Improved Classification and Discrimination by Successive Hyperplane and Multi-Hyperplane Separation

Fred Glover and Marco Better
OptTek Systems, Inc.
2241 17th Street
Boulder, CO 80302

Abstract

We propose new models for classification and discrimination analysis based on hyperplane and multi-hyperplane separation models. Our models are augmented for greater effectiveness by a tree-based successive separation approach that can be implemented in conjunction with either linear programming or mixed integer programming formulations. Additional model robustness for classifying new points is achieved by incorporating a retrospective enhancement procedure. The resulting models and methods may be viewed from the perspective of support vector machines and supervised machine learning, although the new approaches produce regions and means of exploring them that are not encompassed by the procedures customarily applied. We focus primarily on two-group classification, but also identify how our approaches can be applied to classify points that lie in multiple groups.

Introduction

Let $A_i = (a_{i1}\ a_{i2}\ \ldots\ a_{in})$, $i \in G = \{1, 2, \ldots, m\}$ denote a collection of vectors whose elements belong to two groups, indexed by G_1 and G_2. We seek a classification rule to identify whether a given vector A should belong among the A_i for $i \in G_1$ or among those for $i \in G_2$. For example, the elements A_i may refer to people to be classified according to whether they have a particular disease ($i \in G_1$) or are free of the disease ($i \in G_2$), where the first component a_{i1} of A_i may refer to the person′s weight, the second component a_{i2} may refer to the person′s white cell count, and so forth. Common instances of classification problems come from the areas of finance, healthcare, engineering design, biotechnology, text analysis, homeland security and many other areas (see, e.g., Dai, 2004).

The decision rules we investigate are based on hyperplane separation approaches, viewing the A_i vectors as points in an n-dimensional space. In the simplest case, we seek a single hyperplane to differentiate the points A_i for $i \in G_1$ from the points for $i \in G_2$, where as nearly as possible the hyperplane will lie between the two sets of points, so that each group lies predominantly on one side of the hyperplane .

More generally, we identify conjunctions and disjunctions of hyperplanes to yield more complex regions for separating points of the two groups. The models are based on linear and mixed integer programming formulations, whose effectiveness is amplified by embedding them within a successive separation process that constitutes an iterative tree-based procedure. A key variation makes special use of a procedure called successive perfect separation that compels one of the two separating regions to contain all points of one of the groups at each branch, and further refines the outcomes by a process of retrospective enhancement.

The organization of this paper is as follows. We first review literature that provides the background for the new models, observing connections with ideas introduced in the context of support vector machines. A series of basic linear and integer programming formulations is introduced, proceeding from simpler to more advanced considerations. We then introduce an additional layer of refinement, as a foundation for greater practical efficacy, by coupling these models with successive separation procedures and the associated methods of retrospective enhancement. Finally, we propose new mixed integer optimization models and associated streamlined approaches for solving them that give another level of sophistication to the classification tools used in successive separation.

1. Linear Programming Models for Single Hyperplane Discrimination Analysis

To begin we review fundamental ideas underlying the creation of a single separating hyperplane[1] by linear programming (LP). Let $x = (x_1\ x_2\ \ldots\ x_n)$ denote a vector of weights,

[1] Since the hyperplane may not succeed in precisely separating the two groups, as where some points lie on the "wrong side" of the hyperplane, we use the word "separating" in a broad sense, as may alternately be conveyed by the term "quasi-separating."

one for each of the n components of a vector $A = (a_1\ a_2\ \ldots\ a_n)$, where A may represent one of the vectors A_i, $i \in G = \{1, \ldots, m\}$ or a new vector that we may wish to classify. The weights x_j of x are variables whose values we undertake to discover in order to produce the hyperplane. We also seek to determine the value of an additional variable, denoted by b, so that knowledge of x and b will permit the hyperplane to be represented by the equation Ax = b. Once the determination of x and b has been made, these variable quantities may be treated as constants, in order to test whether an arbitrary point $A = (a_1\ a_2\ \ldots\ a_n)$ in n-space lies on a given side of the hyperplane.

The condition that all A_i for $i \in G_1$ lie on one side of the hyperplane and all points A_i for $i \in G_2$ lie on the other may be expressed by writing

$$A_i x \leq b \quad \text{for } i \in G_1 \quad \text{and } A_i x \geq b \quad \text{for } i \in G_2.$$

We prefer if possible to avoid the case where points lie precisely on the hyperplane (by satisfying $A_i x = b$), and more generally prefer to have the hyperplane lie in a space that is "halfway between the two sets of points" in a meaningful sense. A point that is misclassified, by failing to lie in its targeted half-space, can be evaluated by reference to how far it lies from the hyperplane boundary, where the distance from the boundary (which is positive for misclassified points) is measured by $A_i x - b$ for $i \in G_1$ and by $b - A_i x$ for $i \in G_2$. By contrast, points that are correctly classified can be evaluated by reference to distance measures given by $b - A_i x$ for $i \in G_1$ and by $A_i x - b$ for $i \in G_2$, which are non-negative (and preferably positive) for these points.

The study of separating hyperplane models using linear programming was launched by the work of Mangasarian (1965), who introduced a model to minimize the greatest violation of any misclassified point by solving a collection of 2m linear programs. Subsequently Freed and Glover (1981) provided a series of LP models for hyperplane separation, each requiring the solution of only a single linear program and encompassing the goals of minimizing the greatest violation as well as minimizing weighted sums of violations. These models also handled the case of maximizing the minimum internal deviation and of maximizing a weighted sum of internal deviations. Currently all separating hyperplane models use variations of the form that relies on solving only a single linear program.

1.1 A Basic Linear Model Formulation

We start from one of the primary linear programming models of Freed and Glover, and then introduce refinements and generalizations provided by a more recent formulation of Glover (1990). We will call points that lie or fail to lie in their appropriate half-spaces, i.e., which are correctly or incorrectly classified, *satisfying* or *violating* points, respectively. Let s_i denote a variable that measures the amount by which a satisfying point A_i lies inside its associated half-space, and let v_i denote a variable that measures the amount by which a violating point lies outside this half-space. Figure 1 shows an example where hyperplane Ax = b attempts to separate the triangles from the squares. Point 1 is an element of the triangle group that satisfies its hyperplane constraint, while

point 2 violates it. By implication, s_i and v_i are non-negative and at most one member of each pair may be positive, as the figure shows.

