimate the sample outputs with great accuracy (Fig. 3a). If, on the other hand, the problem is of low-order and a high-order model has been applied to it, the result is a function that can be tuned to reproduce the sample outputs exactly, but that also imposes convolutions unsupported by the data that prevent the correct estimation of future output values (Fig. 3b).



a.



b.

Fig. 3. Estimating a function using sample points. In a.), the estimation is of a low order, and the curve does not model the sample points with much accuracy. In b.), the order is increased and all sample points are exactly intersected, yet the estimation produces artifacts not present in the original function.

The effectiveness of polynomial-fitting procedures, along with other basis-function techniques such as neural networks, is limited by the difficulty of extracting general rules

from the training data. To be able to assign the correct output to each fresh input value encountered after the completion of the training process, the algorithm must have some way of deciding to what extent the sample data represents the larger reality of the phenomenon that created it. The special information that each point provides is to some extent merely the result of its accidental selection for the training set from among many possible neighboring points in the input space.

The GMDH (Group Method of Data Handling) approach of Ivakhnenko[2] uses an estimation routine that attempts to resolve this problem by adaptively matching the order of the model to that of the underlying phenomenon represented by the training data, and work has been carried out (see Farlow[3]) to expand the technique to a larger class of non-polynomial basis functions as well. But in addition to the problem of order-matching, the basic assumption that the phenomenon to be modeled has the form of a combination of basis functions may itself be problematic. Since any function can be arbitrarily well approximated by a Taylor series (or Fourier series, etc.) about a particular point, the presumption of this underlying polynomial behavior can be shown to be valid locally, but it may not capture the complete variation in a problem over a range of input values.

A simple approach we have used to avoid this difficulty in our building performance application is to organize training data into histograms (see the chapter on *The Membrane Algorithm*). In a problem with one input dimension, this approach reduces to creating a lookup table in which output values are designated for each of several intervals along the

input axis. The output values in the lookup table are averages taken over all sample data that fall within a particular interval of input values (Fig. 4). Such a table can reproduce with arbitrary accuracy the function underlying the sample data (Fig. 5), provided that a large enough supply of data is used and the "bins" into which neighboring data are grouped cover a suitably narrow range of input values.

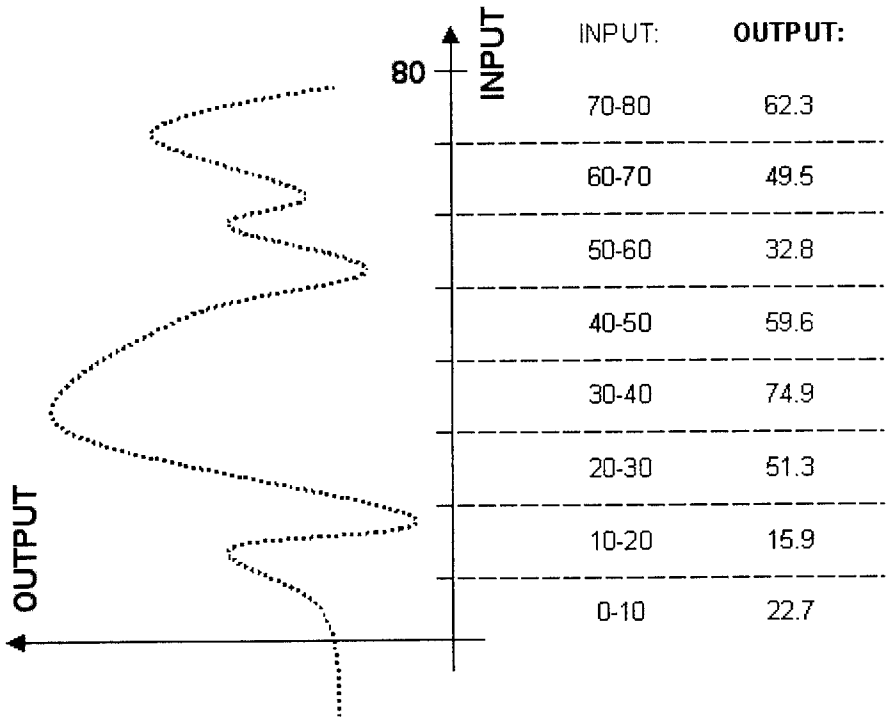


Fig. 4. Construction of a lookup table.

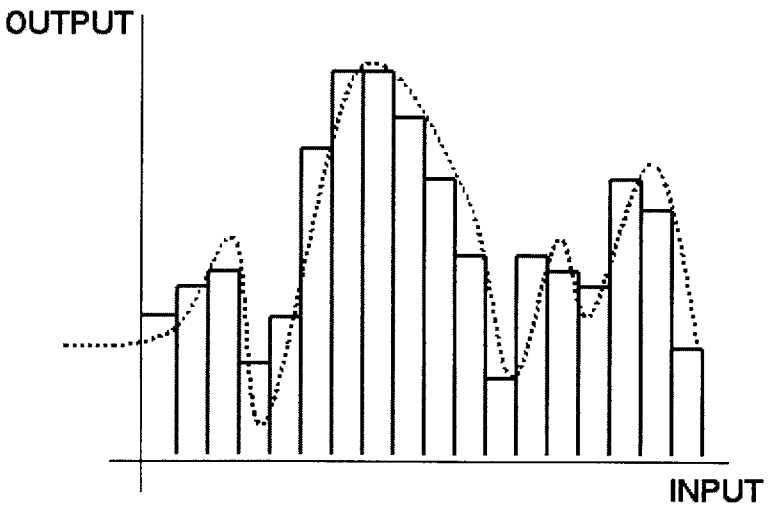


Fig. 5. Representation of a function in one dimension using a histogram.

The obvious disadvantage of this method over basis-function methods is the difficulty of extending it to a problem with multiple input dimensions. The accuracy of the histogram representation of a function depends on the density of points along a given input axis. In our single-input example, a sample size of 100 points would provide a maximum resolution of 100 pixels with which to render of the output-generating function using the lookup table. In the case of a function in two inputs, the same sample size would characterize the 3-dimensional plot of the function surface with a resolution of only $(100)^{1/2} = 10$ pixels in each dimension. As we will explain in the chapter, *The Membrane Algorithm*, the problem of representing data in multiple dimensions with histograms can be solved by using a series of 1-dimensional histograms, rather than a single, multi-dimensional one.

LINEAR METHODS: EMULATING THE BEHAVIOR OF PHYSICAL MODELS

Introduction

The problem of optimizing our existing physical building model is straightforward. We select an optimization algorithm appropriate to the large number of varying inputs, such as Simulated Annealing, and the output from each run of the model provides an objective function for the Annealing algorithm. Each run of the model takes approximately 10 seconds and, since many thousands of cycles are required for the Annealing algorithm to stabilize, an entire optimization should take several hours. Unfortunately, this is too long to wait in the context of our particular application.

The principal use of the optimizer will be as a feature of the Design Advisor, allowing users to find a lowest-energy building configuration. The user will be able to fix any number of input values as exogenous to the optimization problem, so each optimized building configuration will be unique to the particular choices made by each user. Consequently, each new optimization must be performed in real-time as the user requests it. Since running an optimization by using our original building simulation to calculate values of the objective function would take hours, this cannot be the basis of a real-time optimizer. Instead, we must develop a substitute for the original model to run in a fraction of the time.

To create an emulator of the physical building model, we propose a rule-learning algorithm that can be trained using results generated by the physical model to imitate the model's behavior. We will refer to the total annual energy requirement that the physical model calculates for the building as the *output*. The physical model transforms a set of input values – the *input vector* – into a single value of the objective function – the sum of the heating, cooling, and lighting energy that a building defined by those input parameters will require for one year of operation. In our application (buildings), properties such as window reflectivity or room depth are the input parameters that make up an input vector. When enough input vectors have been evaluated by the physical model, we can begin to make crude approximations of the values of the objective function that correspond to new and untried input vectors. The role of the emulator is to estimate the objective function for any input vector by finding rules that link the already-evaluated input vectors with the values of the objective function they have been found to correspond to. The emulator uses only the results produced by the physical model to construct its rules, without reference to the equations of heat transfer used in the original model. It should be able to operate many orders of magnitude faster than a detailed physical simulation.

In the interest of speed, we would like to use the simplest possible algorithm to approximate the behavior of the building model. A lookup table could be used if not for the large number of input variables involved in the simulation. Presently, over 40 are used in the Design Advisor interface, implying a 40-dimensional input space. To populate a lookup table for such a space that could provide even two unique examples for

a trend along any given axis of movement within the space would require 2^{40}, or about a trillion, training data. Using a Pentium III processor at 1 GHz, it would take 322,000 years to accumulate so many results by running the original model, not to speak of the problem of storing and accessing such a library. Clearly, such a scheme would be impractical due to the large number of input parameters.

Linear Regression

One naïve scheme for estimating the output value that results from a large suite of inputs is to use a linear regression. The input values are each multiplied by some constant coefficient that best captures the sensitivity of the output to that variable, and the scaled input values are then added together to make a predicted output value:

$$Output = a_1 X_1 + a_2 X_2 + a_3 X_3 ... + a_n X_n \qquad (1)$$

In set-theoretic terms, the role of a linear regression is to provide an ordering rule for a set of training data. If the linear prediction of an output value is given by the dot product $W \cdot X = Output$, where W is a vector of weighting coefficients, then W is a unique way to order the training data input vectors X_i such that their predicted outputs are monotonically increasing. For a set of n data,

$$W \cdot X_{i+1} \geq W \cdot X_i, \forall i < n \qquad (2)$$

To imitate the dynamics in the building model effectively, we must use a scheme that can accommodate nonlinearity in the training data. Because nonlinear functions are not monotonic in general, they will not be well approximated by a pure linear regression.

The values of the objective function F_i from a nonlinear training set provide a measure of the inaccuracy of the regression in the magnitude of a set **D**, where

$$\mathbf{D} = \left\{X, W \in R^L : W \cdot X_j \geq W \cdot X_i, F(X_j) < F(X_i), \forall i < j \leq n\right\} \qquad (3)$$

This set of points **D**, which we call the set of *misordered* points, is shown graphically in **Fig. 6** for a two-dimensional input space. The nonlinearity of the function in **Fig. 6** is related to the breadth of the region of overlap after a linear classification.

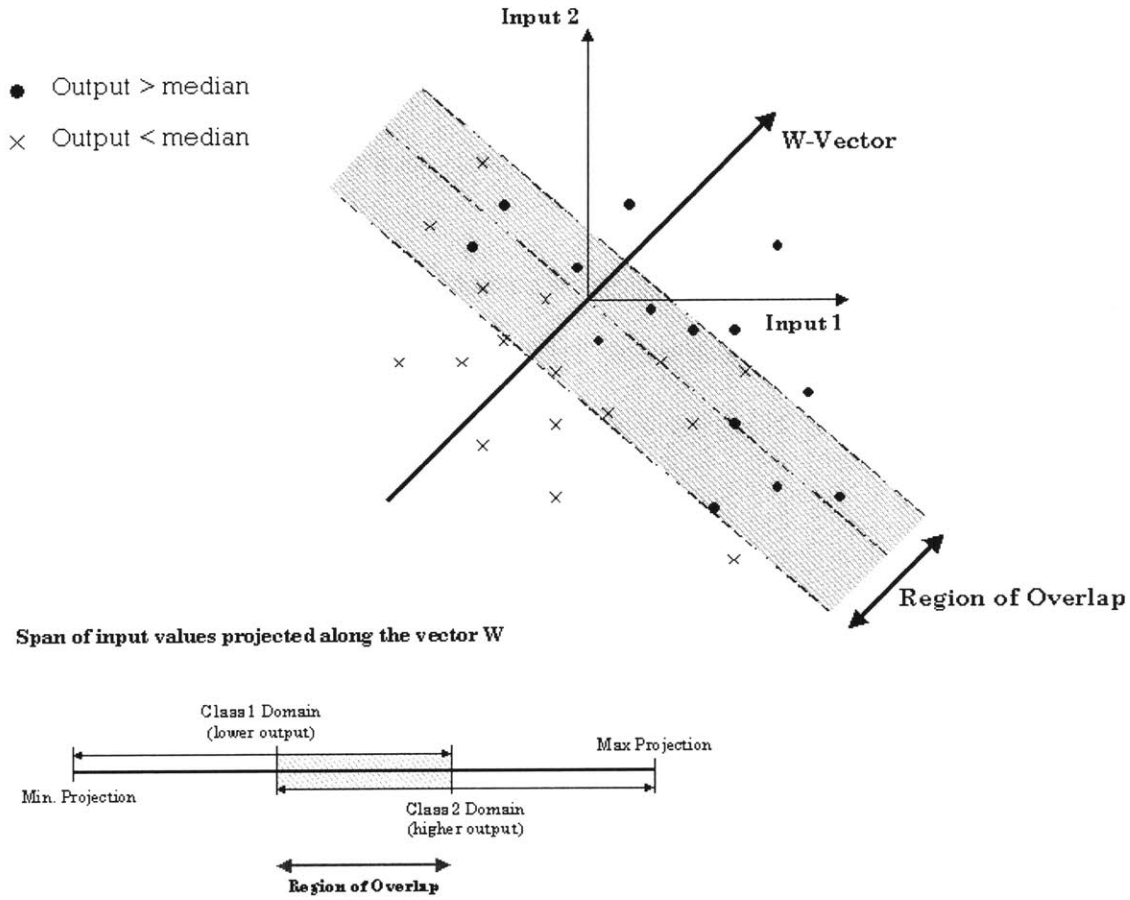


Fig. 6. A function of two inputs. The vector W is oriented in the 2-dimensional input space so as to order the data in accordance with the output values. In the Region of Overlap the data are out of order in respect to their output values, defining a set D.

The particular vector **W** used in the regression can be said to be optimal on a set $\{X_i, i < n\}$ if the magnitude $\|\mathbf{D}\|$ is minimized by that choice of **W**. Although poorly represented by any linear regression, the set of training data that follows a nonlinear objective function will have an associated minimum set $\overline{\mathbf{D}}$ that is nonempty. We can measure the degree to which a subset of the training data submits to a linear ordering

using the average predictive accuracy $\frac{n - \|\overline{\mathbf{D}}\|}{n}$, the ratio of monotonically ordered points to the total number of training points n that are used in the regression. This measure of the accuracy of a regression will depend on two factors:

1) The particular coordinates in the input space where the regression is centered, and

2) The subset of n training data used to determine $\overline{\mathbf{D}}$ in the vicinity of those coordinates.

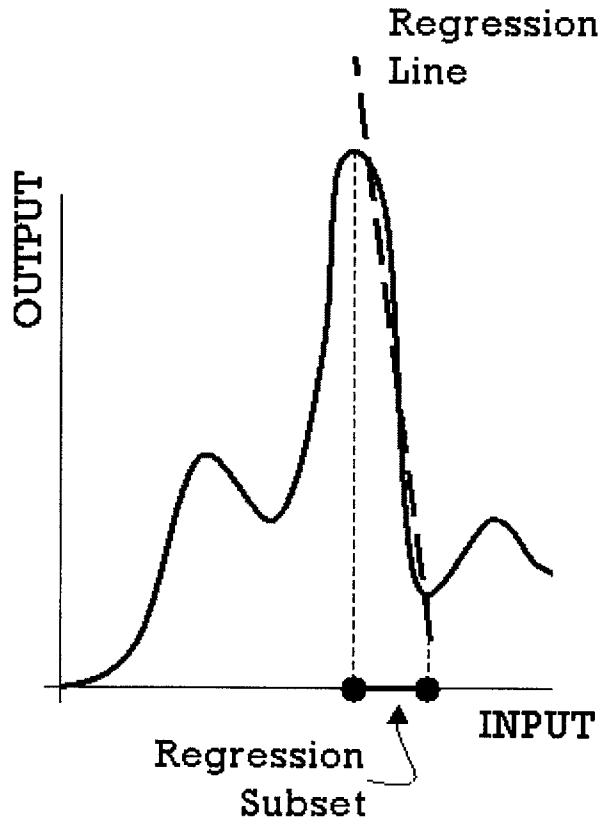


Fig. 7. Applying a regression to a limited subset of the training data allows greater accuracy over a limited part of the input space.

The regression approximates F over the range of input vectors included in a subset of the training data. The slope of the regression line in Fig. 7 represents the vector of coefficients **W**, and is chosen so as to minimize the deviation of the line from a limited set of neighboring data points. If we assume the objective function F is continuous, for a given vector of weighting coefficients $\mathbf{W}_0$ we can show that the accuracy of prediction $\frac{n-\|\mathbf{D}\|}{n}$ approaches 1 in the limit as we reduce the interval defining the region of the input space from which training data may be used in the regression:

First, we define a set **B** representing any monotonic interval in a continuous nonlinear function: if

- **S** is the initial set of training data $X \in R^L : X_1, X_2, ... X_n$,
- **D** is the set of misordered points resulting from the ordering of **S** by $\mathbf{W}_0$, and
- the set $\mathbf{S} - \mathbf{D}$ is nonempty,

there exists a subset of vectors $\mathbf{B}_r \subset \mathbf{S}$ containing a vector X_0 such that:

$$\mathbf{B}_r = \left\{ X : \mathbf{W}_0 \cdot X_j \geq \mathbf{W}_0 \cdot X_i \Rightarrow F(X_j) \geq F(X_i), \forall X \subset \mathbf{S} : \|(X - X_0)\| < r \right\} \quad (4)$$

where r is a scalar. Such a subset is illustrated for the single-input case in Fig. 7, over which the behavior of the function is monotonic. For a single X_0 within **S**, we can define a distance p between the vector X_0 and the nearest point for which $\frac{\partial F}{\partial X^a} = 0$, where X^a is any scalar component of X. Then, for any X_0 defined on F,

$$\exists \varepsilon \in R^L, X_0 + \varepsilon > X_0 \Rightarrow F(X_0 + \varepsilon) > F(X_0) : \varepsilon < p \quad (5)$$

This is to say that the function F is monotonic within a range p of X_0. From (4), we see that within a training set **S** there always exists some set of coordinates **B** for which the outputs vary monotonically with the ordering provided by $\mathbf{W}_0$. (5) allows us to determine the bounds on the largest monotonic set $\mathbf{B_p}$ that contains X_0; since a continuous function is by definition monotonic between points where partial derivatives of the function equal 0, we will end up with a monotonic set simply by narrowing the radius of inclusion r until $r < p$. This means that the accuracy of a linear regression will generally improve as data more remote from a chosen reference point X_0 are removed from the regression calculation. However, although we are able to evaluate a given input more and more accurately in the context of the other points that remain in the regression, we lose the ability to evaluate it relative to the entire training set. The quality of the local ordering of points in the subset improves while the global ordering disappears. There is no "ideal" regression to characterize a particular region of the input space because narrowing the scope of input points in the regression to increase the linearity also makes the regression less comprehensive.

Method of Subdivisions

For any given point in the input space, and given the ability to determine the weighting vector W that minimizes $\frac{n - \|\overline{\mathbf{D}}\|}{n}$ for a prescribed set of training data in the neighborhood of that point, the problem that remains is where to draw the boundary between data that will and will not be included in the regression. Since we cannot choose a subset to maximize linear ordering within the set at the same time that we maximize the coverage

of the regression, we propose a method of multiple regressions that systematically divides up the input space. This method is unconventional in the sense that the space is repeatedly divided in half to create a branching hierarchy of divisions. Most other methods (e.g. Savicky[4] and Hamamura[5]), rather than a segmenting, rather than branching, approach is usually taken when using binary classifiers to represent multiple output categories.

We begin by performing a regression on the entire training set. This will identify linearity at the largest possible scale, but rather than finding an approximation to the function using this very coarse regression, we would like instead to use it to put limits on the range of possible output values. A second regression based on only those training data that lie within the indicated range could then be performed, limiting the range still further. After many such subdivisions of the space we would arrive at the point where further regressions could not be justified due to the small number of chaotic distribution of the remaining samples. At each decision point, where a set of training data is divided into two distinct groups according to output, we create a linear regression that is special to that decision point. What results is a "tree" of linear regressions calculated on the basis of various subsets of the training data. This tree can then be used to classify new input points by passing the component values down the tree to regressions that deal with ever more specific regions of the input space. The reason for using this multi-stage approach is that it exploits linearities in the data at a range of different scales, rather than just at a small scale where a single regression is not comprehensive, or just at a large one where it is not accurate. At the end of a cascade of binary decisions, we can predict the

probable range of outputs that a new input point could evaluate to by noting the limits on the outputs of training data that were given the same classification – the same "slot" at the bottom of the tree.