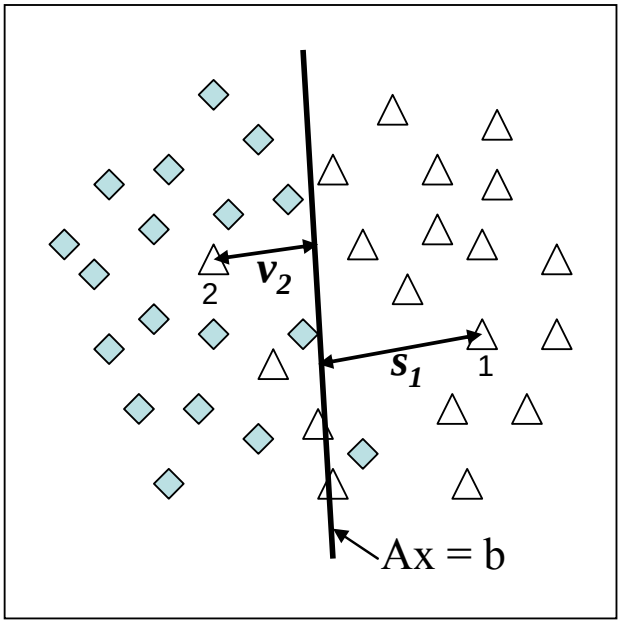

Figure 1: schematic representation of satisfying and violating measures

We incorporate these variables into the inequalities for the half-spaces to convert them into equations, by writing

$$A_I x - v_i + s_i = b, \quad i \in G_1$$
$$A_I x + v_i - s_i = b, \quad i \in G_2$$

The first objective we examine is to minimize a weighted sum of the violations, and subject to this, to maximize a weighted sum of the satisfactions. Let h_i denote a weight associated with the variable v_i to discourage it from being positive, and let k_i denote a weight associated with the variable s_i to encourage it to be a large as possible, in the event that $v_i = 0$. Finally, let $m_1 = |G_1|$ and $m_2 = |G_2|$. Then we obtain the formulation.

$$\text{Minimize } \sum (h_i v_i - k_i s_i : i \in G) \tag{1.1}$$
subject to
$$A_i x - v_i + s_i = b, \quad i \in G_1 \tag{1.2}$$
$$A_i x + v_i - s_i = b, \quad i \in G_2 \tag{1.3}$$
$$x, b \text{ unrestricted} \tag{1.4}$$
$$v_i, s_i \geq 0, \quad i \in G \tag{1.5}$$
$$(m_1 \sum (A_i : i \in G_2) - m_2 \sum (A_i : i \in G_1))x = 1 \tag{1.6a}$$
$$m_1 \sum (s_i - v_i : i \in G_2) + m_2 \sum (s_i - v_i : i \in G_1) = 1 \tag{1.6b}$$

Equations (1.6a) and (1.6b) from Glover (1990) are equivalent forms of a normalization constraint that plays a vital role in the formulation.[2] Usually, (1.6b) is more convenient to use than (1.6a) since it does not require computing the sum of the A_i vectors over $i \in G_1$ and $i \in G_2$, but (1.6a) may sometimes be appealing for having a form that is more nearly invariant over additional formulations we describe subsequently. In addition, we suggest that it can be useful to impose a constraint that achieves a *balanced violation condition* when the formulation (1) does not result in perfectly separating the two groups (hence some of the v_i variables receive positive values). Such a constraint may take the form

$$m_1 \sum (v_i: i \in G_2) = m_2 \sum (v_i: i \in G_1)$$

The constraint has no impact if a solution exists with all $v_i = 0$, but it can have some utility in respect to generating more robust separations.

1.2 Background of Normalizations and Common Uses of the Model

Various normalizations have been proposed through the years to assure that linear programming models for discrimination analysis do not admit the degenerate solution given by $x = 0$, $b = 0$ (which also yields $v_i = s_i = 0$ for all $i \in G$). However each of the normalizations introduced prior to those indicated in formulation (1) was discovered to introduce flaws into the model by failing to yield solutions that were invariant when the problem data undergoes transformations such as rotations or translations. The discovery of (1.6a) and (1.6b) finally removed these flaws, as proved in Glover (1990). Nevertheless, many researchers that use linear programming models for hyperplane separation continue to rely on flawed normalizations, apparently unaware of the deficiencies introduced.

To assure that the solution to formulation (1) is bounded for optimality, it is necessary to choose the coefficients h_i and k_i so that $h_i > k_i$, $i \in G$. Theoretically, the condition can be relaxed to stipulate that $h_i \geq k_i$, but this entails some risk computationally due to round-off error that can cause difficulties when choosing $h_i = k_i$. More particularly, a condition that has been proved to assure bounded optimality in the presence of the normalization constraint (1.6a) or (1.6b) is given by selecting the h_i and k_i values so that

$$\text{Min}(h_i: i \in G) > \text{Max}(k_i: i \in G).$$

Historically, nearly all adaptations of formulation (1) explored in various studies reported in the literature have focused on the special case where all $h_i = 1$ and all $k_i = 0$. In this instance the objective reduces to simply minimizing the sum of violations (sometimes called external deviations), and the model has popularly come to be known as the "Minimum Sum of Deviations" or MSD, model.

---

[2] The right hand side of 1 in these equations can be replaced by any positive constant, which has the outcome of scaling the solution.

It should be stressed, however, that the presence of weights h_i and k_i that differ from 1 and 0 affords several advantages. Among these is an opportunity to emphasize the correct classification of some points more strongly than others (by means of the h_i values) and to pursue a goal of driving some points to lie more deeply inside the half-space of correct classifications than others (by means of the k_i values). Such features can be valuable in applications where incorrectly classifying certain points may have more unfavorable consequences than incorrectly classifying others, or conversely, where correctly classifying some points can have a greater pay-off than correctly classifying others.

Allowing the h_i and k_i coefficients to take values other than the value 1 used in the simple MSD model also has the benefit of permitting these coefficients to be manipulated by means of linear programming post-optimality analysis. This makes it possible to change the emphasis on correctly classifying specific points or subsets of points, and to efficiently determine the effects of such changes. Figure 2.a) shows a hyperplane $Ax = b$ that is obtained by assigning equal weights to all points. In figure 2.b), we assume that it is more important to correctly classify the triangular points than the squares; therefore, the triangular points that are shaded are assigned a higher weight h_i than the other points, as a function of their proximity to the original hyperplane. The result is a new hyperplane $A'x=b'$ which now classifies two more of the triangles correctly, even though one more square is misclassified.

As noted in Glover (1990), such manipulations can also be used to diminish the effects of outliers, e.g., by reducing the size of their coefficients or setting them to 0. (Setting the coefficients to 0 is equivalent to dropping a point from consideration altogether. Such an approach of removing outliers has been proposed more recently in Ma and Cherkassky, 2005.) LP post-optimality procedures can also be used to generate new attributes as nonlinear functions of others. An efficient implementation results by pricing-out the associated new variables in a currently optimal linear programming basis, to identify at once whether these attributes are "profitable" in a linear programming sense and can thereby improve the problem objective by their inclusion (Barr and Glover, 1993, 1995). In this manner there is no need to generate or include such elements in the problem in advance, since they can be produced and evaluated on the fly.

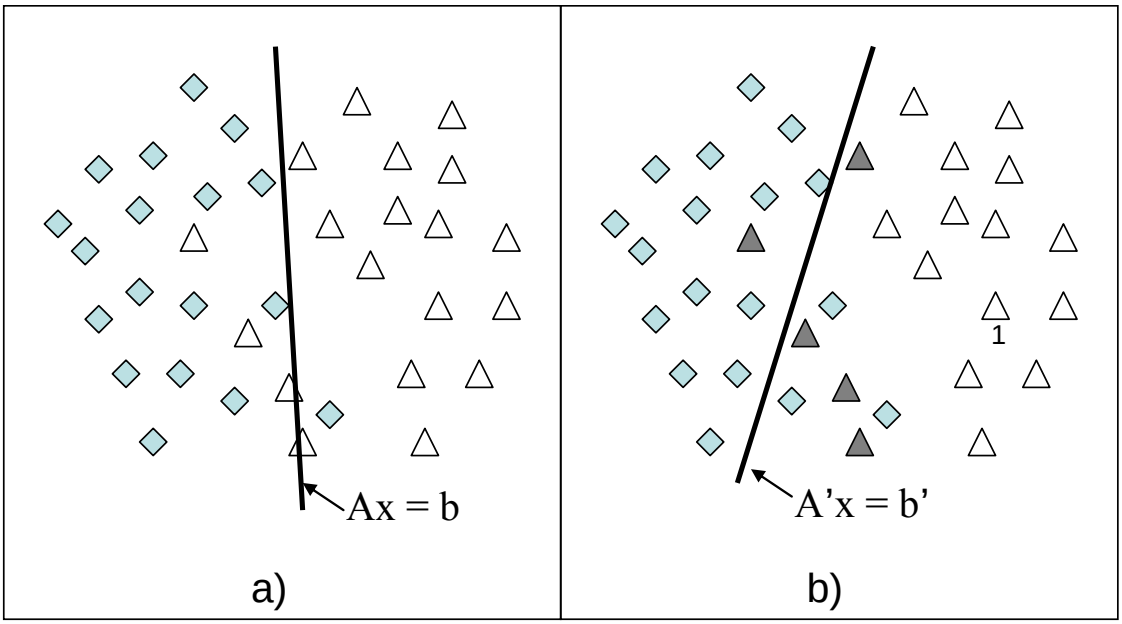

Figure 2: an example of the effects of post-optimality analysis

These types of approaches may be interpreted as belonging to the class called support vector machines (see, e.g., Christiani and Shawe-Taylor, 2000; Schlkopf and Smola, 2002; Wang 2005). Although they originated before the SVM classification and taxonomy was introduced, the LP post-optimization proposals are highly relevant to the kernel function notion that has been popularized in the SVM literature. The purpose of a kernel function, in particular, is to transform the problem data to give it a structure or form that is easier to classify. The outcome of the transformation produces new units of data (increasing the problem dimensionality) as a way to incorporate information implied by the original data, but not originally in a form that is amenable to be treated effectively by the analytical tool used to produce classifications. As can be seen, the proposals to augment the LP model using linear programming post-optimality analysis yield an adaptive method for processing the data to yield new data. Consequently, this affords a means for enhancing the repertoire of SVM kernel generating procedures without the need to rely on an a priori specification or dedication to a particular type of transformation, or to invoke all elements of the transformation at once.

The linear programming models for creating separating hyperplanes can be improved not only by differentially weighting the v_i and s_i variables and by incorporating a post-optimality component, but by additional elements introduced in subsequent sections. We lay the foundations for these improvements by examining natural extensions of the preceding model ideas.

1.3 A More General Linear Model

A simple extension of the model of Section 1.1 arises by introducing a *comprehensive violation* variable v_o relevant to minimizing the maximum violation over all violating points, and a *comprehensive satisfaction* variable s_o relevant to maximizing the minimum

satisfaction over all satisfying points. With these variables included, the formulation becomes

$$\text{Minimize } \sum (h_i v_i - k_i s_i : i \in G) + h_o v_o - k_o s_o \qquad (1.1')$$

subject to

$$A_i x - v_i + s_i - v_o + s_o = b, \quad i \in G_1 \qquad (1.2')$$

$$A_i x + v_i - s_i + v_o - s_o = b, \quad i \in G_2 \qquad (1.3')$$

$$x, b \text{ unrestricted} \qquad (1.4')$$

$$v_i, s_i \geq 0, \ i \in G \text{ and } i = 0 \qquad (1.5')$$

$$(m_1 \sum (A_i : i \in G_2) - m_2 \sum (A_i : i \in G_1))x = 1 \qquad (1.6a')$$

$$m_1 \sum (s_i - v_i : i \in G_2) + m_2 \sum (s_i - v_i : i \in G_1) + m(s_o - v_o) = 1 \qquad (1.6b')$$

We note that the normalization (1.6a′) remains the same as (1.6a), while the equivalent normalization (1.6b′) differs from (1.6b) by including a weighted term $s_o - v_o$. Models that included both the v_o and s_o variables were proposed in the early paper of Freed and Glover (1981), though the model (1′) and the normalizations accompanying it first appeared in Glover (1990).

The ability to place an emphasis on minimizing the greatest violation and on maximizing the least satisfaction (where, in the latter instance, the points can be accurately classified) has evident value in a variety of contexts. Although some researchers have looked at instances of model (1′) that include the variable v_o, apparently none of these instances have also included s_o. As we will show later, s_o has a crucial role in procedures for creating more robust separations. Also, it appears that no study has attempted to examine the consequences of incorporating the v_o variable within the same formulation as one that attaches non-zero h_i and k_i coefficients to the remaining v_i and s_i variables.

A common variation of the formulations (1) and (1′) replaces b by $b - \varepsilon$ in (1.2) and (1.2′), and by $b + \varepsilon$ in (1.3) and (1.3′), for a selected positive value of ε. The purpose of this variation is to push the correctly classified points farther from the quasi-separating hyperplane. This approach faces the difficulty of pre-specifying what an appropriate value of ε should be, particularly since this value interacts with the value of the right hand side constant in the normalization constraints. Formulation (1′) that includes s_o provides a way to implicitly handle the influence of ε, while also handling additional considerations.

In particular, replacing b by $b - \varepsilon$ and by $b + \varepsilon$ in the equations (1.2′), and (1.3′), respectively, is the same as assigning s_o a predetermined constant value of ε in the equations. Introducing s_o as a variable avoids the difficulty of having to figure out in advance an appropriate value for s_o to receive, and permits the flexibility of an interaction between the variables s_i and s_o by varying the coefficients k_i and k_o. In general, we may interpret the inclusion of a constant ε term to be the same as stipulating that s_o has a positive lower bound of ε, making it possible to both retain s_o in the model and also replace b by $b - \varepsilon$ and by $b + \varepsilon$ in the associated equations. By means of this change, s_o will receive a value that is the difference between ε and the true value of s_o. The use of ε as a lower bound on s_o can be appropriate in cases where we seek to assure a minimum

separation from the hyperplane regardless of all other considerations. An appropriate calibration for the value of ε can be achieved by once again making recourse to post-optimization, changing ε in a series of steps and noting the tradeoffs that result. As we argue later, such a calibration can be valuable for the purpose of achieving a model that is robust in its ability to correctly classify data from hold-out samples.

1.4 Variant of the More General Linear Model

The v_o and s_o variables of the model of Section 1.3 can be introduced in another fashion, removing them from the constraints (1.2′) and (1.3′) and incorporating the additional set of constraints $v_o \geq v_i$ and $s_i \leq s_o$, $i \in G$, to produce the following formulation

$$\text{Minimize } \sum (h_i v_i - k_i s_i : i \in G) + h_o v_o - k_o s_o \qquad (1.1'')$$

subject to

$$A_i x - v_i + s_i = b, \quad i \in G_1 \qquad (1.2'')$$

$$A_i x + v_i - s_i = b, \quad i \in G_2 \qquad (1.3'')$$

$$x, b \text{ unrestricted} \qquad (1.4'')$$

$$v_i, s_i \geq 0, \ i \in G \text{ and } i = 0 \qquad (1.5'')$$

$$(m_1 \sum (A_i : i \in G_2) - m_2 \sum (A_i : i \in G_1))x = 1 \qquad (1.6a'')$$

$$m_1 \sum (s_i - v_i : i \in G_2) + m_2 \sum (s_i - v_i : i \in G_1) = 1 \qquad (1.6b'')$$

$$v_o \geq v_i \text{ and } s_i \leq s_o, \ i \in G \qquad (1.7'')$$

Here (1.2″) and (1.3″) are the same as (1.2) and (1.3), and (1.6b″) is the same as (1.6b). Otherwise, except for the addition of the new constraints (1.7″), the rest of formulation (1″) is the same as formulation (1′).