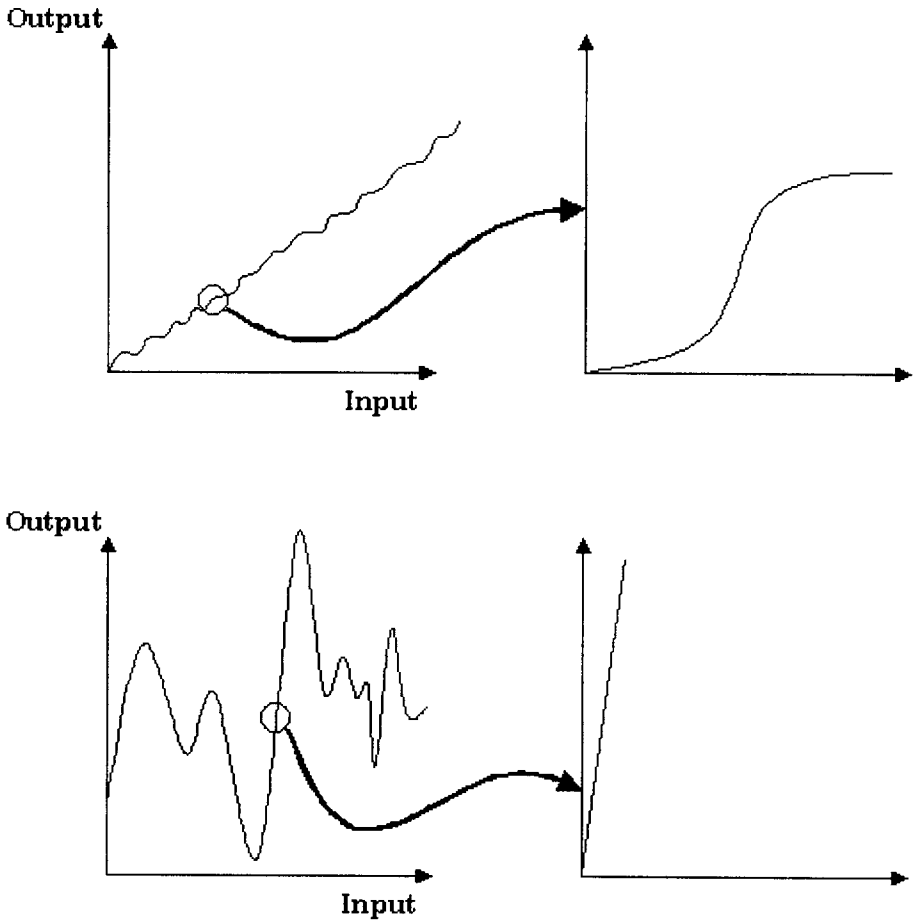


Fig. 8. A broadly linear function (top) can display extreme nonlinearities at a finer scale. Conversely, a highly nonlinear function (bottom) can approximate linear behavior when taken over a small range.

Changes in the value of the function in respect to an input variable can resemble a linear relationship over a large range, but become highly nonlinear at a smaller scale. The

reverse can also be true (see Fig. 8). Using a scheme of successive subdivisions of the input space, we should be able to capture these linearities at many different scales.

The particular coordinates on which a regression should be centered at each stage of this process are not known a priori. However, the multi-stage approach provides a way around this problem: while it does not construct a regression for every possible grouping of training data points it effectively subdivides the input space into successively smaller contiguous data sets. Any single input vector will belong in one of these (arbitrarily small) subsets, so that no matter which input we choose from the training data, it will be associated with some regression constructed from its immediate neighbors. This guarantees that the regression will be locally accurate. At the same time, we can learn which of these local regressions we should use to evaluate an input by consulting the more general regressions in the next layer up in the branching structure. In Fig. 9, two regression lines have been drawn: one follows the general trend of the function taking in the full range of inputs. The other is based on a highly restricted range of inputs, and only functions to distinguish output values within that narrow range. Using only the restricted regression as an ordering for the entire data set would clearly be disastrous, but it provides great accuracy if the crude regression is first used to identify the neighborhood of values where the restricted one will be effective. From (5), we see that we eventually arrive at a perfect linear ordering of a subset of data simply by narrowing the range of that subset. This means that there will always be an accuracy benefit from narrowing the scope of the subset on which the regression is performed. By using a series of

successively narrower regressions, we can design a predictive algorithm that is both comprehensive and minutely accurate.

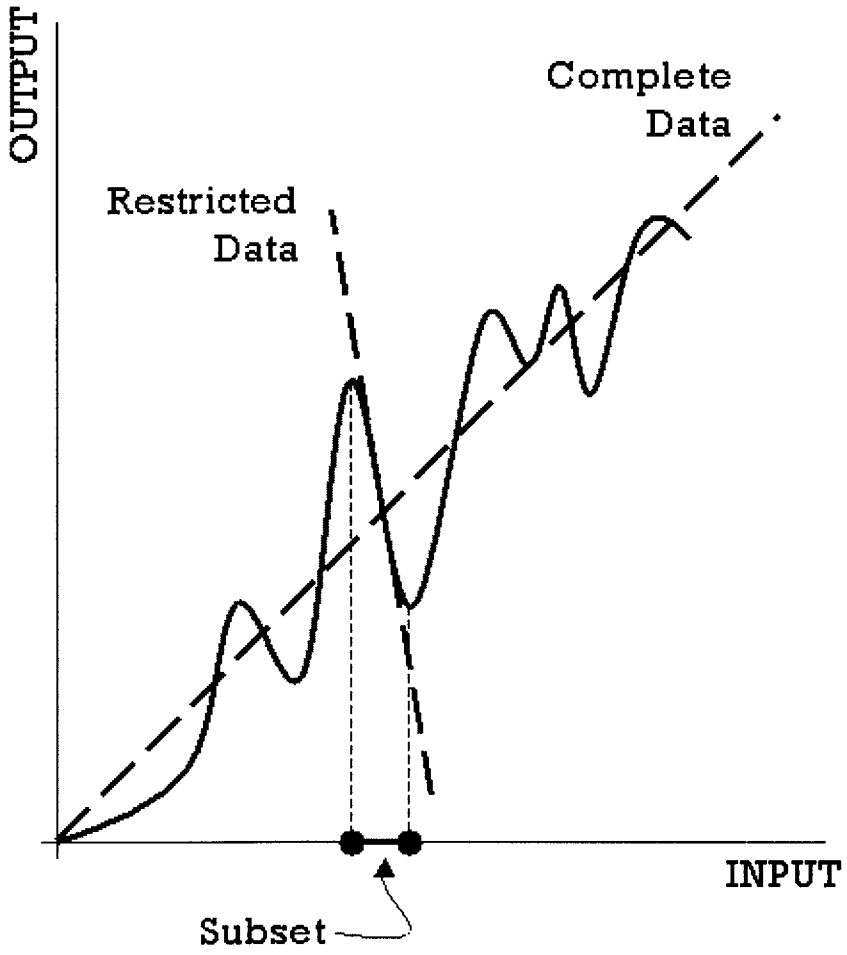


Fig. 9. Two regression lines: one is based on the complete data set and the other on a restricted monotonic subset.

Binary Classification

The process of determining which terminal subgroup a given input belongs in can be compared to skiing down a mountainside on which the snow trails repeatedly divide in

two. At the first branching point, the skier makes a very general choice, such as whether to ski on the shaded or the sunlit side of the hill. As she proceeds to commit to one trail or another at each fork, the quality of the decisions becomes more specific: a decision as to which chairlift area she would prefer to end up in, and then whether to take the smooth run or the one with moguls. Finally, she decides which of two available queues to turn into for the chairlift ride back up the mountain. Her ultimate choice of a queue makes a very specific distinction, but it can only be arrived at by a history of decisions at a more general scale.

Like the skier who chooses a more and more specific route as she progresses down the mountain, our multi-stage procedure uses a series of linear regressions to make increasingly specific predictions of the value of an objective function. At each level of the decision process, we preserve as much generality as possible, so as not to interfere with the better resolving power of subsequent steps. We ensure this outcome by using the regression functions to impose the weakest possible restriction at each level. The vector **w** serves as a two-group classifier, projecting input vectors onto a scalar value that is less than 0 if the input is selected for one group, and greater than 0 if it is selected for the other (Fig. 10).

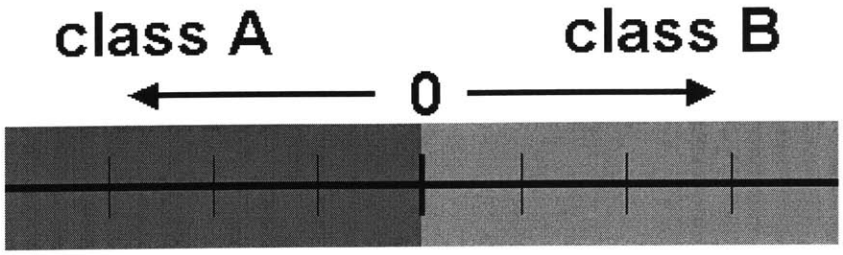


Fig. 10

The two-group (binary) classification tree is the most conservative possible scheme for sorting input vectors because no distinction is made at any level of the tree that would also be possible at the next level down. A decision about which general group a vector belongs in does not bias or influence the next decision about which part of the selected group should be chosen. The only new information added at each selection stage is an identification of the next-most-specific subgroup to which an input will be assigned. We could of course obey this same principle while choosing from among 3,5, or 100 subgroups at each level, but a 2-group scheme should in general be preferred because it allows the subgroups to remain larger and the classifications less restrictive down to a lower level of the tree than any multiple-classifier scheme (Fig. 11).

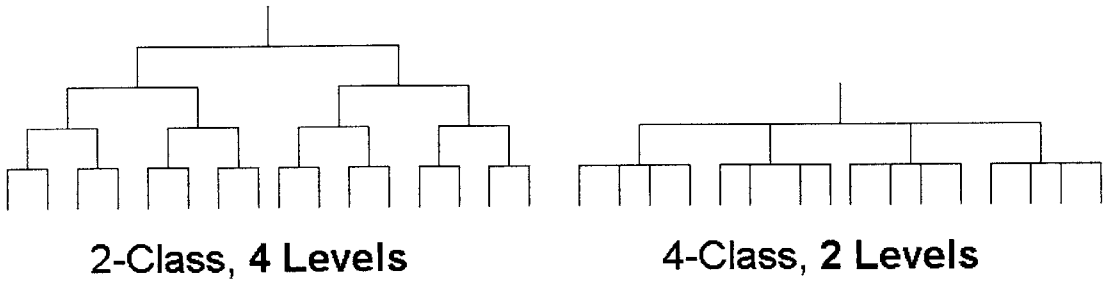


Fig. 11

The more layers of decision opportunities we have, the more individual regressions we bring to bear on the classification and the more informed the ultimate prediction will be.

The Fisher Discriminant

The Fisher Discriminant is a regression technique that is often used as a criterion for separating a group of samples into two categories based on their known parameters. If some of the samples are arbitrarily designated as "Type A," and the others as "Type B," a linear model can be built up to correlate the parameters of each sample optimally with its specified type. For instance, if we were to classify cars on the highway as either "fast" or "slow," we would do so according to whether their speed is above or below a chosen threshold value. Then, the values of input parameters like engine size, body type, color, and weight could be correlated with the side of the threshold on which the car is found. Once the correlation has been established using this training data, it can be used to predict whether a new car, whose type has not been observed, will likely be fast or slow based only on the knowledge of its size, body, color, and weight.

In our experiment, the cars on the highway are replaced by the training data. These consist of a set of vectors (of parameters like size, color, and weight) for which values of the objective function (e.g. speed) are known in advance. Taking any subset of this data, we can choose an objective function value as a marker value that divides the subset into two parts: one containing the inputs for which the objective is below the marker value, and the other for which the objective exceeds the marker value. To divide a set of training vectors evenly in half, we would choose the median of the objective function values as the marker. If we create a linear regression from the training data, finding the linear combination of the components of each input vector that best approximates the objective function, we can use that regression to predict an objective value for any new input vector, and place it in one group or the other depending as the objective value is above or below the threshold.

The Fisher Discriminant is a particular kind of regression that is optimized for the case in which we would like to separate data into two groups. Like any other regression, it projects the multi-dimensional parameter space onto a 1-D vector traversing that space. But instead of adjusting that vector (hereafter called **w**) so as to minimize the sum of the squares of the differences between each vector's true objective value and its predicted value, the Fisher Discriminant orients **w** so as to discriminate optimally between the two classes. The length of the projection of each measurement along **w** can then be used to classify it as either of group "A" or "B" (Fig. 12).

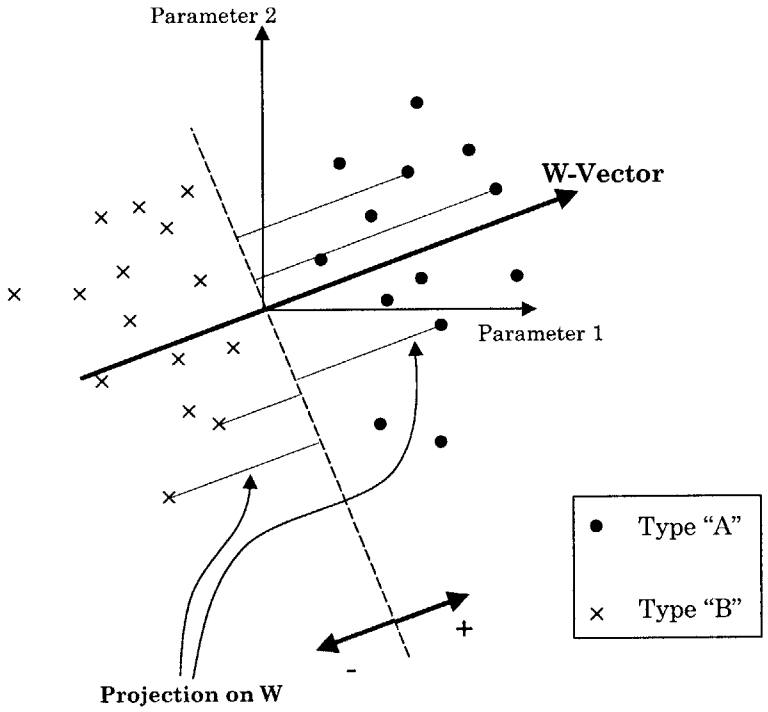


Fig. 12. A line clearly divides the data into two classes in this example, in which the overall mean has been normalized to the origin of the coordinate system. Positive projections on W can be classified as "A," negative as "B."

There are many possible ways to "optimize" the orientation of **w** to discriminate between two classes of data. The Fisher Discriminant uses two principal measures: the projection of the *mean difference vector* on **w**, and the projection onto **w** of the *spread* of the data in each of the two classes. The mean difference vector (a *between-class* metric) is the axis along which the two groups of data are most distinct from each other; the spread of the data (a *within*-class metric) shows how the positions of data points in each of the two groups become more concentrated or more diffuse when measured along a given axis. Optimizing one of these measurements to the exclusion of the other can result in a poor classification in certain circumstances. As shown in Fig. 13 and Fig. 14, either metric can contain the information that is essential to the group-separation problem. The Fisher

Discriminant finds a compromise between the separation critera by optimizing both simultaneously.

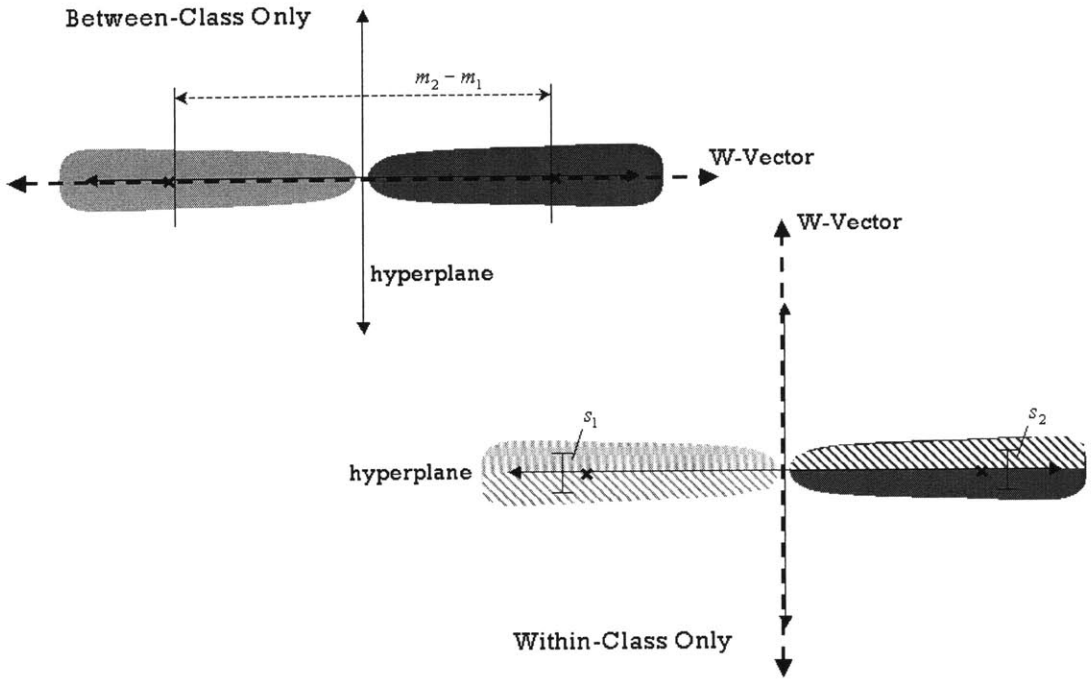


Fig. 13. Insufficiency of the within-class criterion alone. The input vectors are 2-dimensional, and the the regions of points representing groups A and B are shown in two shades of gray. The orientation of *W* (dashed line) that produces an optimal between-class separation is horizontal, allowing the data to be cleanly divided according to its projections on *W*. If the within-class criterion alone is used, *W* is vertical when the projections of the spread of data onto *W* in each group are minimized, yet these projections are insufficient to distinguish between the groups.

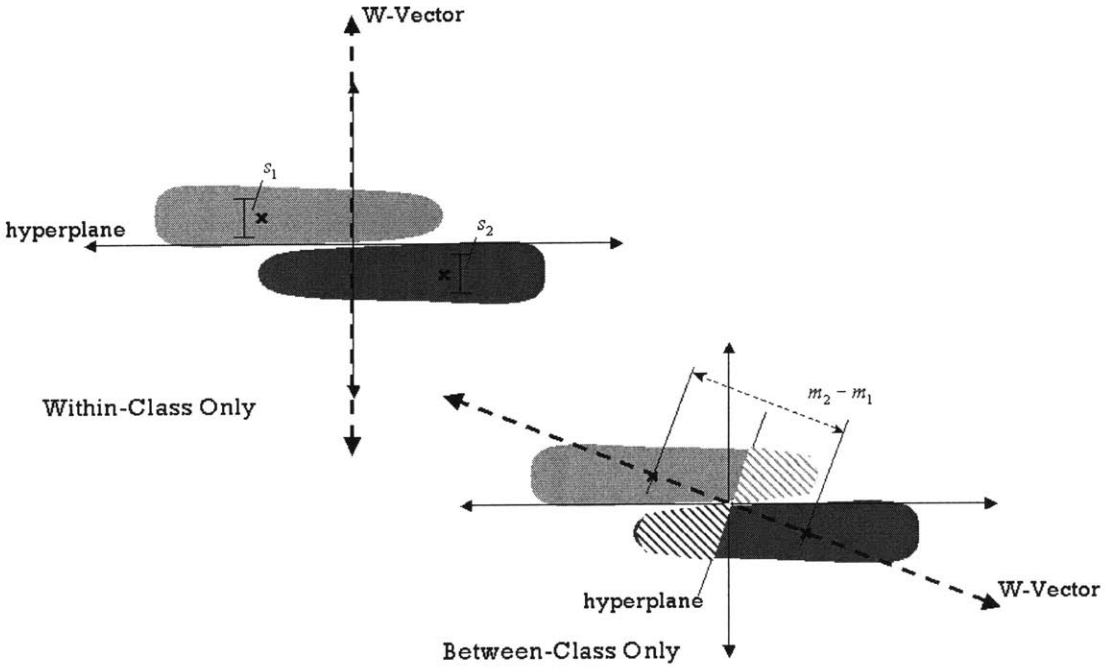


Fig. 14. Insufficiency of the between-class criterion alone. In this example, the within-class spread provides the essential information for separating the two groups of data.

Between-Class Separation

The mean difference vector is oriented along the line connecting the point at the mean coordinate values of the training inputs in one group (or *class*), and the mean point of those in the other group. In Fig. 13 and Fig. 14 this vector is shown for the two-parameter case. In the analysis that follows, **m** represents the vector (of 2 or more dimensions) of mean values for each of the input parameters, taken over the set of data points in group A or B. We wish to orient **w** parallel to the mean difference vector so that the projection of the data points onto **w** shows the greatest possible separation between the groups. We express the degree of alignment of the vector **w** with the mean difference vector $(\mathbf{m}_B - \mathbf{m}_A)$ as the dot product. Since the quantity $\mathbf{w} \cdot (\mathbf{m}_B - \mathbf{m}_A)$ will be maximized when $\mathbf{w} \parallel (\mathbf{m}_B - \mathbf{m}_A)$, we can define a *between-class covariance matrix*

$\mathbf{S}_B = (\mathbf{m}_B - \mathbf{m}_A)(\mathbf{m}_B - \mathbf{m}_A)^T$ such that the scalar expression $\mathbf{w}^T\mathbf{S}_B\mathbf{w}$, equivalent to the square of $\mathbf{w} \cdot (\mathbf{m}_B - \mathbf{m}_A)$, is also maximized when $\mathbf{w} \| (\mathbf{m}_B - \mathbf{m}_A)$. The parallel alignment of $\mathbf{w}$ and $(\mathbf{m}_B - \mathbf{m}_A)$ maximizes the projected distance between the clusters of each class of data.

Within-Class Separation

The other expression optimized in the Fisher Discriminant preferentially indicates orientations of $\mathbf{w}$ that emphasize the density of clustering within each class, as viewed through the projection onto $\mathbf{w}$. Maximizing the clustering density reduces ambiguity due to two class regions overlapping in the input space. The clustering density is inversely related to the quantity $\mathbf{w}^T\mathbf{S}_W\mathbf{w}$, the *within-class* covariance matrix. $\mathbf{S}_W$ is defined:

$$\mathbf{S}_W = \sum_{n \in C_A} (\mathbf{x}^n - \mathbf{m}_A)(\mathbf{x}^n - \mathbf{m}_A)^T + \sum_{n \in C_B} (\mathbf{x}^n - \mathbf{m}_B)(\mathbf{x}^n - \mathbf{m}_B)^T \qquad (6)$$

where $\mathbf{x}$ is a vector representing a sample input. The diagonals of $\mathbf{S}_W$ are the variances of each of the input parameters within class "A," added to the variances from class "B." The off-diagonal elements of $\mathbf{S}_W$ are the covariances – the tendency of one component of the input vector to stray from the mean at the same time that another component strays from its own mean. If the components of the sample input vectors are chosen independently, the covariances should converge to 0.