The changes that produce formulation (1″) enable the model to accomplish more complex objectives than handled by formulation (1′), encompassing more complex trade-offs between v_o and s_o and the other v_i and s_i variables, as the values of the coefficients h_o and k_o are changed relative to values of the other h_i and k_i coefficients. Additional advantages to formulation (1″) arise when the linear programming models are extended to mixed integer programming models.

Formulation (1″) can be generalized further to give weight to objectives of minimizing the maximum degree of violation, or maximizing the minimum degree of satisfaction, over various subsets of points $S_1, \ldots, S_r$. To accomplish this we introduce variables v_{oq} and s_{oq}, $q = 1,\ldots, r$, which we assign coefficients h_{oq} and k_{oq} in the objective function, and impose the inequalities

$$v_{oq} \geq v_i \text{ and } s_i \leq s_{oq}, \ i \in S_q, \ q = 1\ldots., r. \qquad (1.8'')$$

Such inequalities can be incorporated only for the v_{oq} variables or only for the s_{oq} variables, according to the set S_q considered. We can additionally generalize (1.8″) by means of the constraint

$$v_{oq} \geq \sum (h_{iq} v_i : i \in G) \text{ and } \sum (k_{iq} s_i : i \in G) \leq s_{oq}, \ q = 1\ldots., r \qquad (1.9'')$$

noting that (1.8″) results from (1.9″) by setting $h_{iq} = 1$ and $k_{iq} = 1$ only for $i \in S_q$, and $h_{iq} = k_{iq} = 0$ for $i \in G - S_q$. This type of generalization has uses in obtaining approximate solutions to mixed integer programs by solving only linear programming problems (Glover, 2006a).

We do not explore applications of (1.8″) or (1.9″) in this paper, but will make explicit use of the model that embodies (1.7″). In this connection, formulations of subsequent sections that include v_o or s_o in equations such as (1.2′) and (1.3′) can be modified by removing v_o and s_o from these equations (producing equations corresponding to (1.2″) and (1.3″)) and then adding constraints of the form (1.7″), which permits the normalization constraint to be expressed as in (1.6b″), which is the same as the original form (1.6b).

1.5 Pre-processing to Reduce Problem Size and Achieve Better Separations

Pre-processing provides a natural way to reduce the size of the problem to be solved, and also to permit the separating hyperplane strategies to achieve better separations. A form of pre-processing we suggest in this regard is the following. Let $D(A_p,A_q)$ be a measure of the distance between two points A_p and A_q, for $p, q \in G$. For a given point A_i, $i \in G_k$, let k* denote the index complementary to k (k* = 2 if k = 1, and k* = 1 if k = 2). We then define the quantities

For A_i, $i \in G_k$:

$$D_{min}(A_i) = \text{Min}(D(A_i,A_p): p \in G_{k*})$$
$$S(A_i) = \{q \in G_k: D(A_i,A_q) < D_{min}(A_i)\}$$

The larger the values of $D_{min}(A_i)$ and of $|S(A_i)|$ relative to other points A_q, $q \in G_k$, the more "deeply embedded" the point A_i is within the Group k (and isolated from points in Group k*). We conjecture that if we remove such a deeply embedded point A_i from Group k before seeking to generate a separating hyperplane, the chances are high that A_i will automatically fall on the desired side of the hyperplane that is generated without referring to this point.

Consequently, we suggest a pre-processing step that makes use of the quantities $D_{min}(A_i)$ and $|S(A_i)|$ to select more deeply embedded points and temporarily set them aside, thus shrinking the size of the problem to be solved when seeking a separating hyperplane. For example, to identify points to be removed in this fashion, thresholds can be established on minimum sizes of $D_{min}(A_i)$ and $|S(A_i)|$, or of a weighted combination of these quantities, that will admit only a specified portion of the points A_i, $i \in G_k$ to qualify for a "deeply embedded" designation. Once the hyperplane model is solved, if a point A_i that was temporarily set aside lies on the wrong side of the resulting hyperplane, then it can be introduced into the formulation. By using a linear programming post-optimality procedure, the solution process can continue from the current optimized solution (relative to the set of points that excluded A_i) without having to re-start the solution process from scratch. Such an approach complements the approach of using post-optimality to reduce the impact of outliers by reducing their objective function coefficients.

The values $D_{min}(A_i)$ and $|S(A_i)|$ can be refined by applying a second order process. For this, we make use of the quantity

$$T(A_i) = \{q \in G - G_k: D(A_i,A_q) < D_k\}$$

where D_k is a distance measure given by $D_k = Mean(D_{min}(A_i): i \in G_k)$, or more generally determined to insure that a certain portion of the points A_i, $i \in G_k$ satisfy $D_{min}(A_i) \leq D_k$. If $|T(A_i)|$ is relatively large compared to the value $|T(A_q)|$ for other points A_q in Group k, then A_i may be considered as representing an anomalous (hard-to-classify or "out-of-place") point. Such a point can introduce a distortion in defining $D_{min}(A_p)$ and $|S(A_p)|$ for points A_p that lie in the opposite Group k*. (This is particularly true when a small value of D_k is used in the definition of $T(A_i)$.) Consequently, based on a first determination of $D_{min}(A_i)$ and $S(A_i)$, we can make an improved second determination by choosing a threshold T_k for elements of Group k to identify a set E_k of points that are sufficiently anomalous to be excluded from consideration

$$E_k = \{i \in G_k: |T(A_i)| \geq T_k\}.$$

where, as suggested, T_k may be chosen to assure that E_k will not exceed a certain limited size. (Alternatively, E_k can be identified by arranging the $|T(A_i)|$ in descending order and choosing at most a specified number of the largest values, stopping if the size of $|T(A_i)|$ abruptly decreases or reaches 0. A similar process can be used to determine D_k itself.) Then, in particular, for a point A_i, $i \in G_k$, we re-define

$$D_{min}(A_i) = Min(D(A_i,A_p): p \in G_{k*} - E_{k*}).$$

The composition of $S(A_i)$ is then re-determined based on this new (second level) definition.

A more cautious version of the exclusion set E_k is given by

$$E_k^{o} = \{i \in G_k: S(A_i) = \varnothing\}$$

which may be used in place of E_k (i.e., E_{k*}^{o} can be used in place of E_{k*}) in the preceding second level definition of D_{min}. Still more cautiously, first define

$$S[G_k] = \cup(S(A_i): i \in G_k\}.$$

Then the exclusion set E_k may be replaced by

$$E_k^{+} = \{i \in G_k - S[G_k]: S(A_i) = \varnothing\}.$$

We call points A_i for $i \in E_k^{+}$ *strongly anomalous* points, and stipulate that these points, if any exist, may be removed from G_k permanently.

It is important in implementing these forms of pre-processing to first scale the data, so that each attribute j, represented by the (column) vector $A_{\bullet j} = (a_{1j}\ a_{2j}\ \ldots\ a_{mj})$, is normalized, as for example by dividing through by $\sum(|a_{ij}|: i \in G)$, understanding that attribute j may be discarded if $A_{\bullet j}$ is the 0 vector.