The Fisher Discriminant has the form

$$J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}} \quad (7)$$

which is maximized when the between-class covariance is maximized, and the within-class covariance minimized. To do so, we differentiate (7) with respect to **w** and set

$$\frac{dJ(\mathbf{w})}{d\mathbf{w}} = 0 \quad (8)$$

By the quotient rule for matrices, $\frac{d}{d\mathbf{x}}\left(\frac{f(\mathbf{x})}{g(\mathbf{x})}\right) = \frac{f'(\mathbf{x})g(\mathbf{x}) - f(\mathbf{x})g'(\mathbf{x})}{g^2(\mathbf{x})}$. Applying this rule to (8), we have

$$\frac{dJ(\mathbf{w})}{d\mathbf{w}} = \frac{d(\mathbf{w}^T \mathbf{S}_B \mathbf{w})}{d\mathbf{w}} \mathbf{w}^T \mathbf{S}_W \mathbf{w} - \frac{d(\mathbf{w}^T \mathbf{S}_W \mathbf{w})}{d\mathbf{w}} \mathbf{w}^T \mathbf{S}_B \mathbf{w} = 0 \quad (9)$$

By the rules of matrix differentiation, $\frac{d(\mathbf{x}^T \mathbf{C} \mathbf{x})}{d\mathbf{x}} = 2\mathbf{C}\mathbf{x}$, so (9) becomes

$$\left(\mathbf{w}^T \mathbf{S}_B \mathbf{w}\right) \mathbf{S}_w \mathbf{w} = \left(\mathbf{w}^T \mathbf{S}_W \mathbf{w}\right) \mathbf{S}_B \mathbf{w} \quad (10)$$

Because we are only interested in the direction of **w**, and not the magnitude, we may drop the scalar factors $\mathbf{w}^T \mathbf{S}_B \mathbf{w}$ and $\mathbf{w}^T \mathbf{S}_W \mathbf{w}$. As we noted before, $\mathbf{S}_B \mathbf{w}$ has the same direction as $(\mathbf{m}_A - \mathbf{m}_B)$, so we may write (10) as a proportional relationship,

$$\mathbf{w} \propto \mathbf{S}_W^{-1} (\mathbf{m}_B - \mathbf{m}_A) \quad (11)$$

which we use as a formula for **w**.

Simplification for a One-Dimensional Space

The Fisher criterion given by (7) can be expressed in a more intuitive form if we consider the case of only one input parameter for the data under consideration. The mean difference vector $(\mathbf{m}_B - \mathbf{m}_A)$ becomes the simple difference of group means for the

single parameter, (m_B-m_A), and the matrix $\mathbf{S}_B$ becomes a scalar with the value (m_B-m_A)^2. The matrix $\mathbf{S}_W$ can be seen to reduce to the sum of the within-class covariances of the two classes, where the within-class covariance is defined as $s_k^2 = \sum_{n \in C_k} (y^n - m_k)^2$. In this expression, y^n is the (1-D) parameter value of a single datum n within the group C_k, and m_k is the mean input value for that group. Eq. (7) can now be re-written as

$J(w) = \frac{w^2 (m_B - m_A)^2}{w^2 (s_A^2 + s_B^2)}$, or

$$J = \frac{(m_B - m_A)^2}{s_A^2 + s_B^2} \qquad (12)$$

giving the measure of the separateness of the two groups A and B in an input space of one dimension. The measure increases as the mean parameter values of the two groups get farther apart, and as the variation within each of the two groups is reduced.

Minimizing the Nonlinearity

The purpose of this paper is to argue that the same methods used to classify data into two discrete categories are also effective at predicting functions with a continuous range of outputs. For continuous functions, the ability of any linear regression to characterize the output is limited by the nonlinearity of the function. In the case of a continuous linear function of many variables, the Fisher Discriminant can be used to exactly divide the input space into a region that will produce outputs below the mean value and a region producing outputs above the mean. In the case of a nonlinear function, the boundary defined by the Fisher Discriminant between regions of the input space will not cleanly

separate the data according to output values, but will be subject to a degree of "interference" because the true division between output classes is not a straight line or flat plane (or hyperplane, in the case of 4 or more input dimensions), but a meandering boundary (Fig. 15).

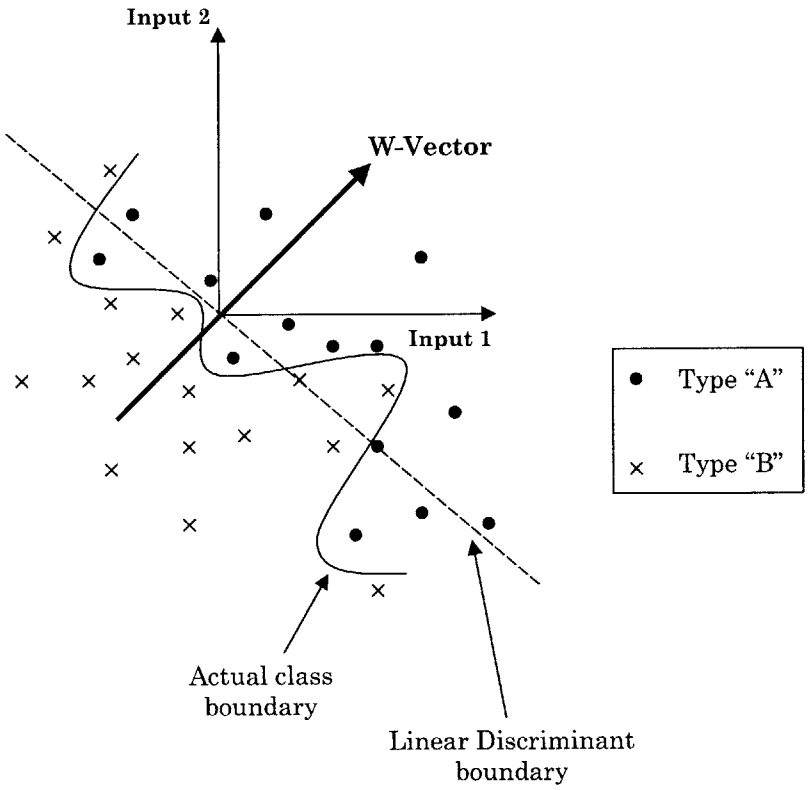


Fig. 15. The boundary between classes in a nonlinear function cannot be completely defined by a linear discriminant.

If our task were to separate the outputs from a nonlinear function into two classes, above- and below-mean, with the minimum of interference, the Fisher Discriminant would provide a way to preferentially select those orientations of **w** that minimize the overlapping of the two classes at the boundary. The quantity **J**, which gives the ratio of

mean separation to clustering density in the discrete two-class problem, would serve to indicate the level of interference between higher-than-mean output and lower-than-mean output in our hypothetical nonlinear function. A smaller value of **J** would indicate a more diffuse distribution of samples belonging to each class in the direction of **w**, and an increase in the region of overlap between the classes. **J** is equivalent to an inverse measure of nonlinearity in a given direction within the input space.

In our discussion of the method of subdivisions, we drew attention to the phenomenon of linearity at different scales (Fig. 8). The objective function we are trying to approximate can be fairly linear at one scale and extremely nonlinear at another. If we now vary one input parameter while keeping the others constant, we can observe that the objective function changes in a more linear way when we vary one input than it does when we vary another.

THE MEMBRANE ALGORITHM: REFINEMENTS BASED ON STATISTICAL TOOLS

Introduction

However well the Fisher Discriminant succeeds in minimizing the region of overlap, we are still left with the question of how well we can characterize a continuous function simply by being able to place a sample input vector in the "above mean output" or "below mean output" category. Even if the power of discrimination between these two classes is great, we still have only determined the likely output of the function to a very low resolution. To increase the resolution of the result, we use the discriminant repeatedly, first dividing the entire data set along the mean output value and then successively dividing up each resulting class along its class mean. Such an approach gives a more resolved answer, but would not be expected to perform any better than a straightforward linear regression, in which a final value of the function output is approximated by the magnitude of the projection of the input vector along **w**.

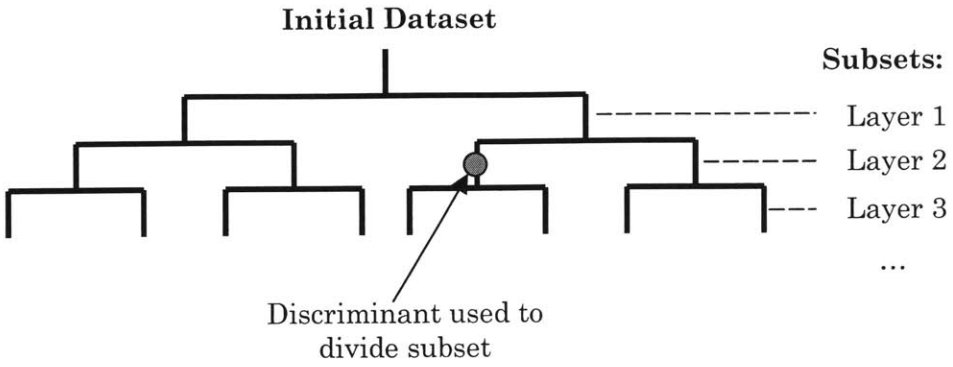


Fig. 16 By dividing the data set into successively smaller pairings, we approximate a continuous representation of the output space

The Fisher algorithm orients **w** to minimize the diffuseness of each class, which has the effect of seeking linearity in the input space. To take full advantage of the special properties of the Fisher Discriminant, **w** must be recalculated at each level of the decomposition of the data set. A decomposition scheme in which **w** is successively recalculated has two main advantages over a simple regression formula:

- In general, a linear approximation will be more appropriate in respect to some inputs than to others, but the emphasis on any given group of inputs, indicated by an orientation of **w** that aligns more closely with them than with others, will vary depending on the region of the input space in question. Different choices of W are appropriate for the different subsets of data being analyzed.
- As we reach lower levels of the decomposition, the range of input values under consideration becomes smaller. Changes in the value of the function in respect to an input variable can resemble a linear relationship over a large range, but become highly nonlinear at a smaller scale. The reverse can also be

true (see *Linear Methods, Fig. 3*). The optimal orientation for **w** can vary as we home in on increasingly specific subsets of data.

Any orientation of the W-vector, no matter how carefully chosen, will necessarily misclassify some inputs in a nonlinear problem. In our branching scheme, a misclassification at the top of the decision tree is more serious than one that occurs lower down. For example, the first classification is the most crude because it can only decide whether a sample belongs in the upper or lower half of the entire catalogue of training data. A misclassification at this level prevents the sample from participating in the successively finer distinctions that occur at lower levels of the decision tree.

Membrane Model

As a way of correcting the misclassification of inputs at each level of the decision tree, we have implemented a histogram method called a "Membrane." The Membrane identifies mistakes made by the existing hierarchy of linear discriminants in classifying the training data, and creates permanent structures that will adjust the classifications of new inputs once the tree is completely "trained," and has been put into service as an estimator. The idea of applying a correction to the classification provided by a linear discriminant is not new; a method known as a Support Vector Machine (SVM)[6] works on the principle of using the mean-squared lengths of vectors between poorly classified points and the hyperplane as bases for a theoretically defined nonlinear boundary. However, because the process of constructing such a boundary is effectively a numerical optimization that becomes more complex with each additional vector, the algorithm

cannot be scaled to accommodate our large set of training data. We are proposing the Membrane as a nonlinear correction scheme that does not increase computational complexity as more correction points are added.

The first step in the construction of the Membrane is to reorient the axes of our input space to align with the vector selected by the Fisher Discriminant (keeping the axes orthogonal), so that the hyperplane perpendicular to that vector, which separates lower-than-median output cases from higher-than-median ones, becomes a horizontal surface through our input space. This is shown for the 3-dimensional case in Fig. 17.

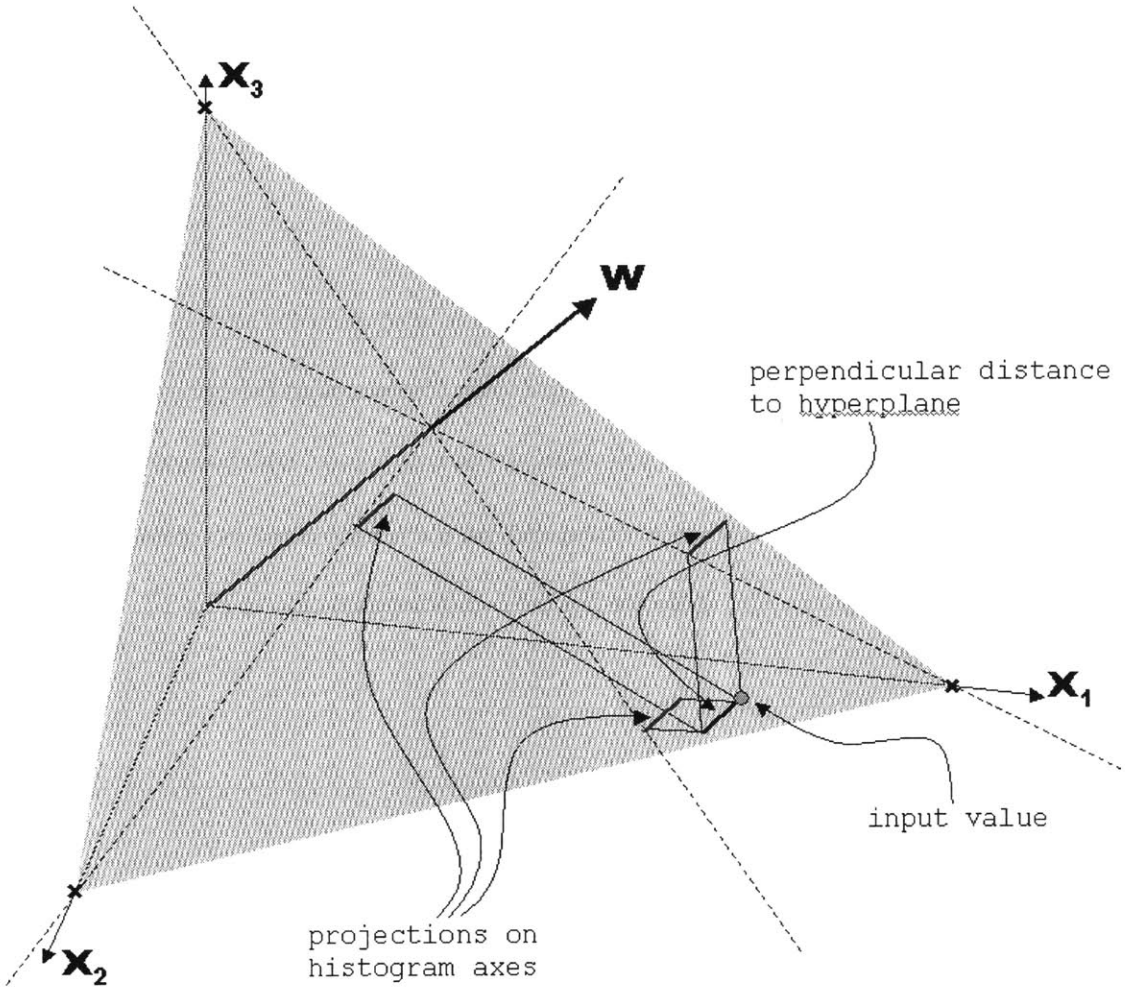


Fig. 17. Construction of axes parallel to the hyperplane along which histograms can be calculated.

During the process of building up the estimator using the training data, we of course have the benefit of knowing the true output, whether above- or below-median, from each training point. Comparing the true outputs from these points to the classification they receive from the linear discriminant, we may now infer the existence of a manifold that intersects the space between cases with lower-than-median outputs (shown as squares in Fig. 18) and higher-than-median (shown as X's), lying roughly parallel to the hyperplane

but separating the two groups according to their true output values, whereas the hyperplane distinguished them only to a linear approximation. In the explanation that follows, we will refer to this postulated manifold as the Membrane.

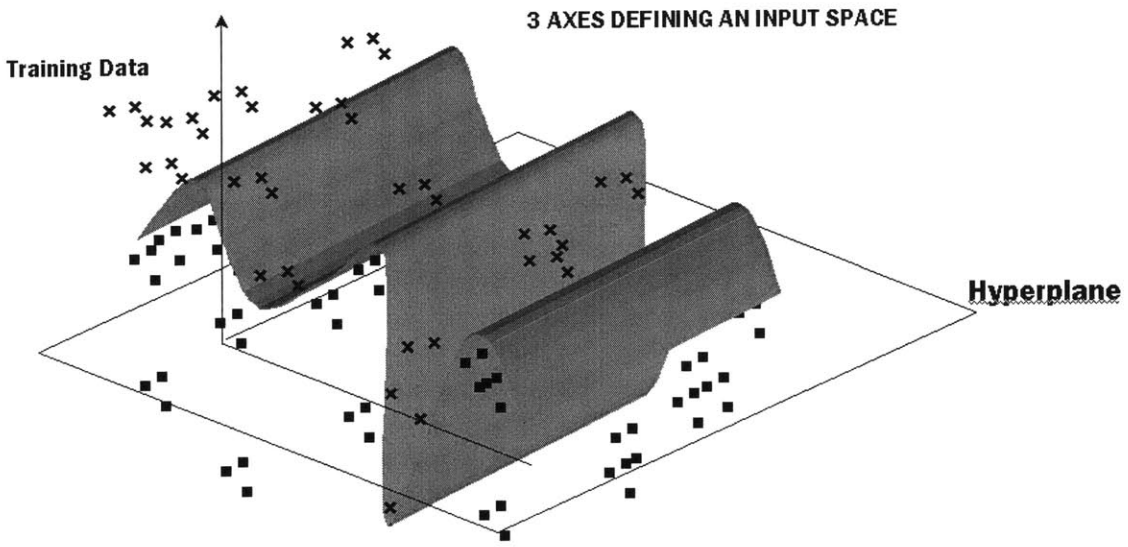


Fig. 18. The hyperplane selected by the Fisher Discriminant algorithm is the flat surface that best separates the training data into two distinct groups in this artificially generated example. The membrane ideally separates the data that the hyperplane can separate only to a linear approximation.

We build up a model of the membrane using histograms to record the deviations of input values that have been misclassified by the hyperplane along with those values that were correctly classified, but which occupy the same region of the input space as the misclassified values. Because of the potentially high dimensionality of the input space, it is not practical to create a single multi-dimensional histogram, particularly when the number of input dimensions is large. If a particular model contains n inputs, and the N training data are distributed uniformly through the input space, the average density of

data points over the range of any given input variable is $N^{1/n}$. For a problem with 30 input dimensions and 1 million data points, this would amount to only 1.58 data points encountered as one follows a line through the input space between the lower and upper limits of one variable, all others remaining constant. Worse, the point density would increase by a factor of only $10^{1/30}$, or 8%, for each order-of-magnitude increase in the total number of training data. Because the number of input variables in our problem is large (30 inputs were selected to represent a 40-input problem), we cannot feasibly construct a multi-dimensional histogram that covers the entire input space with even a minimal resolution. Instead, we have used a probabilistic scheme that synthesizes the information from many partial histograms containing incomplete representations of the input space. For the 30-dimensional problem, we maintain 30 separate 1-dimensional histograms. Each histogram represents the projection of all data points onto a one-dimensional axis running through the input space.

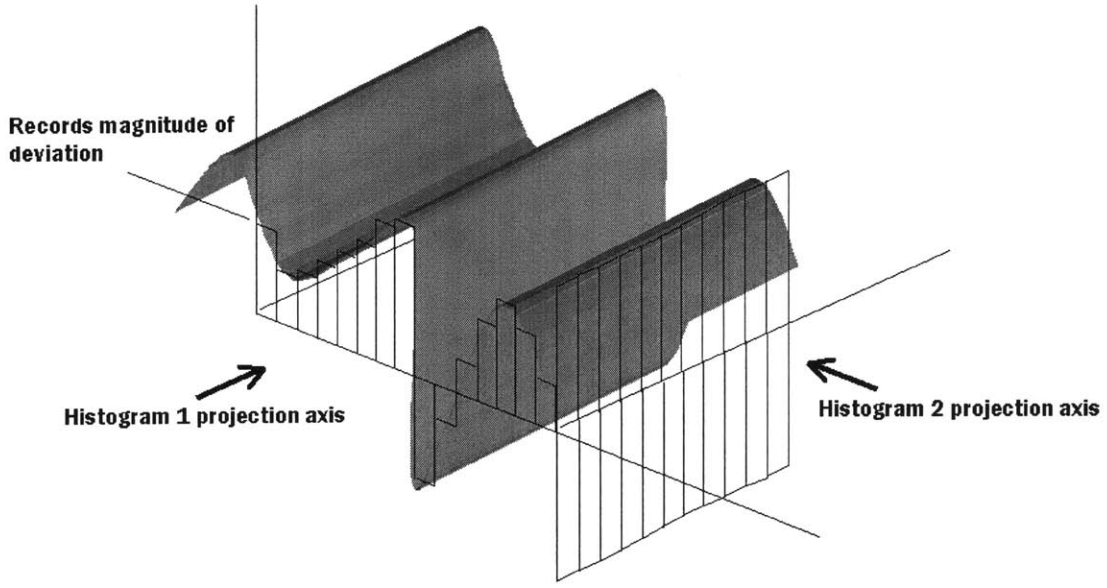


Fig. 19. Two 1-dimensional histograms characterize the membrane surface of this 3-dimensional input space. The height of each bin of the histogram corresponds to the maximum deviation from the hyperplane of any misclassified point that lines up with the bin along the projection axis. Histogram 1 clearly contains more information about the membrane shape than Histogram 2 in this example.

As shown in Fig. 19, each 1-dimensional histogram contains an impression of the shape of the membrane, although each one's information is incomplete and oversimplified, being only a projection of a higher-dimensional space. This operation in shown for the original orientation of the input dimensions in Fig. 20.

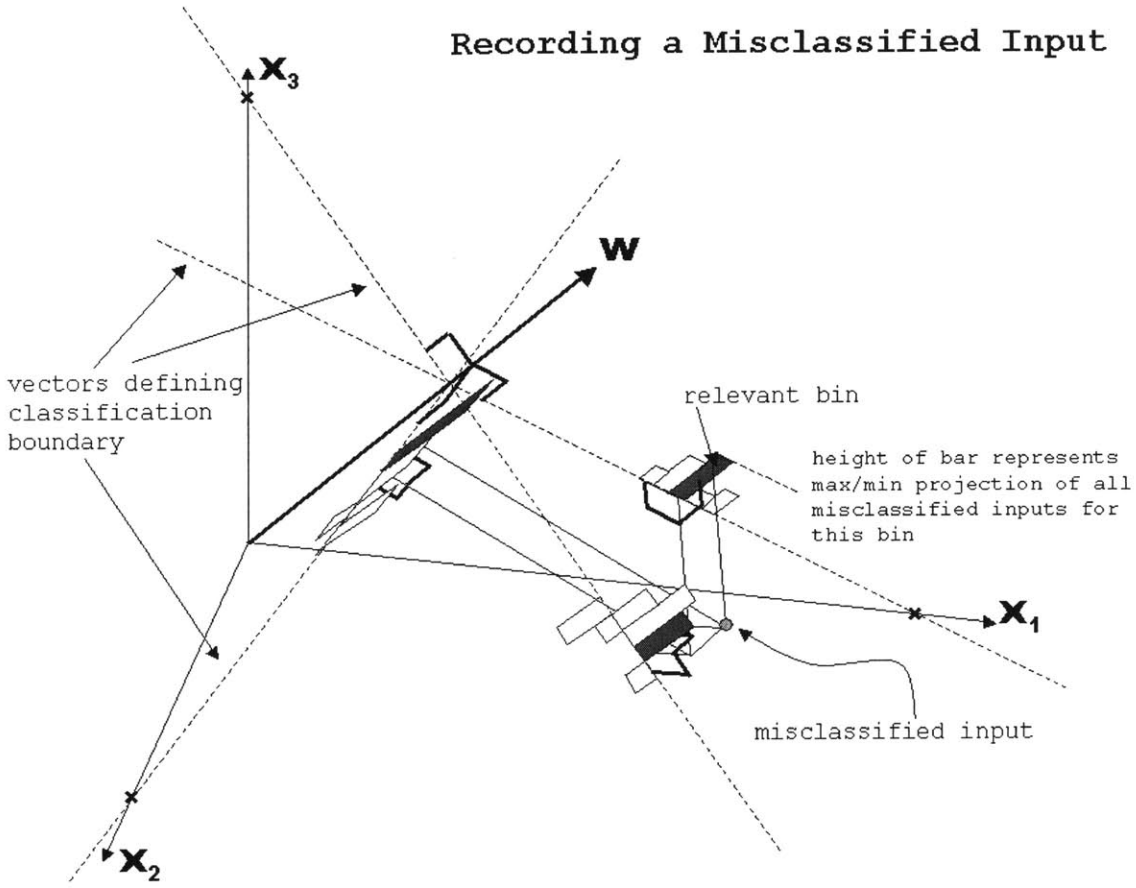


Fig. 20. Each training data point that has been misclassified will recorded differently by each of the histogram axes.

We cannot make conclusive assessments of the true shape of the membrane based on the evidence of any one histogram, but we are able to make statistical predictions about its shape by combining the information from multiple histograms. In certain cases (including the case shown in Fig. 19), a single histogram may contain a clear indication of the true boundary between low-output and high-output cases. For example, if the hyperplane has misclassified nine out of ten of the input values belonging to a certain bin of one histogram, we may predict that future values that belong to this bin should be

oppositely classed. They should be assigned to the group that was *not* originally indicated by the hyperplane's segregation of the input space.

Two different types of histogram are used for each axis of the hyperplane: one to keep track of input values that were found on the low side of the discriminant boundary, but that actually belong (because of the high output value they produce) with the inputs on the high side, and another histogram for inputs that were classed as high, but actually have low outputs. In both cases, the number of misclassified points is recorded alongside the number that inhabit the same part of the input space, yet were correctly classified by the hyperplane. By "the same part of the input space," we mean the region that projects onto the histogram axis within the assigned limits of one particular bin, and which lies within a distance of the hyperplane limited by the maximum distance of any misclassified case belonging to that bin. By comparing the number of correctly and incorrectly classified points that lie within such a region of the input space, we determine the overall probability of a misclassification in that region.

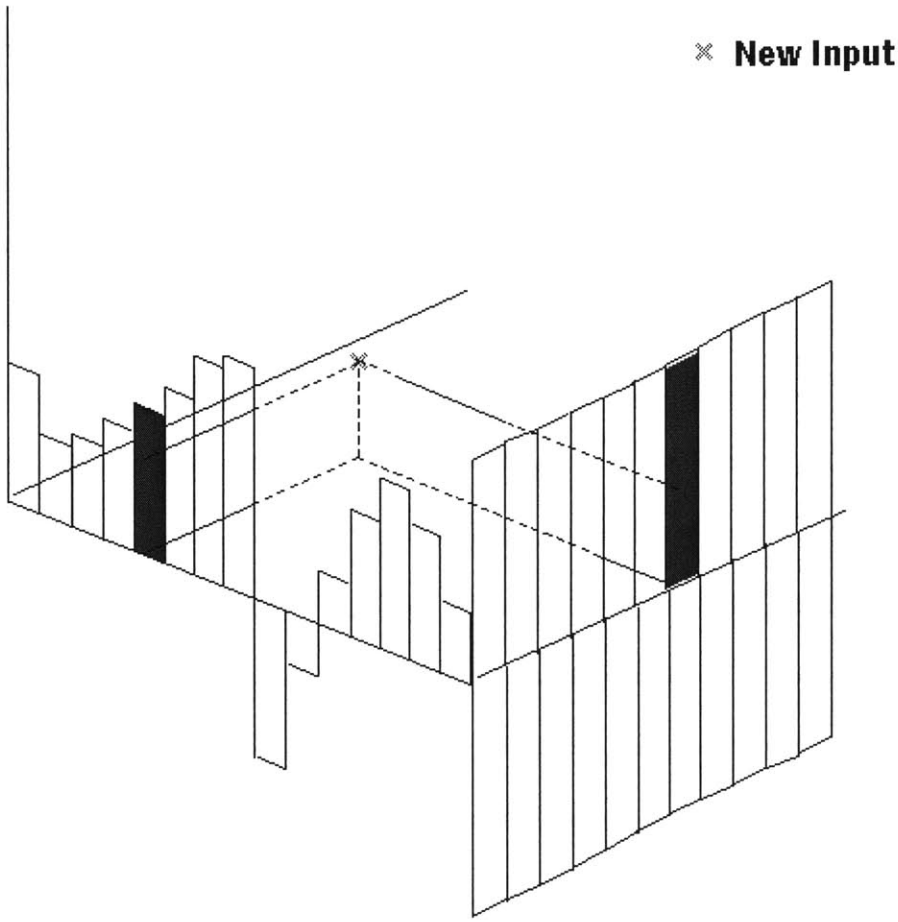


Fig. 21. After the membrane has been created using the training data, the histogram bins (shaded) corresponding to a new input vector can be found by projecting the vector onto each of the histogram axes.

Weighting Information from Multiple Histograms

The shape of the membrane is encoded in the histograms. Each case from the training data has an output value given by the original building simulation – the output that the estimator must learn to predict. We label each case as "well classified" or "misclassified," according to whether or not the classification by the linear hyperplane agrees with the true output value. To be well classified, a case must lie above the

hyperplane if its output is greater than the median of the set, and below the hyperplane if its true output is lower-than-median. We gradually build up a picture of the way in which the true boundary between greater-than-median and lower-than-median cases undulates with respect to the hyperplane by recording the misclassifications in the histograms. After this process has been carried out for each of the hyperplanes in the overall decision tree (one hyperplane per binary classification), we are prepared to attempt the prediction of output values for new cases that have not been evaluated with the original building simulation. When confronted with a new input vector to classify, the estimator first classifies the point based on whether it is located above or below the hyperplane in the input space. The histogram data is then consulted to determine if that classification must be adjusted due to the irregular shape of the membrane. The particular bin that corresponds to the projected length of the new input vector (Fig. 21) along each histogram provides several statistics attesting to probability of the new input being misclassified. They are:

1. **The probability of misclassification, given the history of training data encompassed by the bin and classified correctly or incorrectly by the hyperplane.**
2. **The total number of training data points logged by the bin.**
3. **The distance between the new input and the hyperplane, as a fraction of the maximum distance between any misclassified training point and the hyperplane.**

If the distance between a new point and the hyperplane is greater than the maximum distance of a misclassified training point ever recorded by any one of the bins into which the new point project, we may conclude that the point has, after all, been correctly classified by the hyperplane (Fig. 22). This surmise becomes more reliable the more cases in total have been recorded as aligning with the particular bin.

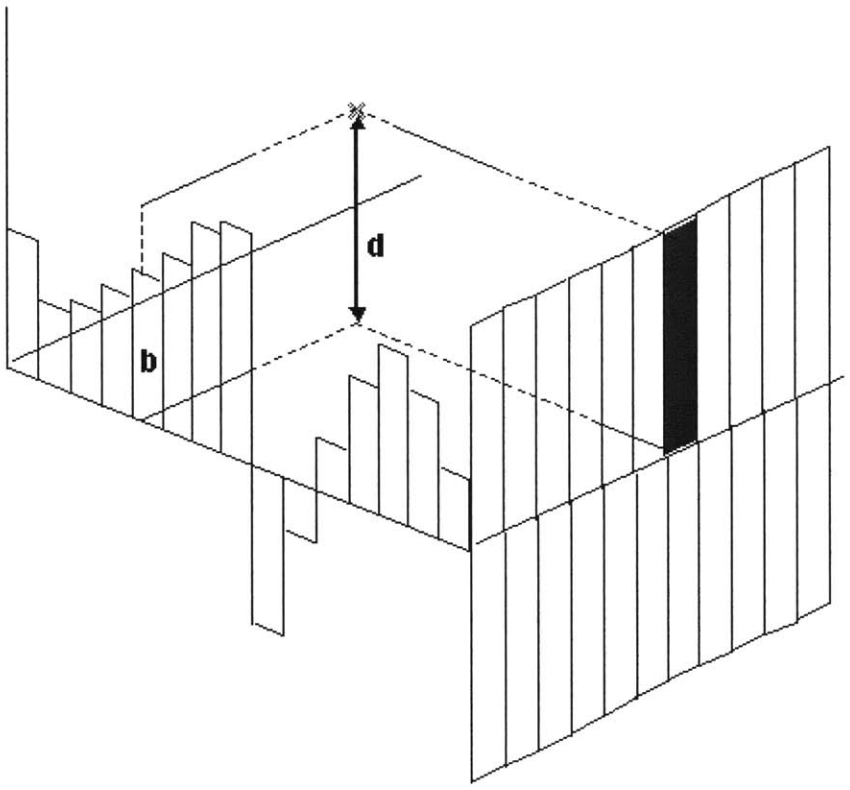


Fig. 22 The new input vector, although lining up with a bin in each of the two histograms, belongs to only one. Its distance *d* from the hyperplane surface is larger than any misclassified vector among the training data belonging to bin *b*. As such, this point would be considered correctly classified by the hyperplane because we conclude that it lies above the membrane surface based on the information from bin *b*.

Discussion of the Weighting Criteria

The second criterion on which histogram information is rated, the total number of training data to have been logged in the indicated bin of the histogram, serves principally to screen out the contributions of histogram bins that have collected too few data to offer statistically meaningful results. Criterion 1 is described in Fig. 23 for two artificially created membrane shapes. On the basis of this criterion, which is the probability that an input vector belonging within the scope of one of the bins has been misclassified by the hyperplane, the histogram shown rates much better in respect to the membrane in Fig. 23*a* than the one in Fig. 23*b*. In case *a*, there are very few points *above* the membrane surface that fall within the scope of any of the histogram bins, because the height of the bins shown in the figure closely follows the projected contour of the membrane surface. The misclassified points in each of these bins therefore vastly outnumber the correctly classified ones that lie above the membrane surface, giving all the bins a high reliability rating from criterion 1. The histogram in case *b* has high reliability in its leftmost bins, but that rating diminishes as we move to the right along the axis, where it becomes less clear from looking at the bin profile whether the corresponding points lie above or below the membrane surface. As we move along the histogram to the extreme right in figure *b*, the probability of a misclassification continues to drop until it is finally quite unlikely that a point belonging to the bin lies below the membrane surface. Since we are able to make this conclusion with high accuracy, the reliability of the negative verdict on misclassification is high in respect to criterion 1, just as the positive verdict had high reliability at the leftmost end of the histogram.

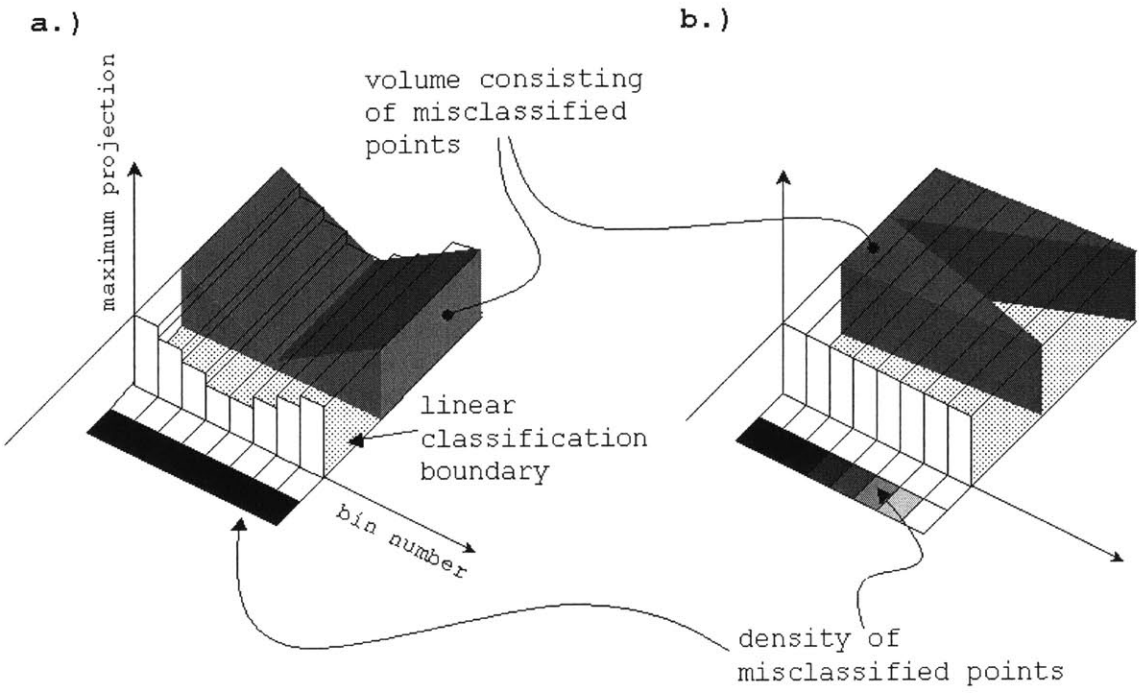


Fig. 23. Histograms use various means to identify the surface geometry of the membrane. In these artificially generated examples, the maximum deviation of a misclassified point is shown for each bin of a particular histogram, along with a grayscale panel showing the ratio of misclassified to correctly classified points in each bin.

By choosing a different histogram axis to represent the membrane surface in Fig. 23, we lose the description of the contour of the surface, and the probability of misclassification for each bin becomes ambiguous, since a roughly equal number of correctly and incorrectly classified points are projected onto the new histogram (Fig. 24).

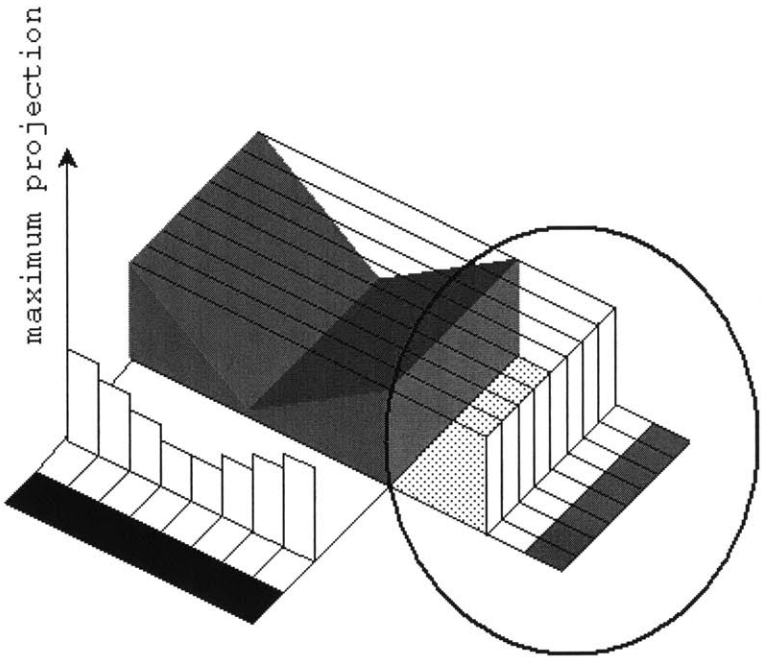


Fig. 24 The information recorded for the same membrane surface as in Fig. 23a has a different character when we change perspectives to the other histogram.

However, we gain new information by switching perspectives that relates to Criterion 3 - the distance from a new input point to the hyperplane as a fraction of the furthest distance within the same bin from any misclassified training point to the hyperplane. This information is useful because the closer a point is to the hyperplane, the more likely it is that the point has been misclassified. To understand the reason for this, we can consider a simplified membrane like the one in Fig. 25. Although a real membrane surface in our problem would vary in as many as 30 dimensions, and this membrane varies in only one, it serves to illustrate the principle that the hyperplane is the linear average of the membrane – its "DC" component, in relation to which the membrane height oscillates up and down but never permanently diverges. The points lying between the membrane and

the hyperplane are misclassified, but if we slice through the input space parallel to but slightly above the hyperplane (as in Fig. 25) we encounter fewer misclassified points within each slice as we get further away from the hyperplane. This demonstrates the usefulness of Criterion 3 – the relative height of new input points above the hyperplane - as a measure of the likelihood of its misclassification. A new input vector that is closer to the hyperplane is more likely to belong to the set of misclassified points.

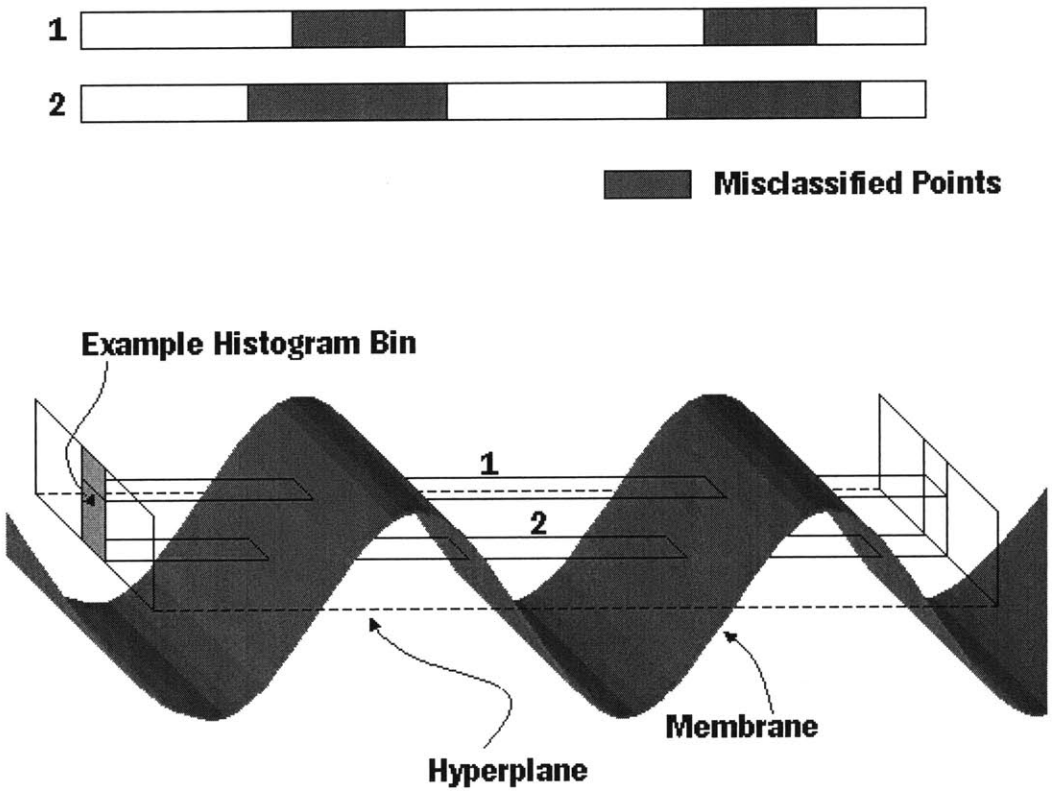


Fig. 25. The higher a point lies above the hyperplane, the less likely it is to belong to the set of misclassified points beneath the membrane surface.