More advanced forms of pre-processing can be based on the use of cluster analysis to produce clusters of points whose elements lie strictly within a given group, and then to subject such clusters to a successive perfect separation analysis as described in Section 3. The preceding use of $D_{min}(A_i)$, $S(A_i)$, $T(A_i)$ and the associated exclusion sets can be incorporated into a more general clustering procedure that can be used in this fashion (Glover, 2006b). (Another use of $D_{min}(A_i)$ is also given in Section 3.)

1.6 Retrospective Enhancement for Robust Separation – First Level

An important goal in applying separating hyperplane strategies is to go beyond the immediate objective of minimizing a measure of misclassifications, and in general, to generate hyperplanes that separate the correctly classified points of Group 1 from those of Group 2 by a greater distance – even where a perfect classification cannot be achieved. A hyperplane that achieves this additional separation objective is likely to be better at classifying new points that are not contained in the initial G_1 and G_2, and thus provide a more robust separation model.

The goal of increasing the separation between correctly classified points is supported by the inclusion of the s_i variables in the objective function of formulation (1) and by the additional inclusion of the s_o variable in formulations (1′) and (1″). However, by themselves, the preceding models do not have full latitude to achieve this goal. In order to identify hyperplanes that place correctly classified points as far as possible from the hyperplane boundary, we make use of the s_o variable in an additional fashion, by employing a *retrospective enhancement* process that re-solves the hyperplane separation problem, taking advantage of knowledge obtained from the previous solution of the problem.

Let C_1 and C_2 denote the sets of correctly classified points obtained by solving model (1′) or (1″). (For convenience, as here, we often employ the convention of referring to points by naming their index sets.) That is, the initial solution correctly assigns A_i to G_1 for $i \in C_1$ and correctly assigns A_i to G_2 for $i \in C_2$, hence $C_1 \subseteq G_1$ and $C_2 \subseteq G_2$. Let $C = C_1 \cup C_2$, and define $m_{c1} = |C_1|$, $m_{c2} = |C_2|$ and $m_c = |C|$.

Then we can seek a better separation of these points by retrospectively solving the new problem

$$\text{Maximize } \sum (k_i s_i: i \in C) + k_o s_o \qquad (1.1^o)$$

subject to

$$A_i x + s_i + s_o = b, \quad i \in C_1 \qquad (1.2^o)$$

$$A_i x - s_i - s_o = b, \quad i \in C_2 \qquad (1.3^o)$$

$$x, b \text{ unrestricted} \qquad (1.4^o)$$

$$s_i \geq 0,\ i \in C \text{ and } i = 0 \qquad (1.5^o)$$
$$(m_{c1}\sum (A_i: i \in C_2) - m_{c2}\sum (A_i: i \in C_1))x = 1 \qquad (1.6a^o)$$
$$m_{c1}\sum (s_i: i \in C_2) + m_{c2}\sum (s_i: i \in C_1) + m_c s_o = 1 \qquad (1.6b^o)$$

The value k_o in the preceding formulation may appropriately be chosen to be significantly larger than $\sum (k_i: i \in C)$. However, we may reasonably give small positive values to the k_i coefficients for $i \in C$ rather than setting these coefficients to 0, since there may be many alternative solutions that maximize the minimum value of s_o, and we are most interested in those that also push individual points away from the hyperplane, as well as those that merely achieve the objective related to s_o. Figure 3 depicts the effects of retrospective enhancement.

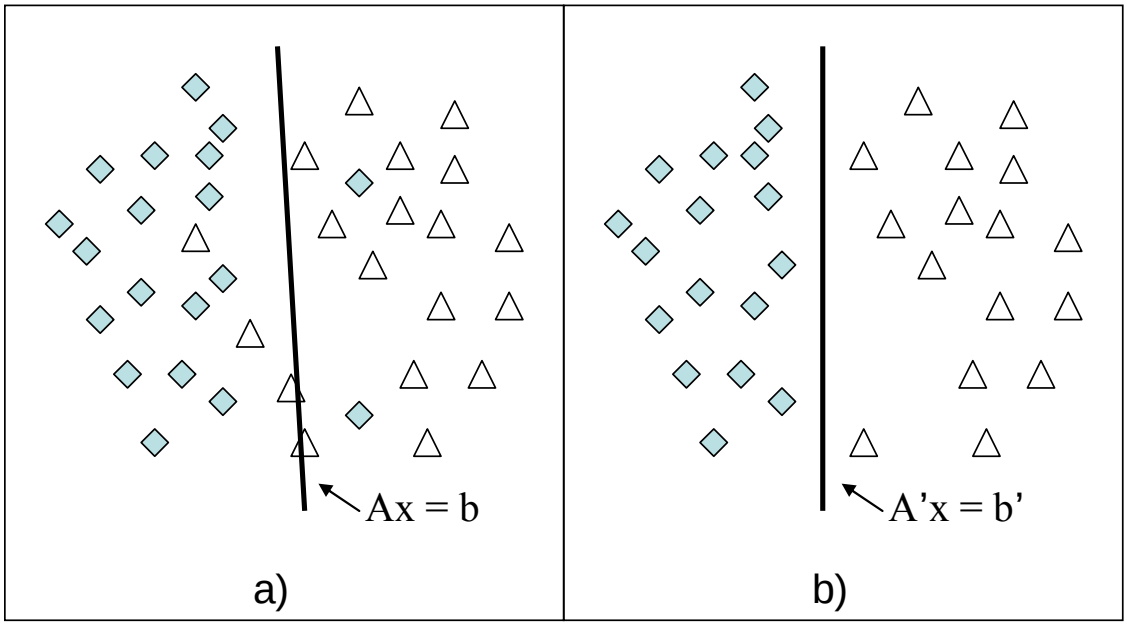

Figure 3: an example of retrospective enhancement

Significantly more advanced forms of retrospective enhancement will be introduced in connection with successive separation strategies, which we examine next.

2. Tree-Based Models Using Successive Separation Strategies

2.1 Beginning Considerations

A tree-based approach proposed in Glover (1990) allows a more complete means of separating two groups by making use of a successive separation (SS) process. Each stage of the process seeks to separate points that are incompletely differentiated, i.e., not yet correctly classified, by hyperplanes generated at preceding stages. This type of approach can also be used as a method for multi-group classification, where (as noted in Freed and Glover (1981)) any collection of groups whose members have not yet been isolated from all other remaining groups can be divided into two subsets, where one subset is defined to be Group 1 and the other subset is defined to be Group 2. Using such a division, a

hyperplane is then generated to isolate Group 1 as nearly as possible from Group 2, producing discrimination sets D_1 and D_2 where D_1 consists of those points in the half-space used to classify points (correctly or incorrectly) as belonging to Group 1 and D_2 consists of those points in the complementary half-space used to classify points as belonging to Group 2 ($D_1 \cup D_2 = G_1 \cup G_2$). ($D_1 \cap G_1$ and $D_2 \cap G_2$ correspond to the sets C_1 and C_2 of correctly classified points discussed in Section 1.5, except that these sets may be targeted in the present case to contain elements of more than a single group.)

Successive subdivisions of this type encompass alternatives ranging from a binary tree (where at each stage approximately half of the groups currently being considered are allocated to Group 1 and the remaining half are allocated to Group 2) to a one-at-a-time form of separation (where a single group is isolated from all remaining groups at each stage). Independent of the mode of subdivision, the process typically generates $g - 1$ hyperplanes to separate g different groups. Fewer hyperplanes may be generated in the case where a classification attempt at a particular stage is done very poorly, and some group assigned to G_1 (or respectively to G_2) fails to have any of its elements appear in the set D_1 (respectively D_2).