Measuring the Aberration

A higher-dimensional space than the 3-D one depicted in Fig. 24 submits to the same analysis. A space of *n* dimensions contains a linear hyperplane of dimension *n-1*. If an input has been misclassified by this hyperplane boundary, it is considered more "aberrant," the greater its distance from the hyperplane. The distance is measured orthogonal to the hyperplane, so that the line along which the measurement is made runs parallel to the W-Vector. The height of the projection above the axis of the histogram is only recorded if it is the most aberrant projection of an input value yet recorded by that particular bin of the histogram.

Each axis sustains two histograms – one for inputs located above the hyperplane whose outputs evaluate to a number lower than the median, and another histogram of inputs found below the hyperplane whose outputs are higher than the median. Each of the two histograms contains data about the number of inputs projected into each bin and the maximum aberration of any misclassed input value in each bin. The tally of inputs for each bin is separated into the number of inputs that are misclassed, and the number that, though classed correctly, fall within the region of the input space between the positions of misclassed values. As shown in Fig. 26, given a nonlinear boundary surface separating high- and low-valued points, a well classified point located above the surface could be closer to the hyperplane than a misclassified point located below the surface.

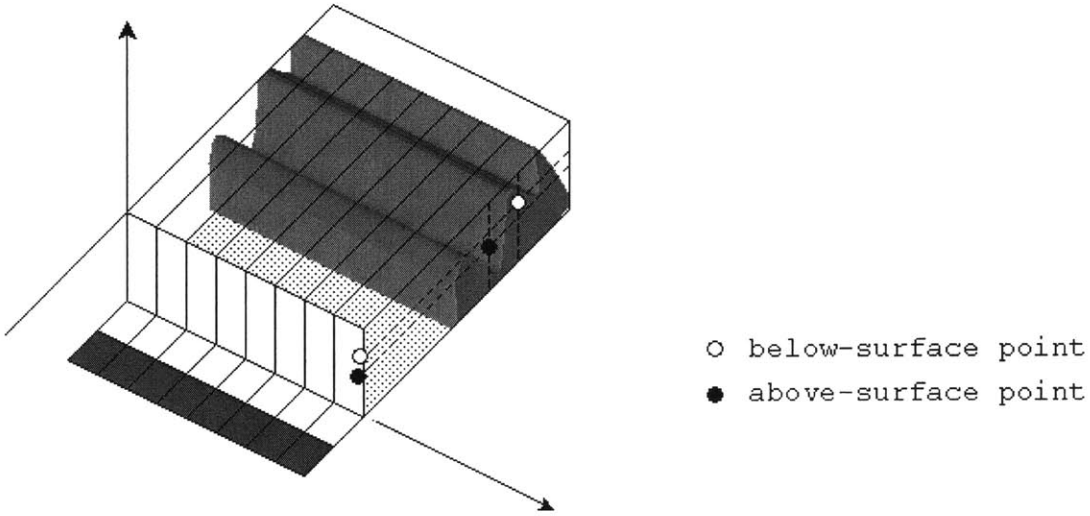


Fig. 26. The linear boundary surface used to discriminate between high- and low-valued points approximates the real "boundary," a nonlinear surface along which the output values corresponding to all points are theoretically equal. Because the surface is nonlinear, the precedence of points can be confused when they are projected onto a single histogram.

The projection of the two points onto the histogram pictured in Fig. 26 shows the distances between the points and the hyperplane only. This particular histogram does not capture the distinction between the high-valued input and the low-valued one. The projection onto a single axis has destroyed certain information that would lead to a correct ordering of the input values. Ambiguities such as these require that we find a robust means of coordinating inputs from multiple histograms to minimize erroneous judgments.

The incompleteness of the data represented in a histogram requires us to make probabilistic judgments when using histogram data to classify new input points. Between

the hyperplane and the projected height of the most aberrant misclassified point, there will be many correctly classified points with intermediate projected heights, like the "above-surface" point in Fig. 26. To make the best possible guess as to whether a new point found in this range is misclassified, we can only ask if the majority of other points in the same range are misclassified. Each axis onto which input values have been projected will express a different view of the data. Any new input point can be matched with the appropriate bin from each histogram, but each will return a different probability that the input has been misclassified by the linear estimator. If we maintain 30 variables in our model, each new point will correspond to 30 different estimates of the probability that this point has been misclassed.

Reconciling the Experts

We are now faced with the question of how to value each of the histogram estimates in relation to each other. No single histogram is likely to contain all the information necessary to make an informed choice about how best to classify the inputs, so the optimal decision will be a negotiated mixture of many single estimates. In general, a probability above 0.5 would indicate that a point was more likely misclassified, and below 0.5 the original classification would tend to be correct. Since each estimate is a numerical probability between 0 and 1, the simplest scheme for combining them would be to use either the average or the median value to determine an overall verdict of "misclassified" or "correctly classified." However, as the probability varies strongly between different histograms and their verdicts are therefore determinedly contradictory,

averaging will tend to dilute the useful information that is exclusive to individual histograms.

The Fuzzy Preference Relation

It is more correct to say that each histogram offers unique information, than to say that all approximate a "correct" average. A given histogram will be quite accurate in representing certain features, and will have nothing useful to say about others. Unfortunately, we have no explicit measure of a histogram's accuracy in reporting a given probability of misclassification – only a few vague guidelines. For example, we can say that 1.) histogram bins containing many recorded hits provide more reliable information than those that have only recorded a few. We can also judge a verdict of "misclassified" as more likely if 2.) the input in question had a small projected height above or below the hyperplane, relative to other inputs recorded in a given bin. Finally, we have more confidence in a verdict if 3.) the corresponding probability is closer to 0 in the case of a correctly classified point, or 1 in the case of a misclassified point. Probabilities closer to a value of 0.5 are inherently more ambiguous, or of higher "entropy." Claude Shannon's *Mathematical Theory of Communication*[7] uses this quantity of entropy in the analysis of communication channels to represents the degree of variability of the channel. In a system that produces a stream of bits, each bit having a value of 1 or 0, the entropy can be said to decrease as restrictions are placed on the probability of reading one value or the other from the bit stream as it emerges from the channel. As an example, suppose we find that a particular bin records n misclassified inputs of which the most aberrant is a distance a from the hyperplane. There may also be

a number m of correctly classified points that come within the same range of input values that are also within a distance a from the hyperplane. These are allotted to the same bin. The relative proportions of correctly and wrongly classified points provide some guidelines about our ability to classify any new points that fall within the bin's input range. The *probability* of a new misclassification in a given bin is based on the relative number of misclassifications already recorded. We define the probability p of a misclassification as

$$p = \frac{n}{m+n} \qquad (13)$$

If this probability is close to 0.5, we have very little basis for predicting whether future input vectors assigned to that bin will be correctly or incorrectly classified. On the other hand, if p is close to 1, we can be reasonably certain that any new vector assigned to the bin will turn out to be a misclassified point, like the majority of those that preceded it. The Shannon Entropy **H** can be used to assign a numerical value to the predictive usefulness of the bin based on its probability value. **H** is defined as

$$-\sum_{i}^{n} p_i \log p_i \qquad (14)$$

where each p_i represents the probability of a possible state of the system. In our case, there are two possible states: any new input applied to the system will either be correctly classified or misclassified by it. There are two possible states, namely the probability of misclassification by a given bin, p, and the probability of a correct classification, 1-p. Substituting into (14), we have

$$\mathbf{H} = -p \log p - (1-p)\log(1-p)$$

which is graphed for values of p between 0 and 1 in Fig. 27.

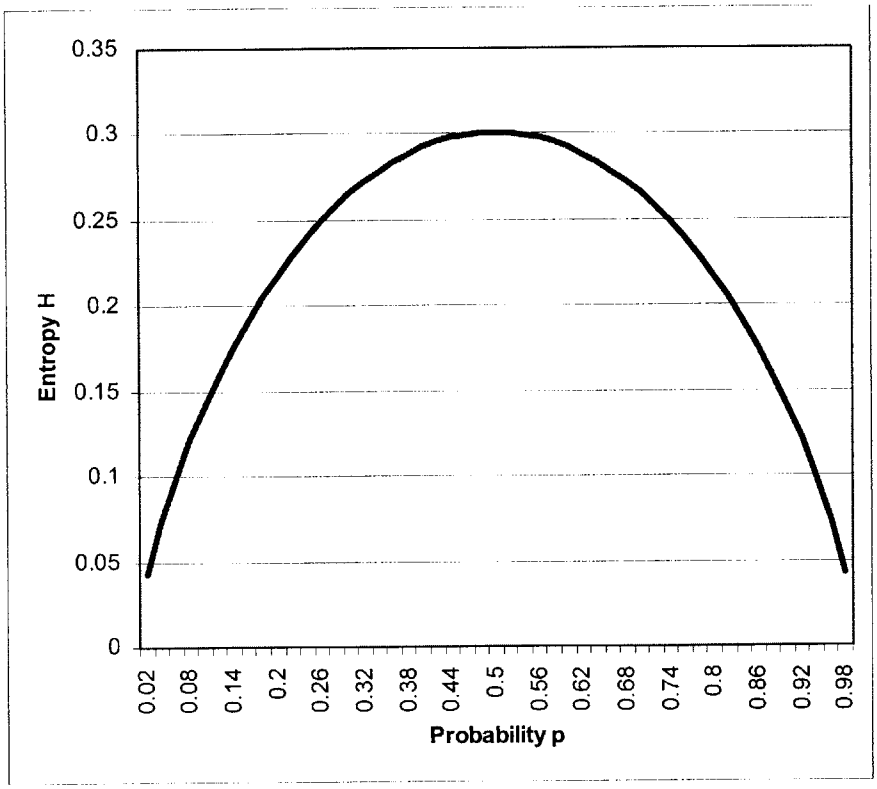


Fig. 27. The entropy of an experiment with 2 mutually exclusive possible outcomes, in relation to the probability that the first outcome will occur. The high entropy at P=0.5 indicates the condition of maximum uncertainty

The entropy is greatest for the value $p = 0.5$, and is minimized for $p = 0$ and $p = 1$, in agreement with our natural intuition about the relative uncertainty of predictions made using those values of p. Using the entropy formulation as a reference, we rate a probability of 0.5 as having zero usefulness, and apply monotonically higher ratings to histogram bins the greater the value of the expression $|p - 0.5|$.

Because each histogram's estimate may have unique information to offer, we would prefer to preserve information from each, weighting the histograms according to the overall quality of prediction that each provides. We have already identified three criteria

on which to judge the soundness of particular histogram bin's prediction, which we re-phrase as follows:

1. **Difference between predicted probability of misclassification and the value 0.5**
2. **Total population of points recorded by the bin**
3. **Greatest recorded distance of a misclassified output from the hyperplane**

Each time we attempt to predict the output group based on the input coordinate values, a different criterion may contain the crucial information. The criteria are not comparable on a numerical basis, so instead of asking which histogram has the highest weighted sum of scores, it would be more appropriate to pose the question in a weak form, namely "which estimate scores well in the greatest number of categories?" In general, we would prefer to use a histogram whose representation of the feature of interest is guaranteed by all available guidelines, rather than excelling only in one. We cannot know which guarantor of histogram quality is most relevant, so we would prefer to have some confirmation from all of them. More particularly, we would like to know which estimates can be said to be the least overshadowed by other estimates when all the criteria are considered.

We have used a decision-making algorithm called a "Fuzzy Preference Relation" to find optimal combinations of histogram estimates. The Fuzzy Preference Relation[1] is a development of the idea of a "fuzzy set," first introduced by Zadeh[9]. A fuzzy set is a group of elements for which the requirements of membership are not strict; that is, there

[1] After Orlovsky[8]

are different degrees to which an element can "belong" to the set. This "fuzziness" is reflected in the comparison between one histogram and another. A histogram x may not be strictly better or more useful than histogram y, but could dominate y on the basis of certain criteria only, just as y may simultaneously dominate x in respect to other criteria. In general, the dominance of one histogram over another will not be complete, or "strict," because the measure of dominance comes from several different sources, and the sources are allowed to disagree. In our case, these sources are the several different criteria we would like to apply to a comparison of 2 histograms at a time. The traditional approach to Fuzzy Decision-Making is to define a *preferability index* that ranks the members of a set - in our case, histograms - according to a measure of their reliability (Baas and Kwakernaak[10]). However, this idea is not appropriate to our application because we wish to preserve the special information that an unreliable source may be able to provide, even if it is not favored in general. Instead, we have pursued a weighting scheme that asks, not "which is the most reliable source," but "to what extent do the fuzzy ratings imply that [source] x_1 is better than [source] x_2?"[2] Each of our 3 criteria provides a basis on which to judge the dominance of one histogram over another. We can visualize these judgments as coming from 3 different "experts" on the question of histogram dominance. One expert specializes in applying the first criterion, and in a match-up between two histograms, always chooses the one that dominates on this criterion as the more reliable. In our model, this first expert chooses the histogram with the higher population of recorded input values. The second expert discriminates between histograms based on the projected height of the present input vector. The third will choose based on the proximity of each histogram's probability value to either 0 or 1. We have no way of combining the

[2] Dubois[11], pp. 283.

judgments made by the experts because they are of completely different kinds. The most we can do is to observe how often the experts' opinions align with each other. The criterion used by each expert is listed at the left of Table 1. These criteria are evaluated for each of 4 hypothetical histograms, and the results displayed in the table.

	Histogram X1	**Histogram X2**	**Histogram X3**	**Histogram X4**
Probability of Misclassification	0.26	0.71	0.34	0.49
Bin Population	4	15	22	19
Distance from Hyperplane	0.03	0.02	0.05	0.01

Table 1. Performance statistics for 4 histograms

The values given to each histogram are interpreted as follows: in the first row, the probability furthest from a value of 0.5 is preferred. In the second, the higher the bin population, the more reliable the histogram. In the third, the histogram showing smallest projected height for the given input value is the most reliable. With this basis for preferring one histogram over another, we can begin a pairwise comparison. In each pairing, if we decide that the unanimous choice of one histogram over the other (that is, the agreement in all three categories that one histogram is superior) should be represented by a 1, and the unanimous choice against a histogram by a 0, we fill in the remaining scores as 0.33 for the vote of one expert, and 0.66 for two experts' votes. These numbers do not amount to a weighting scheme, but will rather be used as a way of coding for the poll of experts on each histogram pair. They indicate the extent to which a histogram can be said to "dominate" its partner.

Using the Fuzzy Preference Relation proposed by Orlovsky, we try all possible pairings of the histograms to determine the degree to which each dominates the other. In Fig. 28, these pairings are shown in a matrix, in which the numbers represent the degree to which the histogram labeled by row dominates the histogram labeled by column. The number between 0 (no dominance) and 1 (strict dominance) that is used to describe the intermediate degrees of dominance is called a "preference" in fuzzy logic. The Fuzzy Preference Relation is a matrix made up of the preferences that result from each pairing of two histograms. The numbers on the diagonal of the matrix are all "1," indicating that there is no meaningful preference of a histogram over itself.

HISTOGRAM \ PARTNER	x_1	x_2	x_3	x_4
x_1	1.0	.33	.66	.33
x_2	.33	1.0	.33	.33
x_3	.33	.33	1.0	.66
x_4	.66	.33	.33	1.0

Fig. 28. The matrix of fuzzy preferences for the histogram performance figures in Table 1. Where the cross-diagonal entries do not sum to 1, there was a tie[3] between histograms in at least one category.

Comparing the histograms against each other in groups of 2, we award points to one histogram for each category in which it dominates the other. Since we are using 3 criteria for comparing the histograms, dominance in each category is awarded 0.33 points, so that dominance in all categories should add up to a score of 1 except where there is a tie (see

[3] In this context, a tie can only arise from two situations: the fact that cases closer to the hyperplane are more likely to be misclassified does not allow us to decide precedence between a bin giving a verdict of "misclassified" in which the aberration is great, and a bin giving a verdict of "well classified" in which the aberration is small. In each bin, the relative size of the aberration tends to support the verdict, and neither bin can be said to be more correct on this score. This would also be true if the relative aberration tends to contradict the verdict in both cases.

footnote 3). In Fig. 28, we can see that X2 dominates X1 to the extent of 0.33, and X1 dominates X2 to the extent 0.66. These two preferences are complimentary, and must add to 1. They also show that X1 dominates X2 to a larger degree than vice versa, and we can extract a "net preference" value from the comparison of the two numbers – the degree to which one dominance overwhelms the other. In Fig. 29, the matrix is redrawn, but this time the numbers have been reduced so as to indicate only the *net* preference. The dominance by X1 over X2 becomes 0.33, which is the net difference between X1's dominance of X2 and X2's dominance of X1. To the cell showing the dominance of X2 over X1, we write in a 0.

	x_1	x_2	x_3	x_4
x_1	0.0	0.0	.33	0.0
x_2	0.0	0.0	0.0	0.0
x_3	.00	0.0	0.0	.33
x_4	.33	0.0	0.0	0.0
Extent Non-Dominated	.66	1.0	.66	.66
	x_1	x_2	x_3	x_4

Fig. 29. The matrix of *net* fuzzy preferences based on the values in Fig. 28. The largest number in each column is the maximum extent to which each histogram is dominated by another. The tallies at the bottom give the residual (1-*max. domination*).

Fuzzy Decision Making

The net preference numbers in the matrix in Fig. 29 derive from two set theory operations defined by Orlovsky as the "fuzzy indifference relation" and the "fuzzy strict preference relation." A "fuzzy set" is a set defined by a membership criterion that does not simply include or exclude elements, but includes them to varying degrees. The degree of

inclusion is given by a membership function μ, such that the membership of an element x in a fuzzy set **C** is given by $\mu(x)$.

Orlovsky demonstrates that the value of a membership function $\mu(x)$ can be interpreted as the degree to which something is true of the relationship between x and y. If $\mu(x)$ gives the degree of membership in a set, then a function $\mu(x, y)$ gives a degree of membership indicated by x, y, and the relationship between them. For our purposes, the value of represents the degree of truth in the statement, "x is preferred to y," expressed $x \geq y$. Orlovsky refers to the fuzzy set **R** constituted by this $\mu(x, y)$ as a *Fuzzy Preference Relation* (FR), because it admits the elements x and y to the degree that $x \geq y$. Accordingly, an FR $\mathbf{R}^{-1}$ is defined by the membership function $\mu^{-1}(x, y) = \mu(y, x)$ [4]. New FRs can be derived from **R** and $\mathbf{R}^{-1}$ as follows: If $\mathbf{R}^{e} = \mathbf{R} \cap \mathbf{R}^{-1}$, then $\mathbf{R}^{e}$ is called the *Fuzzy Indifference Relation*, because it captures the degree to which a preference of x to y is compensated by a preference of y to x. Similarly, a relation $\mathbf{R}^{s} = \mathbf{R} \setminus \mathbf{R}^{-1}$ is a *Fuzzy Strict Preference Relation* because it is the extent to which x is preferred to y beyond the preference of y to x.

Fig. 28 shows a matrix of preferences **M**, where $\mathbf{M}_{ij} = \mu(h_i, h_j)$ and h refers to a histogram, and the case *(i=j)* is the trivial comparison of a histogram against itself. We have defined a function $\mu(h_i, h_j)$ by awarding 0.33 points for each criterion on the basis of which h_i is preferred to h_j. The elements of **M** are therefore given by a fuzzy

[4] Orlovsky, pp. 157

relation **R**, and the *net* preferences in Fig. 29 are given by the *Strict Preference Relation* $\mathbf{R}^s$.

It should be noted that in applying the Strict Preference Relation to the bins, we are not taking account of one important criterion that may influence the preferences significantly. The orthogonal axes along which the bins are constructed, taken along with the *W*-vector, represent a sufficient basis for the full input space, just as did the original input parameter directions. The axes of the space have been reoriented parallel to the hyperplane, so each represents a mixture of components from the variables originally chosen for the given binary classification of training data. Since each of the inputs has a certain importance for the output, both independently and in conjunction with other inputs, it is likely that the bins of certain axes will be inherently more significant than others. This disparity is not explicitly represented by our scheme of weighting criteria, and may represent an intractable source of error in the Fuzzy Preferences.

Configuring the Membrane

The previous section on the Fuzzy Preference weighting method completes our discussion of the role of the Membrane in improving the accuracy of the estimator. Certain difficulties related to the deployment of the Membrane in a real application remain to be addressed. The analysis presented in this chapter has not covered the determination of the following parameters that control the Membrane's practical performance:

1. The number of histogram bins assigned to each axis of the input space

2. The total number of training cases used to populate the histogram bins.

3. The number of input variables involved in the classification at each decision point

The first issue requires us to make a compromise between the resolution at which we characterize the membrane and the statistical certainty with which we do so. For a given number of training cases, defining a large number of bins along each axis into which to sort the cases provides appraisals of new input points that are highly specific to their locality in the input space. On the other hand, the more bins are created, the fewer training data will be observed to project into each bin, lowering the statistical value of a verdict reported by any one bin. The outcome of Point 1 therefore depends on Point 2; the more training cases we can accommodate initially, the larger the optimal number of bins will be for each axis. Our program, constrained by the memory limits of the 512-Mb/RAM workstation on which the simulation was assembled, can use up to 15,000 training cases from the building model to construct the estimator framework. At this capacity, we have determined through a process of trial-and-error that the optimal number of bins to use for each histogram-axis is about 30. In general, the best choice for Point 3 – the optimal number of input parameters to include in each binary classification – is the full complement, since a given set of input dimensions generally provides more information than any subset that it contains. In the case of a Membrane, however, the decision reached for a given classification is an approximation based on many competing suggestions, and is beset by the noise from the ambiguity of a projection of an *n*-dimensional space onto one axis. The optimal number of inputs will likely be something less than the full complement, since the less influential inputs will tend to confuse rather than clarify the verdict of the Membrane. We have found that choosing a subset of 30 out

of a possible 40 inputs provides the best balance of information content and noise. A different set of 30 inputs may be chosen at each branch point in the decision tree.

REAL-TIME OPTIMIZATION WITH SIMULATED ANNEALING

Overview of the Simulated Annealing Procedure

The optimization problem is to try many different configurations of building parameters until we can be satisfied that one set has been found that minimizes the energy consumption. For a concave function, the optimal solution is found in the shortest series of steps by using a gradient-search algorithm to modify an initial guess through several iterations. Newton's Method[5] may be employed to control the scope of successive adjustments and prevent divergence. This technique is not appropriate for functions that are not everywhere convex, as a gradient search may converge at locally minimal points in the design space that represent inferior solutions.

The model of building physics contained in the Design Advisor software can be represented as a function in several variables giving the total energy consumption of the building. Although the function does not have an explicit formulation, approximations to the gradient at a given coordinate location can be found through successive evaluations of the function. In this way, the minimum value of the building function could be found using a gradient search, provided the underlying function is convex. However, the function contained in the building model can be shown to be nonconvex through simple experiments. For example, we can see that given a certain elevation of the sun as seen from the window of a building, there will be a particular angle of the blinds A, equal to the sun elevation, that maximizes the amount of direct sunlight that the room receives.

[5] As summarized in Papalambros[12] pp. 151-152

We have a lower solar input for angles that are less than this maximizing A, but a lower solar intensity will also result from greater angles. Starting with the blinds set horizontal, we observe the sun intensity increasing as we increase the angle to A, and decreasing thereafter. If we further assume both that the room is presently being cooled to maintain a comfortable temperature and that the sunlight is sufficiently bright at any angle to maintain the minimum level for work activities, the angle A is the least efficient angle for energy conservation because it maximizes the surplus of solar heating. In at least one input dimension, the blind angle setting, a gradient search would either reduce or increase the input value depending on whether it began just above or just below A (Fig. 30). If the blind angle that minimized the need for cooling were less than A, a gradient search would never correctly identify it if it began with an angle value greater than A. Applied to a nonconvex function, the gradient search will reveal only erroneous local minima. We have discovered a nonconvexity in the building model that by itself disproves the convexity of the general building energy function.

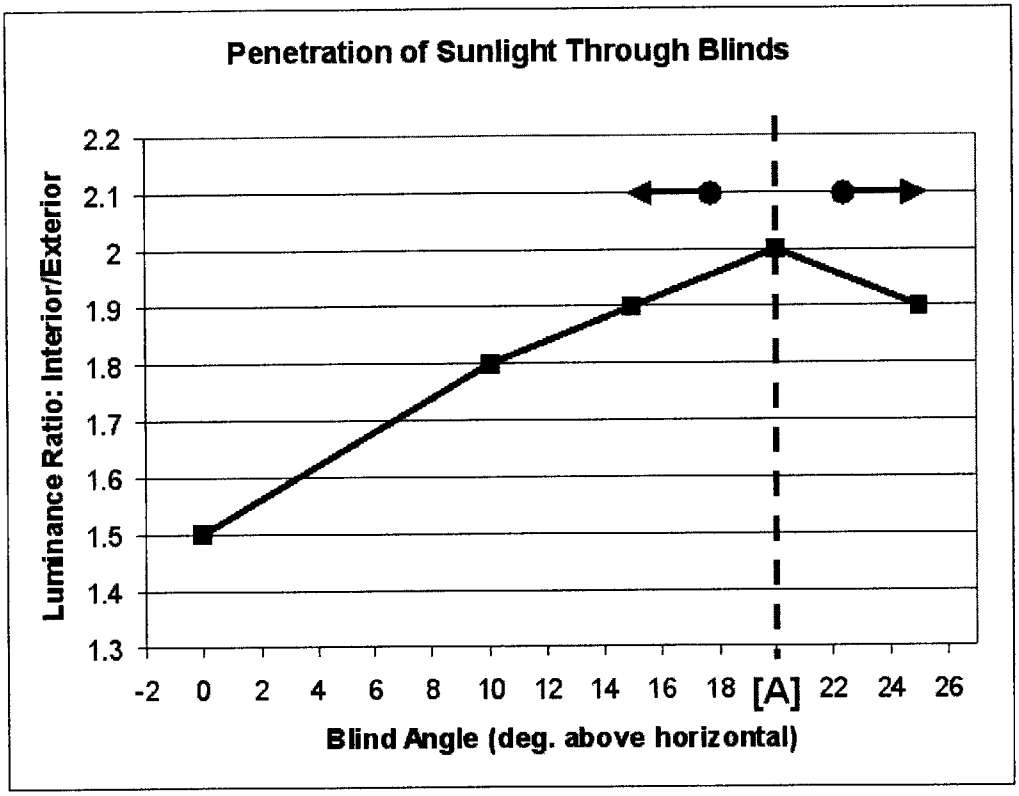


Fig. 30. The light intensity inside a room as a function of blind angle, using venetian blinds of high surface reflectivity. If the optimization begins using a scenario with a blind angle less than 20 degrees, it will move the angle to the left to achieve lower cooling loads. If it begins above the 20-degree threshold, the optimization moves to the right to achieve lower cooling (data excerpted from McGuire[13]).

In view of this nonconvex, nonlinear behavior in the building model, a method of optimization different from the gradient search is called for. The problem of finding a global minimum energy configuration is complicated by the nature of the input variables in the building problem. The variables represent a mixture of continuously-valued quantities and discrete ones such as compass directions. In moving from West to North, or North to East, we have no idea whether we are "increasing" or "decreasing" the value of the input variable. Finding that a north-facing building façade produces a lower-energy result than an west-facing, we would not be able to infer the likelihood of further

improvements from moving to an east-facing configuration. We must simply try all the different orientations and then compare the results, and the optimization procedure becomes a challenge of controlling the randomness in the choice of inputs to promote lower-energy performance.

Optimization procedures that employ randomness to explore the input space of a function are called *heuristic*. They differ from *deterministic* search procedures in the sense that the steps followed to arrive at an optimum set of inputs are not guaranteed to be the same each time the algorithm runs. We wish to apply such a heuristic to the estimator that we are using as a surrogate for the building simulator. In the discussion that follows, when we refer to an instance of "evaluating" of the objective function, we mean that the entire decision hierarchy of our estimator – the cascade of many successive binary decisions that ultimately places the probable output within a narrowly bounded range of values – will be invoked for each evaluation. A complete run of a heuristic process often involves thousands of evaluations, so the speed with which we can navigate the tree of binary decision points is critical to the performance of the optimization. The optimum value provided by a given run of a heuristic process will not necessarily be reproduced by subsequent runs, although all such "optima" will all lie within a certain distance of the true optimum with a certain probability. The three heuristic search procedures that were examined for this project were the *Genetic Algorithm*, the *Particle Swarm Optimization*, and the *Simulated Annealing Algorithm*.

Genetic Algorithm and Particle Swarm

Like gradient search methods, heuristics are trial-and-error calculations. By repeated tests of the output corresponding to a particular set of inputs, we gather information about not only the performance of the algorithm and the rate of improvement in input choice, but also the likely direction within the input space in which further improvements will be found. The idea of a Genetic Algorithm is to code some prediction of likely future improvements into the present solution. It accomplishes this in a manner similar to a population of living organisms striving to deal with a stress in their environment. Just as stronger individuals in a group of wild animals often win precedence in the competition for healthy mates, populations of "solutions" (proposed sets of input choices) in a genetic algorithm are compared against each other on the basis of the output value they produce. Solutions judged to be stronger on this basis will be mated - averaged or otherwise combined - with other strong choices, and the "offspring" of such pairs of strong individuals are then given extra representation in the next candidate pool. With each passing generation, the "genes" expressed in the pool of surviving combinations of inputs align ever more closely with the globally optimal solution, whose output value is the lowest of all. Random selection provides the combinations of inputs in the starting population, and perturbs the variable settings of certain individuals during mating to simulate mutation and provide better coverage of the input space. The Particle Swarm optimization exercises the wild animal metaphor in a different sense. This time, the various trial solutions are like birds in a loosely organized migration. The entire group of solutions follows a path through the input space that they trace out in parallel, evaluating each new set of output values after an interval of movement. Each individual adjusts its

own movement to match the bulk movement of the swarm, but it will also be pulled somewhat in a direction that it personally evaluates as constructive. For example, if the swarm is moving in direction y, but then shifts direction to move in the x-direction, an individual agent within the swarm will tend to continue somewhat in the y-direction if it has noticed its local output value increasing due to movement in that direction (Fig. 31). As with the Genetic Algorithm, the practitioner will usually look for the many individuals in a population to begin to cluster around the same ideal combination of input values, thereby signaling the end of the experiment.

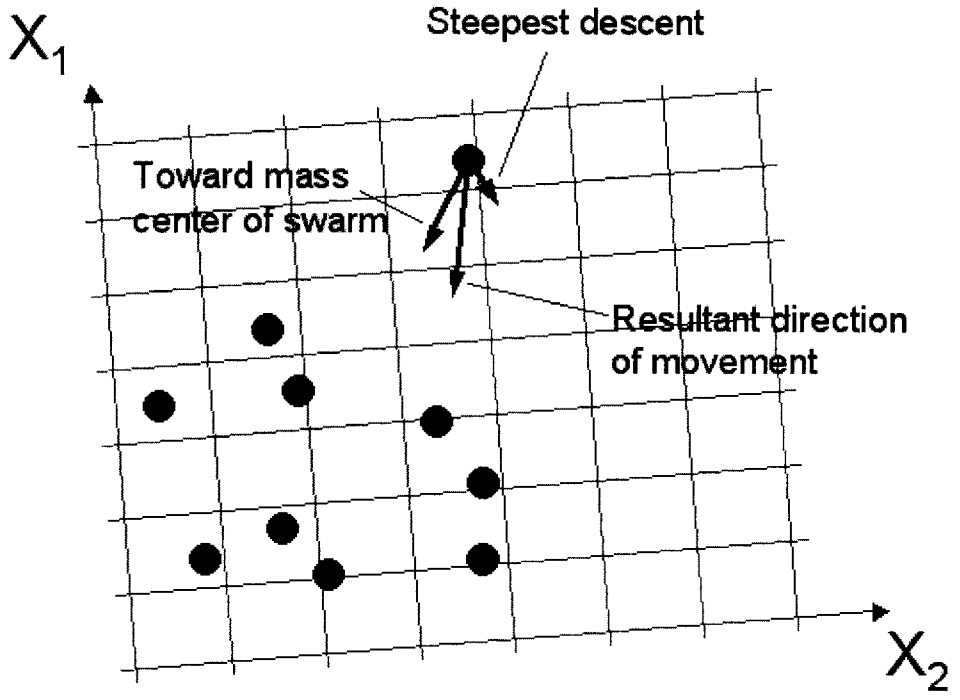


Fig. 31. Direction of movement of a single particle within the input space, in a Particle Swarm Optimization. The movement to a new sample point is influenced both by the value of the gradient at the present position (steepest descent) and by the direction in which the center of mass of the rest of the swarm lies. The "momentum" of a swarm of particles often serves to prevent individual agents from becoming trapped in local minima.

Simulated Annealing

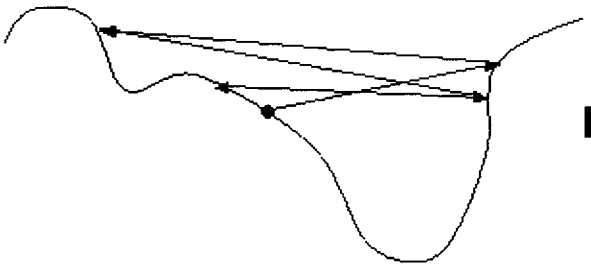
Genetic Algorithms (GA) and Particle Swarm Optimizations (PSO) can demonstrate an interesting variety of ways to approach a minimum energy building configuration, and may include optimal designs in their candidate pools that resemble each other very little in their input values. They are especially useful when we desire to understand the trade-off between different variable quantities – how much of x we must sacrifice if we wish to increase y and still keep our output in the neighborhood of the global optimum. On the other hand, both methods have the disadvantage that they are computationally expensive when compared with analytical methods such as the gradient search. The number of evaluation steps must be multiplied by the number of individuals in the population to give the total number of function evaluations required for the optimization. By contrast, the method of Simulated Annealing uses only one function evaluation per iteration of the search algorithm. Because our simulation software is run through an online interface, we have placed a high priority on the speed of our optimization procedure, and do not immediately wish to present an array of trade-off comparisons to the user of the Design Advisor website, although that may be a direction for future development. In the short term, our software delivers a single optimized building design that minimizes energy consumption within the user's design thresholds. The Simulated Annealing (SA) is conceptually simple compared with the other heuristics discussed above. The procedure is based on the idea of a "random walk" through the input space (Fig. 32a). Beginning with an initial guess, the input parameters are varied randomly and then re-evaluated. A record is kept of the lowest function value yet encountered, and given an unlimited

number of iterations, we will eventually find the globally optimal value by accident. This kind of optimization takes the opposite approach to the gradient search (Fig. 32b), which hones in very rapidly on a locally superior solution, but lacks the comprehensive coverage of the random walk, forever ignoring the true global optimum if it becomes trapped in the vicinity of a local minimum. To try to combine the comprehensiveness of the random walk and the efficiency of the gradient search, the Annealing algorithm begins to search the input space in a highly random way, and progressively reduces the randomness to the point at which the process becomes purely a gradient search (Fig. 32). Equation (15) describes the dynamics of this evolution from random walk to gradient search:

$$P = e^{\left(\frac{E_{old} - E_{new}}{T}\right)} \qquad (15)$$

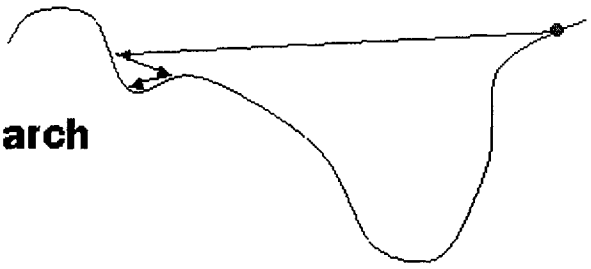
P represents the probability of moving from a certain point in the input space to an adjacent point whose corresponding output E has a greater, and therefore worse, value. Adjacent points whose outputs are smaller are accepted unconditionally.

a.)



Random Walk

b.)



Gradient Search

Fig. 32. The two conceptual bases of the Simulated Annealing algorithm.

The parameter *T* is analogous to the temperature of a bar of iron during a metallurgical annealing process. When the bar is at a high temperature, its constituent atoms are relatively free to assume a wide range of energy levels. An atom will spontaneously enter a more energetic state with high probability. As the bar cools, the atoms become increasingly less able to assume states of higher energy, and will tend to progress only to lower energy levels. If the bar begins at a high temperature and is then flash-frozen, it can become *tempered*, a condition in which permanent interatomic stresses develop in the material because the atoms are not energetic enough to escape local energy wells and find lower-energy lattice configurations that relieve the stress. If the bar is instead cooled very slowly, the atoms tend to explore the vicinity of the lowest-energy positions through the accidents of sustained random motion, then settle into a final, unstressed state when

the temperature finally becomes low enough to confine them to their local wells. In our application the temperature can be thought of as the degree of liberty given to the algorithm to explore new locations within the input space, regardless of possible increases in the output value corresponding to that new location.

The ability of the system to reach its minimum possible energy state depends on the design of the cooling strategy. In (15) this takes the form of the function $T = f(e)$, where e is the degree of the completion of the annealing operation. If f is a steeply decreasing function, the process resembles flash-freezing; the search allows movement in the input space only to coordinates where lower output values are found, and concludes quickly like a gradient search. If a shallow slope is chosen for the function f, the algorithm will allow the exploration of inferior solutions to persist for longer, and the "random walk" phase of the experiment will provide more coverage of the input space before the search is confined to choose only better-valued solutions. If the temperature iron bar in our analogy could be lowered by infinitesimally small increments over an infinitely long period, the bar could be said to be fully annealed, all atoms having found their lowest possible energy states. In any real annealing problem, we must settle for a compromise between the duration of the experiment and the robustness of our final minimum-energy configuration. We use a temperature program $T_{i+1} = cT_i$, where c is a constant between 0 and 1, to produce a cooling schedule that approaches an absolute zero or fully "frozen" condition asymptotically. The experiment is allowed to come to an "equilibrium" condition at each temperature step, reached when the random perturbations cease to improve the best recorded output value at the given temperature. Then the

temperature is lowered by one step, where it remains until an equilibrium has been established at the new temperature. The experiment is finally stopped when the lowest recorded output value ceases to change with further reductions in temperature.

STRUCTURE OF THE COMPUTER PROGRAM

Functional Modules and Their Responsibilities

The Linear Discriminant

The library of building cases used to train the estimator program is handled by a class called *BroadTooth*. The name *BroadTooth* distinguishes this class, which is concerned only with implementing the Fisher Discriminant, from a separate class called *FineTooth*, which invokes the membrane technique to improve on the output from *BroadTooth*. Both classes are recursive, in the sense that they are capable of dividing a given library of data in half (along the median output value), determining the vector that optimally separates the two halves in the input space, and creating two new copies of themselves to process each of the two halves as separate datasets. We stop creating new copies of *BroadTooth* at the point when a sufficient number exist at the lowest level of the tree to provide the desired resolution for the overall classification of data. At the conclusion of the exercise, we are left with a decision tree consisting of a Fisher Discriminant at each branch point that can linearly separate input data into "high" and "low" categories. This process is illustrated in the block diagram in Fig. 33.

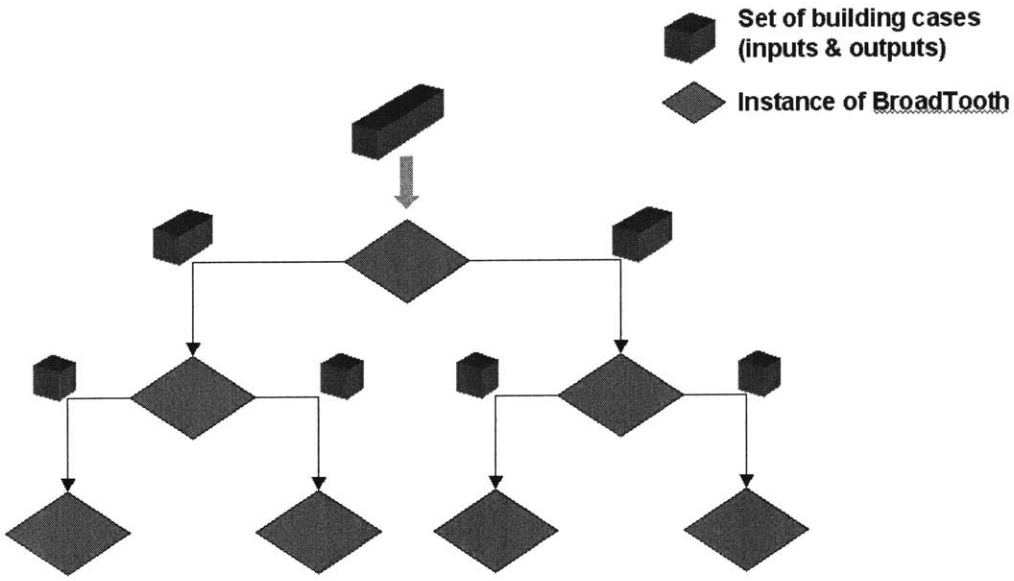


Fig. 33. The creation of a decision tree. Each BroadTooth object accepts a block of building data, separates the training data into two groups according to output value, and passes each group to a new copy of itself, which will further subdivide the data.

The estimator can function at the most basic level using only the tree of *BroadTooth* instances. New input points, for which we would like to estimate the likely output value, cascade through the tree structure in the same fashion as a case from the training data. At each decision point the new input encounters in the tree, it will be classed either to the left or the right, depending on the discriminant vector calculated for that decision point by the corresponding instance of *BroadTooth*. The numerical output value can be estimated from the limits on the range of values that belong in the chosen category at the bottom of the tree, as in Fig. 34. It will be remarked that the range of output values for a given category is not necessarily proportional to the number of library cases attributed to

each – the number of library cases is the same for all categories because each group is divided along the median output value.

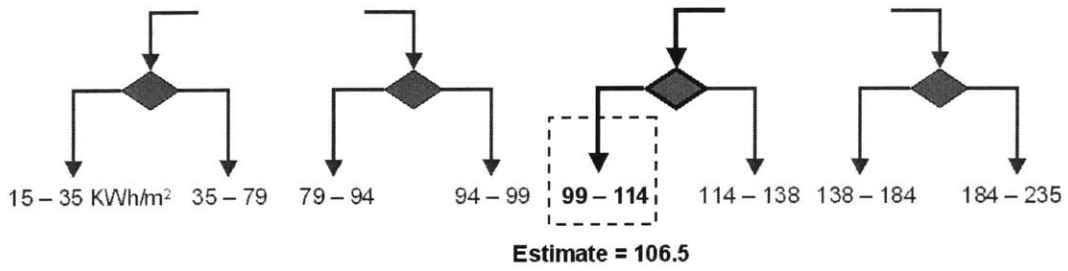


Fig. 34. A classification tree terminating in 8 distinct categories. The limits ascribed to the fifth category (99-114 KWh/m^2 in this example) subtend the range of training case output values between the 50th and 62.5th percentile (1/8th) of the library. If a new input is classed in one of these categories, its likely output value is estimated as the average of the range limits of the category.