To achieve a more effective classification, the SS approach can be performed multiple times, using different forms of subdivision on each occasion. The outcomes can then be subjected to conditional Bayesian analysis to provide a composite classification scheme, yielding a rule that accounts for the decisions produced by each tree. The composite scheme can accordingly be used to determine the group that a new point should be assigned to. In a related manner, a voting scheme based on the outcomes of the different trees can also be used to provide an overall rule (Glover, 2006b).

2.2 Extended SS Approaches

In contrast to the first level approach sketched in Section 2.1, the main use of the SS process is to create a mode of successive separation that does not stop its examination of a discrimination set D_1 or D_2 when the elements of the set belong only to a single one of the original groups, but continues to generate hyperplanes to achieve an improved classification.

We discuss this process by returning the focus to the two-group case. When G_1 and G_2 give rise to the two discrimination sets D_1 and D_2 by the hyperplane separation process, if either D_1 or D_2 contains points of both G_1 and G_2, then this set D_k may be subjected to a new hyperplane separation effort to try to separate the residual elements of G_1 and G_2 that it contains. The examination of set D_k to separate its G_1 elements from its G_2 elements (i.e., to separate $D_k \cap G_1$ from $D_k \cap G_2$) thus gives D_k the same role as the original G, whose G_1 and G_2 elements were subjected to a classification attempt on the original step. Thus, each step simply repeats the process applied on the first step. The SS procedure stops upon reaching a point where a new hyperplane fails to achieve a more effective classification or fails to improve on the previous classification by a pre-specified amount. The procedure may also stop by reaching a limit imposed on the number of subdivisions allowed, i.e., by reaching a selected limit on the depth of the tree.

As observed in Glover (1990), such an approach can be conveniently supplemented at each stage by employing a simple calculation to determine how far to shift the current hyperplane in each direction (by increasing and decreasing discriminant b) to reach a position where all points of either Group 1 or Group 2 will lie entirely on one side of the hyperplane (the side that includes the original hyperplane). We call the group that falls entirely on one side of the shifted hyperplane the *primary* group, and call the other group the *secondary* group. Correspondingly, we refer to the two sides of the shifted hyperplane as the primary and secondary sides. (That is, the primary side contains all points of the primary group. The primary/secondary classification that results by shifting the hyperplane in a given direction may be the same or different from the classification that results by shifting the hyperplane in the opposite direction.) It can be useful in this approach to determine the original hyperplane by incorporating the balanced violation constraint

$$m_1 \sum (v_i: i \in G_2) = m_2 \sum (v_i: i \in G_1),$$

as discussed at the end of Section 1.1.

Some points of the secondary group may lie on the primary side of the shifted hyperplane, but by changing b as little as possible to yield the separation, as many points as possible of the secondary group will lie on the secondary side. Moreover, all points of the secondary group that lie on the secondary side are perfectly classified by this separation, since no points of the primary group lie on the secondary side. Consequently, the subset of perfectly classified secondary points can be eliminated at once before continuing additional stages of separation.

For some types of configurations, it may be that no points of the secondary group can be eliminated in this fashion.[3] However, in many instances the approach of shifting b in the two different directions will be able to eliminate points from at least one of the two groups.

2.3 Successive Perfect Separation (SPS)

Taking the idea of the shifting hyperplane process a step farther, it is natural to employ a *successive perfect separation* (SPS) strategy that operates as follows. Rather than adopting the objective of trying to separate the groups in order to minimize the total weighted sum of violations (or one of the other related objectives addressed in Section 1), the SPS strategy seeks to establish a separation based on the primary/secondary group distinction, and thereby to achieve such a separation in a more rigorous fashion. For a given choice concerning which of Group 1 or Group 2 will be treated as the primary group, the SPS approach employs a model where the violation variables v_i of this group

[3] A simple worst case scenario is illustrated by the situation where a square is partitioned into 4 regions created by its two diagonals, and Group 1 and Group 2 each occupy two non-adjacent regions. Then no points of either group can be perfectly classified by this approach. We later describe a way to thwart this situation.

are given pre-emptively large h_i coefficients in the objective function, so that all of its points are assured to lie on one side of the hyperplane, while achieving a "best separation" for the secondary group subject to this pre-emptive objective. Still, more directly, we may simply structure the model so that all points of the primary group fall on one side of the hyperplane.

If Group 1 is treated as the primary group, the model may be expressed in the form

$$\text{Minimize } \sum (h_i v_i - k_i s_i : i \in G_2) + h_o v_o - k_o s_o \qquad (2.1)$$

subject to

$$A_i x + s_i + s_o = b, \quad i \in G_1 \qquad (2.2)$$

$$A_i x + v_i - s_i + v_o - s_o = b, \quad i \in G_2 \qquad (2.3)$$

$$x, b \text{ unrestricted} \qquad (2.4)$$

$$s_i \geq 0, \ i \in G \text{ and } i = 0; \ v_i \geq 0, \ i \in G_2 \text{ and } i = 0 \qquad (2.5)$$

$$(m_1 \sum (A_i : i \in G_2) - m_2 \sum (A_i : i \in G_1))x = 1 \qquad (2.6a)$$

$$m_1 \sum (s_i - v_i : i \in G_2) + m_2 \sum (s_i : i \in G_1) + m s_o - m_1 v_o = 1 \qquad (2.6b)$$

As previously noted, we can remove s_o and v_o from the equations where they appear in this formulation, and instead introduce inequalities corresponding to those of (1.6b″). In most applications of this model, the variable v_o can be disregarded, while the variable s_o takes a dominant role in relation to the variables s_i for $i \in G_2$ (by making k_o appreciably larger than the k_i coefficients). Formulation (2) is particularly useful for its ability to achieve the same effect as the first level retrospective enhancement provided by model (1^o), for situations in which the primary and secondary groups can be perfectly separated. We make use of this ability later in a more advanced form of retrospective enhancement.

In order to decide which group should be primary in an SPS approach based on formulation (2), two instances of this formulation are solved at each stage, one for Group 1 in the role of the primary group and one for Group 2 in this role. Then the instance that yields the best outcome (as measured, for example, by correctly classifying the largest number of points, or by correctly classifying the largest proportion of points in the secondary group) is used to identify which group will be designated primary. If both choices for the primary/secondary roles are unattractive, because the solution to (2) yields too few elements of the secondary group that lie in its targeted half-space (i.e., too few secondary points A_i yield $v_i = 0$), then the SPS approach incorporates an "SS intervention step" to generate a hyperplane by the customary successive separation process (using, for example, a formulation such as (1′) or (1″)), thus producing two continuations via discrimination sets D_1 and D_2 as discussed in the preceding section. For each continuation, the method then re-establishes the SPS focus by solving the type of model illustrated in formulation (2), based on invoking the primary/secondary distinction.

The SPS approach employs the same type of termination criteria employed in the regular SS process. However, if a continuation is terminated by the criterion of limiting the depth of the tree, the final step likewise discards the primary/secondary distinction and reverts to the type of branch step employed in the regular SS procedure. The discrimination sets D_1 and D_2 thus produced are the leaf nodes of the indicated continuation.

We now turn to a context that makes it possible to take much fuller advantage of the hyperplane separation models, and of extensions we subsequently identify.