Each time an instance of *BroadTooth* is called on to provide the vector that optimally separates data into two groups, it must perform two basic operations:

1. Enquire which set of n input variables out of the total of m have the greatest influence on the separability of the two groups, and
2. Find the optimal vector w corresponding to the particular set of n inputs that was chosen.

A class called *Tree* is responsible for providing both of these pieces of information (Fig. 35). It is desirable to use only those input variables that most significantly influence the output value, because insignificant inputs can add noise and ambiguity to the membrane procedure. Simply performing a sensitivity analysis on each variable will not reveal effects on the output that emerge from the interactions of two or more variables. Instead, we determine the degree to which library cases may be distinguished by output value

from a knowledge of their position in a space consisting of selected subsets of the input variables.

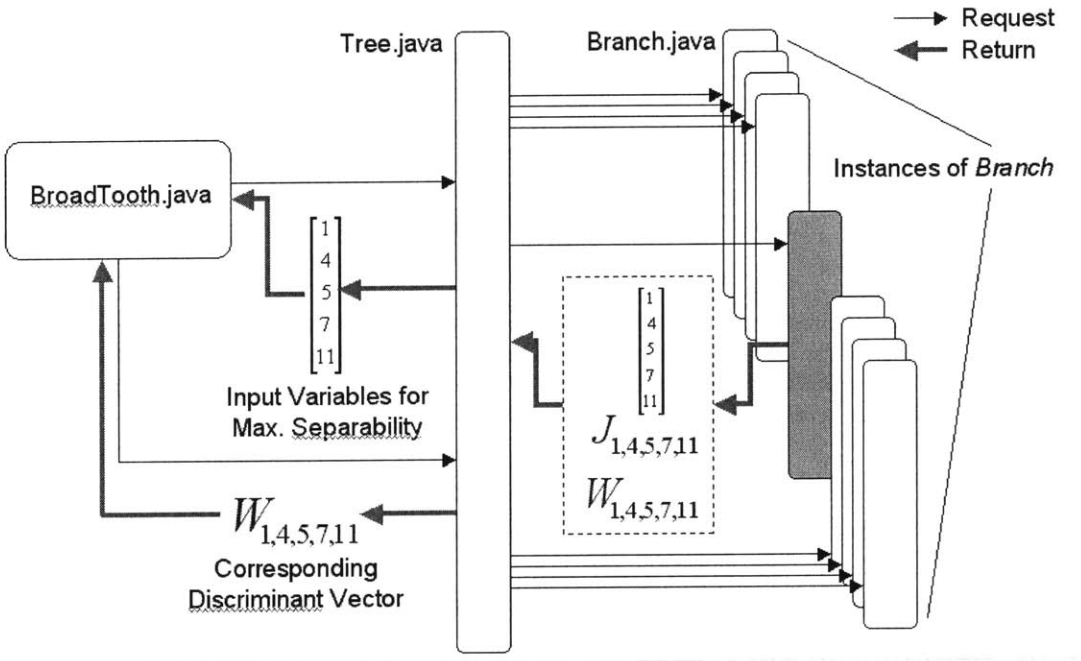


Fig. 35. Each instance of *BroadTooth* finds an optimal vector to separate its dataset into 2 groups using the class *Tree*. *Tree* establishes a different instance of *Branch* to explore each permutation of input variables – in this example, a subset of 5 variables has been requested. The permutation giving the strongest separability *J* is returned to *BroadTooth* along with its optimal Discriminant vector, *W*.

The measure of our power to distinguish output values based on selected input variables is given by the separability criterion *J*, which has been derived in detail in the chapter called *Linear Methods*. Briefly put, *J* is the ratio of the convolution of the *W*-vector with the *between-class* covariance matrix to the convolution of the *W*-vector with the *within-class* covariance matrix for a given set of cases, when only a certain subset of input variables are considered as the basis of the input space. To find the particular

combination of n input variables that gives the greatest value of J, we might expect that the separability criterion would need to be checked for every possible permutation containing n members. That would imply a number of checks equal to

$$\frac{m!}{(m-n)!n!} \qquad (16)$$

where m is the total number of input variables available to choose from. If we elect to use 30 variables out of a possible 40, expression (16) evaluates to 848 million checks. This calculation is performed during the training phase of the program and therefore does not delay the process for a user of the website. On the other hand, it represents about a week of processor time to calculate variable selections for an entire hierarchy of *BroadTooth* instances like the one shown in Fig. 33. The optimal selection of 30 variables will change with the level of the hierarchy, and even with different areas of the input space represented by *BroadTooth* instances that lie on the same level. Fortunately, we can use a simplifying principle that substantially reduces the required processing time.

Our definition of the separability criterion J satisfies the *monotonicity relation*[6]:

$$J(X^{+}) \geq J(X) \qquad (17)$$

"where X denotes a set of [variables], and X denotes a larger set of [variables] which contains the set X as a subset[7]." This means that the separability of a given set of cases will always be greater, the more dimensions are added to the input space in which we calculate the Fisher Discriminant. Accordingly, if we find any permutation X of 31 or

[6] Bishop0, pg. 108: "In reducing the dimensionality of the data we are discarding information, and this cannot reduce (and will typically increase) the theoretical minimum achievable error rate." A set of variables necessarily possess more power to resolve the output value than any subset of those variables.
[7] Ibid., pg. 305.

more variables that has a lower separability than some other permutation Y of subset of only 30 of those 31 variables, we can discount all subsets X' of X that contain 30 variables, since by transitivity,

$$J(X') < J(X) < J(Y) \qquad (18)$$

The class *Tree* uses this reasoning to reduce the number of instances of *Branch* that it must create (originally one for each possible combination of 30 variables). The process follows the pattern shown in Fig. 36: say we wish to choose 2 input variables out of a total of 5 to characterize the output from a function. We begin by choosing the first variable that will be excluded from consideration – represented by the first level of the tree diagram in Fig. 36. The *Tree* class creates an instance of *Branch* with an instruction to calculate the value of J for the complete set of variables minus the one chosen for exclusion. The *Branch* instance then recurses, creating a new instance of *Branch* from which the first plus an additional variable are excluded. This *Branch*, in turn, calculates the corresponding value of J and recurses again, and so forth until a *Branch* instance has been created in which all the variables save 2 have been excluded from the analysis. This terminal *Branch* corresponds to the point A in Fig. 36. The class *Tree*, which manages the process from above, now stores the value of J returned by the terminal branch and begins another chain of *Branch* objects, this time specifying a different variable as the first to be excluded. At any junction in the heirarchy, each *Branch* object may spawn several new *Branches* at the next level down to explore all possible permutations with one less variable. At each of the junctions in Fig. 36, the index of the variable excluded is indicated next to the corresponding branch point. However, because the value of J for any new *Branch* is immediately reported to *Tree*, it can immediately close any avenue of

exploration if the root *Branch* returns a value of J that is lower than the current minimum J for a terminal *Branch*. Such a halt on the recursion process is represented in Fig. 36 by the point B. Although we have only explored the hierarchy down to the first level, we already know after excluding only one variable that the direction is not promising. The monotonicity relation dictates that no value of J that we calculate at any point below B will be as high as the J at the point B itself. If J is already lower at B than for the terminal point at A, we can safely overlook all the terminal points that emanate from B.

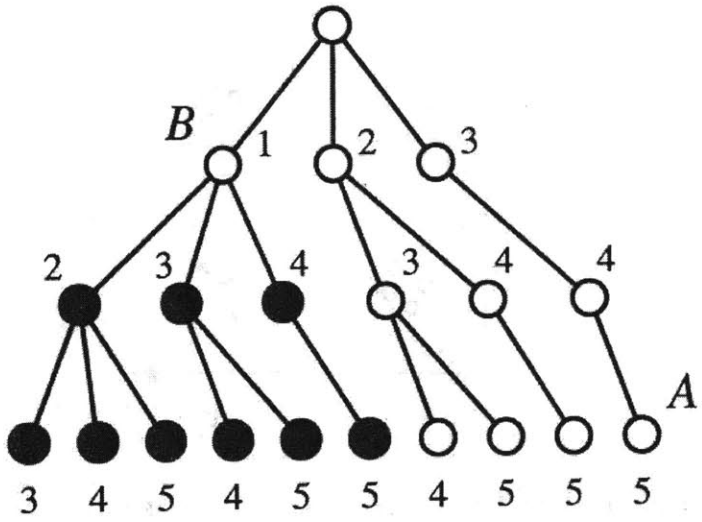


Fig. 36. (graphic credit: C. Bishop)

DESIGN OF A USER INTERFACE

Graphical Layout of the User Interface

The practical "last step" in the design of the classifier package has been to give the software an internet presence so that it can directly interact with the existing modules of the Design Advisor website. This purpose-built HTML interface has been written to accept information from building scenarios that users have already created using the other tools of the site.

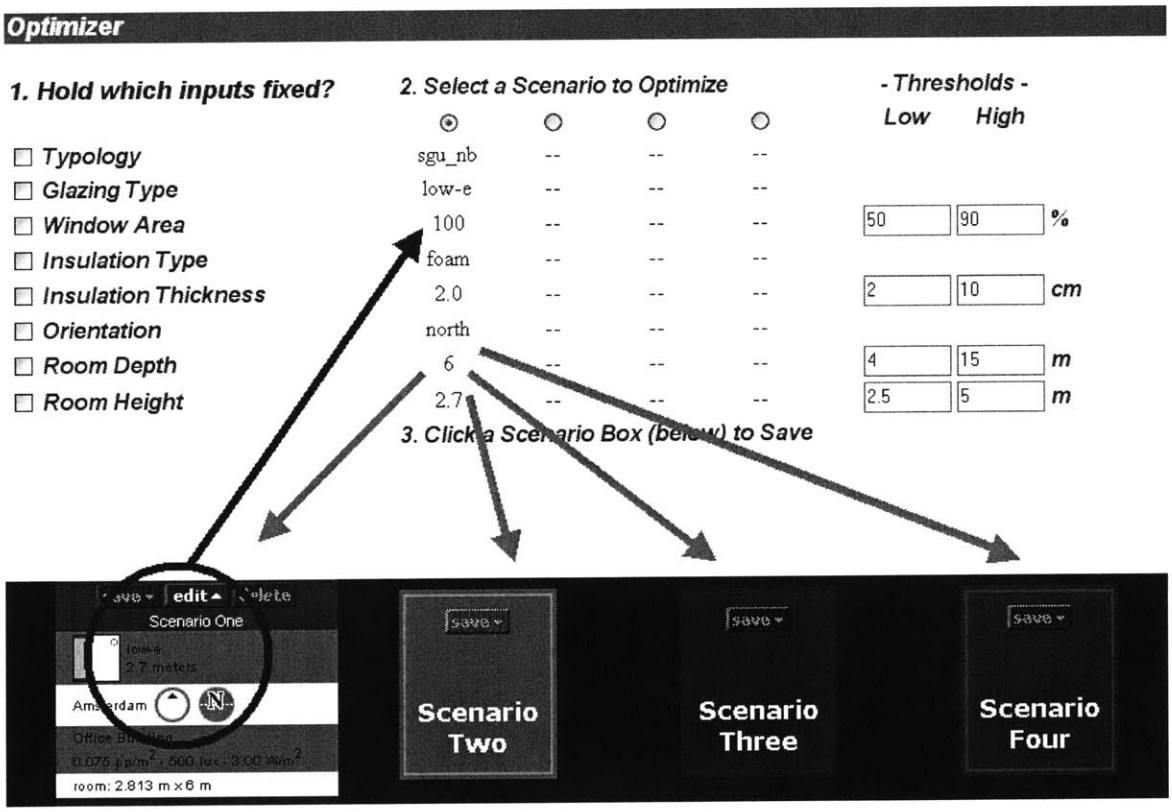


Fig. 37

Previously saved building scenarios are represented in the website by tabs at the bottom of the screen, which show selected information about the building design such as window type and plan area. When the user enters the site's optimizer mode, that information is copied and listed in a table under its scenario number. By selecting the appropriate column of this table, the user indicates which pre-saved scenario he would like to optimize, and then clicks one of the occupied or unoccupied tabs to choose a cell in which to store the newly optimized scenario. After a short delay, the parameters of the optimized building appear in the selected tab. The optimized building scenario functions in the same way as a manually entered scenario; its energy consumption, comfort rating, and daylighting properties can be viewed, and it can be sent to the Setup page where the user can adjust the values of the parameters and re-save.

User Control of Optimization

Among the roughly 40 different design variables that influence the output of the building simulation, only a small subset are subject to control by the user during the optimization process. The variables that the user is permitted to hold fixed or otherwise constrain are indicated in Table 2, along with some others that are either permanently fixed during optimization or always free to vary. Those variables whose participation the user does not control (marked as "fixed" and "free" in the table) are so designated because it is assumed that the user would not wish to have that control. For example, once the user has designated the city in which the proposed building will be situated, it is most unlikely that he would wish to find a new, more optimal city to put the building in. Conversely, there is little enough reason to expect that a designer would insist on a particular width

for the blind slats that we have allowed this parameter to vary without consulting the user.

Variable	Typical Settings	User Option (X)	
		Can Fix	Can Limit
Window typology	Single glazed, double-skin facade	X	
Glazing type	Clear, blue-tinted, low-e glass	X	
Window area	65% of facade area glazed	X	X
Insulation type	Foam, fiberglass	X	
Insulation thickness	4cm	X	X
Room orientation	South-facing	X	
Room depth	8m	X	X
Room height	3m	X	X
Room width	3m	Fixed	
Percent overhang	20% of window height	X	X
Minimum light level	400 lux	Fixed	
Location	Los Angeles, London, Cairo	Fixed	
Lighting system	Variable bulb brightness	X	
Type of ventilation	Mechanical (sealed), Hybrid	X	
Occupancy	.075 persons/m^2	Fixed	
Equipment	3 W/ m^2	Fixed	
Air exchange rate	0.5 L/person/sec	Fixed	
Blind angle when closed	45 degrees	Free	
Blind width	3cm	Free	
Blind color	Shiny aluminum, white plastic	X	
DSF cavity depth*	10cm	X	X
DSF cavity flow rate*	50 L/hr.	Fixed	
DSF cavity source*	Interior	Free	
DSF cavity extract*	Exterior	Free	

Table 2

Error Checking Procedure

The estimator achieves an average accuracy of 90% (implying an average error of 10% of the true output) when benchmarked against the full building simulation used by the Design Advisor. We use the estimator to calculate the output for each input vector generated during the annealing process, and to verify at the end of that process that we

have found outputs that cannot be improved by further exploration of the input space. Because the estimator does not replicate the behavior of the full simulation perfectly, it is impossible to say with complete certainty that the best input vector indicated by the estimator is actually best overall. If the building performance function is highly nonlinear, many local minima, characterized by widely varying input vectors, will be detected. An accuracy of 90% may not be sufficient to distinguish one minimum from another with great certainty. To provide for this eventuality, we use the original building simulation to verify the 5 best cases returned by the estimator. Each run of the original model requires as much time to complete as 1000 runs of the estimator, so the price in computation is high for the final verification procedure.

Further Accuracy Improvements

The library of cases used to train the estimator was constructed using inputs generated randomly between agreed reasonable limits. The vast majority of cases produce outputs with a value between 0 and 1500 KWh per m^2 of building floor area. Certain input limits, labeled "can limit" in Table 2, can be adjusted by the user to constrain the optimization. To constrain the process further, particular inputs may be held fixed during optimization if the user finds that certain aspects of his building design are not negotiable. We generally observe a significant increase in the minimum possible energy consumption as constraints are added, but the magnitude of this effect is smaller if the starting configuration is already a relatively low-energy building. The resolution of the estimator may not be sufficient to give accurate predictions of these marginal improvements. Whereas the estimator might move an inefficient building by a distance

of 20 or 30 output slots (roughly 600 KWh/m^2) to an optimal configuration, an already low-energy building may only move by a distance of 1 or 2 (from 20-50 KWh/m^2). Additionally, the superior cases found by the annealing algorithm, though quite different in their specific combinations of input values, may all be attributed to the same output slot. In other words, they may be resolved into the same range of output values at the lowest level of the binary classification tree. In this case, the annealing algorithm will not be able to distinguish which one is best.

The problem of discerning improvements in input configurations and comparing the available improvements against each other becomes difficult near the bottom of the range of possible output values. We have addressed this issue by creating a special, second-pass estimator that applies only to the cases that belong in the lowest range of output values. The library on which this estimator is trained before being deployed is restricted to cases having total energy consumptions within the lowest bracket of the original estimator's predictive range. Any new case assigned to this lowest slot by the original estimator will then be re-evaluated by the new low-output-only estimator. Fig. 38 shows one example of a decision path through the estimator that invokes this second estimator.

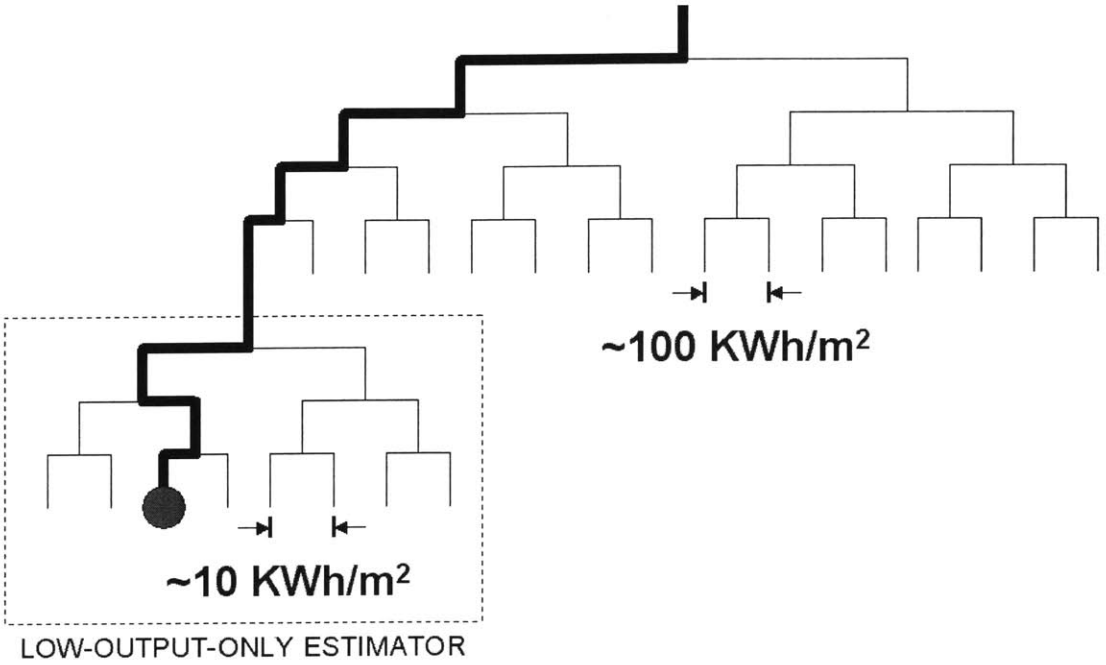


Fig. 38. The branching discriminant tree has recourse to a secondary tree to increase the resolution of small output predictions.

RESULTS 1: ACCURACY OF THE ESTIMATOR

Accuracy Criterion

We have measured the accuracy of the estimator by testing it on 3000 randomly generated scenarios. In each scenario, the imputs have been randomized independently. We predict an output value E' for each of these scenarios, and calculate a "true" output E in parallel using the full building model. The overall error can then be expressed as a normalized standard deviation σ:

$$\sigma^2 = \frac{\sum_i^n \left(\frac{E_i - E'_i}{E_i} \right)^2}{n-1} \qquad (19)$$

where n = 3000 scenarios.

Performance Statistics

The library data used to train the estimator conforms to the input range limits specified in Table 3, below:

Variable	Min.	Max.
Room depth (m)	4	15
Room height (m)	2.5	5
Room width (m)	3	10
Sill height (% of window height)	5	25
Overhang depth (% of window height)	1	30
Minimum allowable light level (lux)	0	1000
Angle of blinds when "closed" (deg.)	0	90
Depth of window cavity (m, double-skin façade typologies only)	0.1	0.2
Rate of airflow through cavity (kg/hr.)	30	90
Conductivity of insulation (W/mK)	0.02	0.04

Thickness of insulation (m)	0.02	0.1
Density of occupants (people/m^2)	0.025	2
Density of equipment (W/m^2)	0	15
Air-change rate (L/s/occupant)	7.5	40
Blind width (m)	0.01	0.05
Blind emissivity	0.1	0.9
Blind absorptivity	0.1	0.9

Table 3. Limits on randomly generated input variables in the training library.

As a baseline for the accuracy study we use a process that selects output values randomly from the feasible range of building model outputs – between 10 and 1500 KWh per square meter of floor area. This range accounts for roughly 99% of outputs from full building simulations that conform to the input limits in Table 3. A random process is the lowest-quality estimate possible, since it is completely independent of the quantity being estimated. By performing a numerical simulation it can be shown that randomly generating solutions gives a normalized standard deviation of 6.8 (Fig. 39).

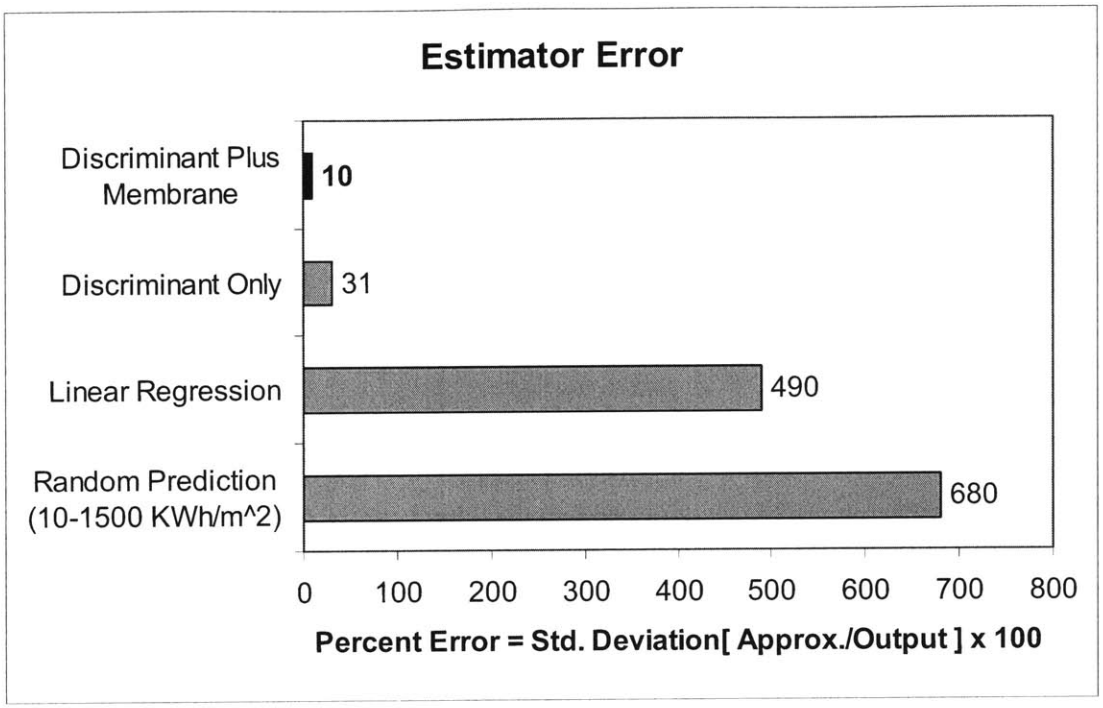


Fig. 39. Normalized standard deviations for the estimator with and without the membrane module, against a baseline error of 680% for the random (minimally accurate) case.

The difference between the error of the Linear Regression in Fig. 39, with a value of 490%, and the accuracy of the branching linear Discriminant method (31%) confirms the usefulness of a branching decision tree for nonlinear problems. The result labeled "Discriminant Only" is the error for an estimator using a hierarchy of dual-outcome decisions to reduce the range of the predicted output to a narrow range. The "Linear Regression" uses the same Fisher Discriminant technique as each branch of the decision tree, except that instead of using the projection onto the vector W to choose between two smaller output ranges, it measures the absolute magnitude of that projection to give an immediate prediction of the output value (Fig. 40).

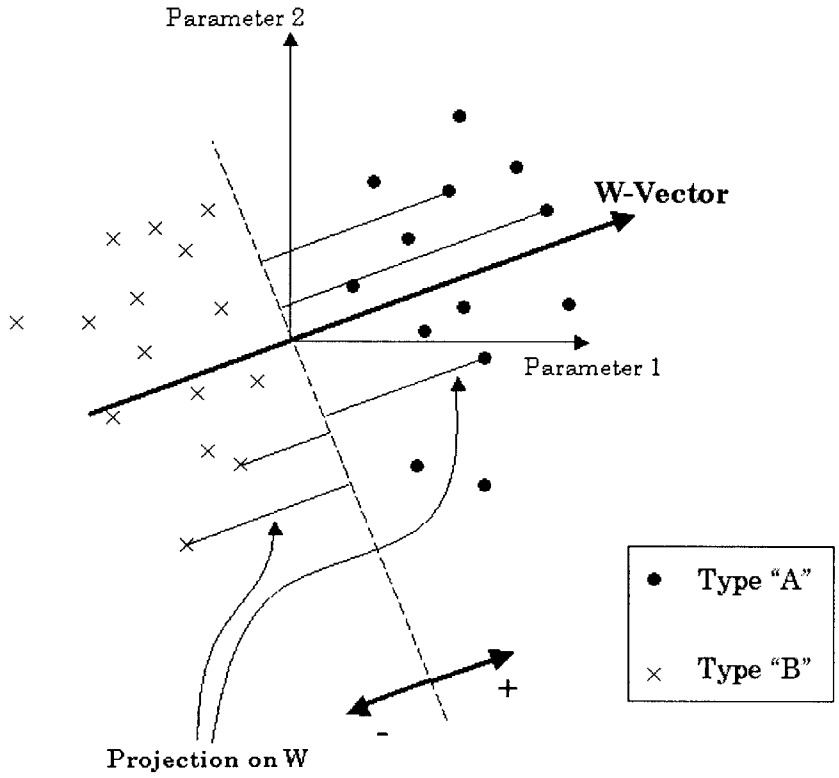


Fig. 40. Using the projection of an input point onto the W-Vector, we can either distinguish lower-than-median (Type A) inputs from higher-than-median (Type B) in a binary classification, or find a numerical estimate of the output directly by taking the magnitude of the projection itself, as in a basic linear regression.

The likely explanation for the large performance difference in Fig. 39 between the linear regression and the discriminant-based approach is that the building performance function is quite nonlinear. It is only by segregating the input space into small discrete sections, as in the branching method, that a linear analysis can succeed in identifying output values within a reasonable approximation. The uppermost item in the graph in Fig. 39 is the error measured in the predictions of the estimator in its final version. By adding the membrane feature to the discriminant algorithm, we reduce its overall error margin by a factor of 3.

Validation of Coding Strategies

Certain input variables not listed in Table 3 are *compound inputs*, meaning that by setting them to a particular value, we are effectively choosing a set of fixed numeric values to be represented by a single designation. All of the discrete input values are composed in this way. For example, if we choose the value "west" for the input describing the orientation of a building, this corresponds to the choice of a particular numeric coding that can be manipulated in the same way as other numeric values during the training and optimization phases. In the case of the orientation, we use 4 different numeric inputs to represent the four cardinal directions. For the purposes of the estimator, each orientation corresponds to a set of 3 "zeros" and 1 "one," with a different input taking the value "one" for each direction (Table 4).

	Input 1	Input 2	Input 3	Input 4
North	1	0	0	0
South	0	1	0	0
East	0	0	1	0
West	0	0	0	1

Table 4. Coding for the building orientation

The choice of the city in which to site the building is another example of a non-numeric input. The coding scheme used for the city input is more complex than the orientation, since many different kinds of information are associated with the location choice. As an approximation, we have used four numeric inputs to code the name of the city: average direct solar intensity, average diffuse solar intensity, average outdoor temperature, and latitude. Each possible choice of siting for a building corresponds to a particular combination of these four numeric descriptors. The weather files invoked by choosing a

city in our program contain hour-by-hour information on the first 3 of these, for a period of 1 year. This amounts to a true count of (1 year times 365 days times 24 hours times 3 parameters plus latitude equals) 26, 281 numeric inputs, but these are highly coupled to each other. We hypothesize that the total variability of building performance by city can be adequately represented using only 4 inputs. To test the validity of this assumption, we have compared the accuracy of the estimator for cities that appear in the training data against its accuracy for cities that are new, but which fall within the scope of the training data when only the 4 characterizing inputs are considered. If the 4 inputs were completely adequate for describing the city, there would be no difference in the accuracy when the estimator is applied to the new location. In actuality, of course, some information is lost in the operation of reducing the dimensionality of the city input from 26,281 to 4, but the additional error incurred is relatively small; from the baseline of 9.7, the predictive accuracy changes to 13.8 when the estimator handles cities that it has not yet encountered in the training data.

RESULTS 2: OPTIMAL TUNING OF THE ANNEALING ALGORITHM

Using the final version of the estimator, we can test the performance of hundreds of hypothetical buildings in the time it would take to perform a single run of the original building simulation. The estimator has been developed to serve as an extremely rapid surrogate for the building simulation – one that would allow an optimization to complete in only a few seconds. Using Simulated Annealing, excellent coverage of the input space can be achieved by testing about 1000 building configurations. We have adjusted the parameters of the annealing operation to allow the swiftest possible discovery of global optima without sacrificing thoroughness. As described in the chapter, *Real-Time Optimization with Simulated Annealing*, the probability of moving the search to a new point in the input space depends on "temperature" of the algorithm and the predicted output value of the building represented by the new input configuration. If the new output evaluates to a lower number than the previous point, the new point is accepted unconditionally. If the new output is higher (therefore *worse*), the new point is accepted with the following probability p:

$$P = e^{\left(\frac{E_{old} - E_{new}}{T}\right)} \qquad (20)$$

where E is the predicted output value and T is the monotonically decreasing temperature of the annealing process. We used a fitness criterion for judging the effectiveness of different suites of control parameters. Having chosen in advance a particular low-energy output value, we count the average number of function evaluations that an annealing

algorithm requires to find a scenario with an equal or lower value. Each suite of settings tested is used in 10 consecutive annealing experiments. We have found as a result of this test that the settings for the annealing parameters shown in Table 5 generally permit the most efficient searches for our building problem.

Starting T	500 KWh/m^2
Number of repetitions before temperature decrement	50
Number of repetitions at successive temperatures before termination	3
Temperature multiplier	0.2

Table 5. Best parameters for the Simulated Annealing segment.

Using these figures, it is possible to put bounds on the degree to which our algorithm will tolerate moving to a worse-performing input point, in the interest of covering the input space as completely as possible. As the algorithm begins, $T = 500$, which according to (20) is high enough that the algorithm will move to new input points that are worse than the initial point by an amount 350 KWh/m^2 (i.e. $E_{old} - E_{new} = -350$) with a probability $P = 0.5$. As the algorithm proceeds, we put an increasing priority on finding lower-valued outputs, and we become less interested in exploring the design space fully to avoid focusing on local minima. Accordingly, as the temperature is reduced, it becomes less likely that the algorithm will move to points with worse predicted outputs. After we have twice decreased the temperature by the *temperature decrement factor* of 0.2, the current temperature becomes 500 x 0.2 x 0.2 = 20. At this stage, $P = 0.5$ is the probability of moving to a point that is only worse by an amount 14 KWh/m^2. In our cooling schedule, the temperature continues to be reduced each time by the same factor until moving to worse ouputs becomes vanishingly unlikely (Fig. 41).

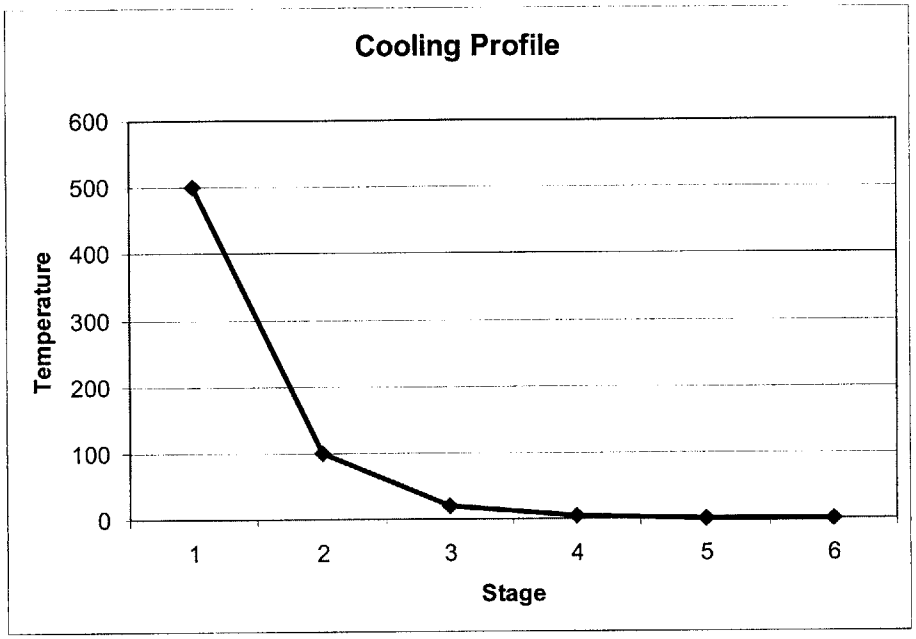


Fig. 41. The experiment temperature during an annealing operation can follow any number of profiles that end by letting the temperature approach zero. We have selected this exponential decay.

During the annealing process, we reduce the temperature by steps, waiting at each step for the algorithm to arrive at a point where the lowest output yet recorded at the current temperature is not beaten by an even lower output for a duration of 50 further new point-selections. The number of selections that pass without improving the output is referred to in Table 5 as the number of *repetitions*. At that point the temperature is reduced, and continues to be lowered by the same fraction until finally terminating when the experiment has recorded three repetitions at successive temperatures. The lowest recorded output from the experiment is then reported as the global optimum.

RESULTS 3: SENSITIVITY ANALYSIS FOR OPTIMAL BUILDINGS

The ability to generate building scenarios that are close to the global minimum for energy consumption in a given geographic region leads to the question of how robust these solutions are. If the output value changes dramatically in the vicinity of the optimal point for a relatively small change in one of the input variables, we can say that the output has a high sensitivity to that input. Generally high (>1) sensitivities for all variables usually indicate that the solution space is discontinuous or ill-conditioned. Fig. 39 shows that the sensitivities in our model not only vary greatly by variable, but also by the conditions under which an optimum was chosen. We have compared two optimizations: one for a building located in Edmonton, Canada, and another for Cairo, Egypt. As we might expect, the Edmonton building is more sensitive to variations in required minimum light level, since the intensity of daylight will be barely sufficient or insufficient to meet the lighting needs of an office room. On the other hand, the depth of the window overhang is much more critical in Cairo, where strong overhead sunlight can contribute greatly to the cooling load if not intercepted by shades.

It will be remarked that the output is more sensitive to the thickness of the wall insulation in Cairo than in Edmonton. This seems strange, considering that insulation plays a far more important role in a cold environment such as Edmonton, but the fact that a great deal of wall insulation is required in Edmonton is precisely the reason why the *optimal* output is less sensitive there. In locations where the optimal amount of wall insulation is large, a large amount will be selected by the optimizer, meaning that slightly more or less

than the considerable thickness chosen will have relatively little impact on the result. On the other hand, the optimizer chooses a very low thickness for Cairo, but a little more or less than the chosen value may make the difference between having some insulation and none at all; this difference would certainly have a strong influence on the output.

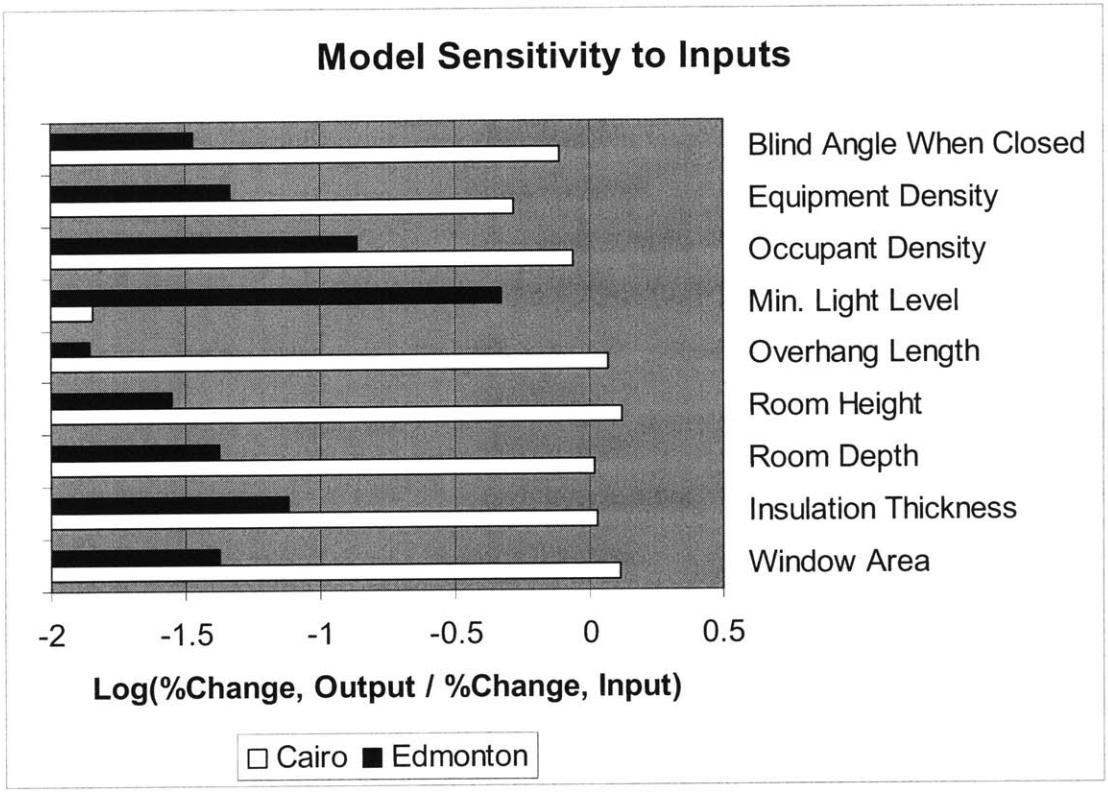


Fig. 42. Input sensitivity of the energy performance of buildings in the Design Advisor.

On the logarithmic scale of the graph in Fig. 42, a value of 0 on the horizontal axis indicates that changing a particular input produces a proportional change in output. That this should be true of any input variable in the vicinity of an optimal point suggests a sharp, discontinuous shape for the solution as a function of the inputs.

CONCLUSION

The MIT Design Advisor belongs to a group of software products that explore how architects' design strategies will affect the energy performance of their buildings. At this stage, none of these products is equipped to recommend improvements to a design; the trial-and-error guidance they provide is passive, and practitioners must find other means to intelligently guide them toward configurations that make the most efficient use of their site conditions. The optimizer package presented in this thesis is a first effort towards the automation of such an "intelligence." By the standard of an experienced practitioner of energy-efficient design, this tool is not "knowledgable," and for all the complexity of its searching algorithms, represents an approach to finding the lowest-energy design that is completely ignorant of the rules of good architectural practice. The optimizer does not teach a user why its recommendations lead to a more efficient building, it only presents a checklist of improvements to follow. Very often, the results derived from several optimizations of the same building produce entirely different recommendations. Yet this is also a strength of the program because, blind to the pedagogy of established energy-conscious design practice, it rates solutions solely on the basis of their bottom line – the energy savings they represent. There are often many different ways to reduce the energy consumption of a particular building, and they can contradict each other, as we should expect based on the nonlinear and ill-conditioned nature of the solution space in our own building simulator. The advantage offered by a "blind" software program is that avoids making many limiting assumptions that even the most experienced designers sometimes adopt. This optimizer is able not only to refine the architect's existing approach to his

design, but even to circumvent that approach entirely, suggesting radical re-designs that lead to the best building performance.

The purpose of the optimizer as a feature of the larger Design Advisor suite is to serve the original vision of the site: to provide reassurance that the most basic choices in the design of a building will not commit the client to a wasteful and expensive project. The remaining question is, "how do we know what is wasteful? Wasteful relative to what?" A building using a large amount of energy may actually be extremely efficient, given the function it needs to perform. The answer to the question is unique to each architect's own set of requirements. It is the question we have sought to address with the addition of an optimizer to the larger Design Advisor tool. Beyond the straightforward goal of providing recommendations of better designs is has the more important function of showing the limits of possible improvements.

REFERENCES

[1] "MIT Design Advisor" Website. Leon Glicksman, James Gouldstone, Matthew Lehar and Bryan Urban: © 2001 http://designadvisor.mit.edu

[2] "Past, Present, and Future of GMDH." A. G. Ivakhnenko. Published in *Self-Organizing Methods in Modeling: GMDH Type Algorithms*. Stanley J. Farlow, ed. Marcel Dekker, Inc. New York: 1984.

[3] "The GMDH Algorithm." Stanley J. Farlow. Published in *Self-Organizing Methods in Modeling: GMDH Type Algorithms*. Stanley J. Farlow, ed. Marcel Dekker, Inc. New York: 1984.

[4] "Combining Pairwise Classifiers with Stacking." Petr Savicky and Johannes Fürnkranz. Published in *Intelligent Data Analysis* (2003), pp. 219-229.

[5] "A Multiclass Classification Method Based on Multiple Pairwise Classifiers." Tomoyuki Hamamura, Hiroyuki Mizutani, and Bunpei Irie. Published in *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, January, 2003.

[6] "A Tutorial on Support Vector Machines for Pattern Recognition." C.J.C. Burges. Published in *Data Mining and Knowledge Discovery*, Volume 2, No. 2 (1988), pp. 1-47.

[7] *The Mathematical Theory of Communication*. Claude Shannon and Warren Weaver. University of Illinois Press. Urbana: 1963.

[8] "Decision-Making with a Fuzzy Preference Relation." S.A. Orlovsky. Published in *Fuzzy Sets and Systems 1*, 1978, pp. 155-167.

[9] "Fuzzy Sets." L.A. Zadeh. Published in *Information and Control*, June 1965, pp. 338-353.

[10] "Rating and Ranking of Multiple-Aspect Alternatives Using Fuzzy Sets." S. M. Baas and H. Kwakernaak. Published in *Automatica*, Volume 13 (1977), pp. 47-58.

[11] *Fuzzy Sets and Systems: Theory and Applications.* Didier Dubois and Henri Prade. Academic Press. New York: 1980.

[12] *Principles of Optimal Design: Modeling and Computation.* Panos Y. Papalambros and Douglass J. Wilde. Cambridge University Press. New York: 2000.

[13] *A System for Optimizing Interior Daylight Distribution Using Reflective Venetian Blinds with Independent Angle Control.* Molly McGuire, Masters Thesis in Building Technology at the Massachusetts Institute of Technology, 2005.

[14] *Neural Networks and Pattern Recognition.* Christopher M. Bishop. Oxford University Press. New York: 1995.