3. Advanced Forms of Retrospective Enhancement and SPS

Retrospective enhancement, as discussed in section 1.4, becomes of greater significance in the successive separation approaches, and especially in the context of successive perfect separation, than in the simpler strategies that rely on generating a single hyperplane. To introduce a form of retrospective enhancement applicable to this setting, we generalize the discussion of the SPS approach to consider the use of regions that include half-spaces as a special case. This generalized perspective makes it possible to exploit regions generated by mixed integer multi-hyperplane separation methods, subsequently discussed.

3.1 General Description of Successive Perfect Separation

A general form of successive perfect separation implicitly includes ordinary successive separation as a special case, since as previously noted we employ an SS intervention step whenever an SPS step fails to perfectly classify a sufficiently large portion of either group. Likewise, in the more common situation where the SPS process reaches a limiting depth without having completely separated Group 1 from Group 2, we conclude the process with a final SS step.

To describe steps of a general SPS approach (that includes such SS steps), we replace the reference to hyperplanes and half-spaces by a reference to *regions* R_1 and R_2 that are generated with the goal of isolating the two groups G_1 and G_2 as nearly as possible from each other. More precisely, we refer to a collection of regions denoted by $R_1(d)$ and $R_2(d)$, generated at successive depths $d = 1, \ldots, d_o$.

Let $G = \{A_i: i \in G\}$ and $G_k = \{A_i: i \in G_k\}$ for k = 1, 2. In contrast to our usual convention of referring to a set of points by naming its index set, it is useful in the present setting to differentiate points from index sets more precisely. As in the preceding notation, we use italicized symbols in general to refer to collections of points.

In the same way that we consider a pair of half-spaces defined by a separating hyperplane to be complementary (by implicitly supposing one of member of the pair is an open set, which may be enforced by a lower bound ε on s_o), we assume that the regions $R_1(d)$ and $R_2(d)$ are mutually complementary and partition the space R^n of all real n vectors $A = \{a_1, \ldots, a_n\}$ (of which the points A_i in G are specific instances). Later integer programming formulations that likewise incorporate an s_o variable will be designed to assure this condition.

In a successive perfect separation process, where portions of G_1 and G_2 are perfectly classified and removed from consideration at various stages, we refer to the residual subsets of G_1 and G_2 that remain to be considered at depth d by $G_1(d)$ and $G_2(d)$.

Likewise, we refer to the residual portion of G itself by $G(d)$ (= $G_1(d) \cup G_2(d)$). (Initially, for d = 1, we have $G_1(d) = G_1$, $G_2(d) = G_2$ and $G(d) = G$.)

Let $C_k(d)$ for k = 1, 2 identify the set of points that are correctly classified by the region $R_k(d)$ as a result of belonging to the associated subset $G_k(d)$ of G_k at the current depth d; that is $C_k(d) = G_k(d) \cap R_k(d)$. The associated set of points $I_k(d)$ that are incorrectly classified by the region $R_k(d)$ is given by $I_k(d) = G_k(d) - R_k(d)$. (Hence by the complementary relationship between $R_1(d)$ and $R_2(d)$, we also have $I_1(d) = G_1(d) \cap R_2(d)$ and $I_2(d) = G_2(d) \cap R_1(d)$.)

In a direct parallel with the type of separations based on half-spaces, an SS approach generates regions $R_1(d)$ and $R_2(d)$ by reference to the goal of optimizing a function F($G_1(d)$, $G_2(d)$), where this function favors regions that give rise to empty sets $I_1(d)$and $I_2(d)$ of misclassified points, when this is possible – as by minimizing a weighted sum of violations of points lying in $I_1(d)$and $I_2(d)$, or by minimizing the number of points lying in these two sets, and so forth. (The formulations of preceding sections give examples of various forms of F($G_1(d)$, $G_2(d)$) when the regions $R_1(d)$ and $R_2(d)$ consist of complementary half-spaces.)

For the SPS approach, we refer to the primary region by the index k and refer to the secondary region by the index k*, i.e., k* = 2 if k = 1 and k* = 1 if k = 2. Thus $R_k(d)$ and $R_{k*}(d)$ will refer the primary and secondary regions at depth d (where the identity of k can change as d changes), and we similarly refer to $G_k(d)$ and $G_{k*}(d)$ as the associated primary and secondary groups. By this designation, a successive perfect separation approach operates by enforcing the requirement that $R_k(d)$ includes all of $G_k(d)$ (i.e., $R_k(d) \supseteq G_k(d)$), hence assuring $I_k(d) = \varnothing$. Subject to this requirement, the approach seeks to generate regions $R_k(d)$ and $R_{k*}(d)$ that optimize a function $F_o(G_k(d), G_{k*}(d))$, which like the SS function F($G_1(d)$, $G_2(d)$) undertakes to minimize a weighted sum of violations (which in the present case is restricted to points in $I_{k*}(d)$) or to minimize the number of violations, and so forth.

It should be pointed out that the condition $I_k(d) = \varnothing$ in the SPS approach does not imply that the set of correctly classified points $C_k(d) = G_k(d)$ is perfectly classified, since on the contrary the primary region $R_k(d)$ may include points of $G_{k*}(d)$ (i.e., the subset of $G_{k*}(d)$ that constitutes the set $I_{k*}(d)$). Rather, it is the set $C_{k*}(d)$ that identifies the perfectly classified points, given that the region $R_{k*}(d)$ containing $C_{k*}(d)$ includes no points of $G_k(d)$.

In accordance with earlier discussions, we also emphasize the importance of selecting the SS function F($G_1(d)$, $G_2(d)$) and the SPS function $F_o(G_k(d), G_{k*}(d))$ to include provision for encouraging the greatest possible separation of the sets $C_1(d)$ and $C_2(d)$. For example, the objective functions in each of these cases may include reference to maximizing some function of the distances of points in $C_1(d)$ from the boundary of $R_1(d)$ and of points in $C_2(d)$ from the boundary of $R_2(d)$. Instances of such an objective are given by the models of previous sections that seek to separate correctly classified points by assigning

objective function weights to the variables s_o and s_i. $i \in G$. This robustness consideration will be treated more fully in the material that follows.

To describe the SPS approach based on these conventions, we let d_o denote a selected upper limit on the depth d of the tree implicitly generated by the SPS process,[4] and let *List* denote a list of groups $G(d)$ $(= G_1(d) \cup G_2(d))$ that are slated to be examined for separation. Also, let *Correct* and *Incorrect* respectively denote the sets of correctly classified and incorrectly classified points that are updated at various junctures in the application of the method (corresponding to the events of identifying leaf nodes of the tree implicitly generated).

To specify the situation where an SS intervention step is employed, we make use of an *aspiration threshold* $T \geq 0$ that provides a strict lower bound on an admissible number of correctly classified points that result by optimizing the SPS function $F_o(G_k(d), G_{k*}(d))$. Thus, T provides a bound on the desired number of perfectly classified points in $G_{k*}(d)$ – hence the number of points in $C_{k*}(d)$ $(= R_{k*}(d) \cap G_{k*}(d))$. The signal to perform an SS intervention occurs when the optimization of $F_o(G_k(d), G_{k*}(d))$ subject to $R_k(d) \supseteq G_k(d)$ fails to satisfy $|C_{k*}(q)| > T$, and instead yields $|C_{k*}(q)| \leq T$.

General SPS Procedure

0. (Initialization) Set d = 1, and $G(d) = G(1) = G$. Let $List = \varnothing$ and similarly set $Correct = Incorrect = \varnothing$.
1. Identify the two component groups making up $G(d)$, given by $G_1(d) = G(d) \cap G_1$ and $G_2(d) = G(d) \cap G_2$. Select one of these, denoted $G_k(d)$, to be primary and the other, $G_{k*}(d)$, to be secondary.
2. Identify the regions $R_k(d)$ and $R_{k*}(d)$ that optimize the SPS function $F_o(G_k(d), G_{k*}(d))$ subject to $R_k(d) \supseteq G_k(d)$.
   (a) If $|C_{k*}(d)| \leq T$ (hence the requirement set by the aspiration threshold is violated), proceed to Step 5 to execute an SS intervention.
   (b) If $|C_{k*}(d)| > T$, proceed to Step 3 to complete the updates of the perfect separation process.
3. Set $G(d+1) = G_k(d) \cup I_{k*}(d)$ (where $I_{k*}(d) = G_{k*}(d) - C_{k*}(d)$, and $G_k(d) = C_k(d)$). Add the perfectly classified points of $C_{k*}(d)$ to the set *Correct*. (The points in $C_{k*}(d)$ are automatically eliminated from future consideration since they do not belong to $G(d+1)$.)
   (a) (Branch termination by complete separation) If $I_{k*}(d) = \varnothing$, then all points of $G(d+1)$ $(= C_k(d))$, have been perfectly classified. Add the points of $C_k(d)$ to *Correct*, and proceed to Termination/Continuation at Step 6.
   (b) If $I_{k*}(d) \neq \varnothing$, proceed to Step 4 .
4. (Depth update) Let d := d + 1. If $d < d_o$ (the limiting depth) then return to Step 1. Otherwise, if $d = d_o$, proceed to Step 5.

[4] In many applications, the value of d_o can be chosen relatively small, such as 4 or 6. It would be rare to find an application making use of a d_o value greater than 8.

5. (SS Intervention (from Step 2(a)) or final step of SPS process (from Step 4)) Optimize the SS function F($G_1(d)$, $G_2(d)$).
   (a) (SPS final step, on reaching limiting depth d_o) If Step 5 is reached from Step 4, then the method ends its examination of the current $G(d) = G(d_o)$. Based on the optimization of F($G_1(d)$, $G_2(d)$) for $d = d_o$, add the final correctly classified sets $C_1(d_o)$ (= $R_1(d_o) \cap G_1(d_o)$) and $C_2(d_o)$ (= $R_2(d_o) \cap G_2(d_o)$) to *Correct*, and the final incorrectly classified sets $I_1(d_o)$ (= $R_2(d_o) \cap G_1(d_o)$) and $I_2(d_o)$ (= $R_1(d_o) \cap G_2(d_o)$) to *Incorrect*. Proceed to Termination/Continuation at Step 6.
   (b) (SS Intervention) If Step 5 is reached from Step 2(a), identify two new potential continuations by defining the groups $G_1(d+1) = C_1(d) \cup I_2(d)$ and $G_2(d+1) = C_2(d) \cup I_1(d)$ (constituting the children of $G(d)$). Test for reaching the limiting depth d_o.
      (1) If $d + 1 < d_o$: set $d := d + 1$ and add $G_1(d)$ and $G_2(d)$ to *List* (to become candidates to be selected as a group $G(d)$). Proceed to Step 6.
      (2) Otherwise, if $d + 1 = d_o$: add $C_1(d_o)$ and $C_2(d_o)$ to *Correct* and add $I_1(d_o)$ and $I_2(d_o)$ to *Incorrect* (without incrementing d and without enlarging *List*). Proceed to Step 6.
6. (Termination/Continuation). If *List* is empty, the method terminates. Otherwise, choose and remove a set $G(d)$ from *List* (keeping track of the associated d value) and return to Step 1.

It is possible that the final step 5(a) of an SPS process caused by reaching the limiting depth d_o may achieve a perfect separation of both groups by optimizing the SS function just as if the method had instead optimized the SPS objective function; i.e., the conditions $I_1(d) = \varnothing$ and $I_2(d) = \varnothing$ may result in Step 5(a) just as $I_{k*}(d) = \varnothing$ (and by construction, $I_k(d) = \varnothing$) in Step 3(a). Figure 4 depicts an example of a SPS procedure, requiring a SS intervention at $d_o = 3$. In the first iteration we obtain hyperplane h_1, placing all triangular elements to the right of it. For the second iteration, we no longer need to consider any diamond-shaped points correctly classified by h_1 (shown as white diamonds in Figure 4.b); we then obtain h_2 by placing all remaining diamonds to its left. Again, for the third iteration we can ignore the triangles that were correctly classified by h_2 (shown as white triangles in Figure 4.c), and we perform an SS intervention – given that we are already at depth d_o – to fully separate the residual elements.

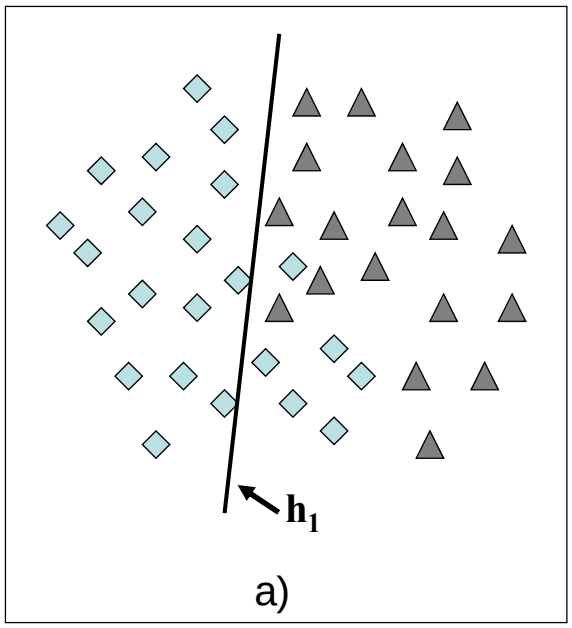

Figure 4a: step 1 – SPS by a single hyperplane (all triangles lie on one side)

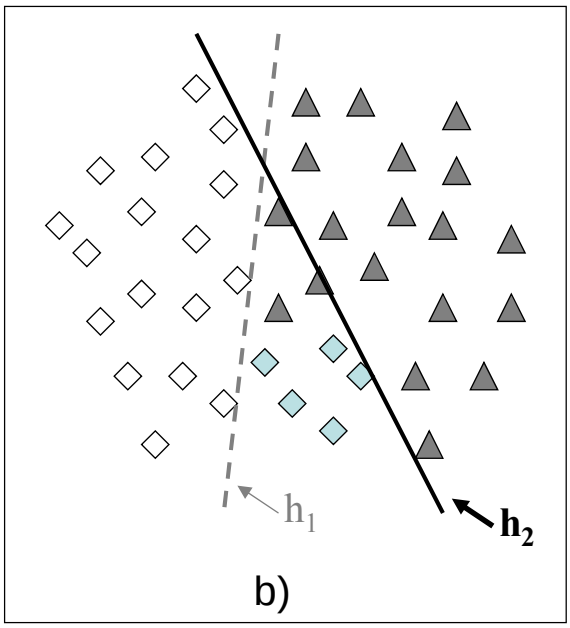

Figure 4b: step 2 – SPS of residual points by a second hyperplane (all diamonds lie on one side)

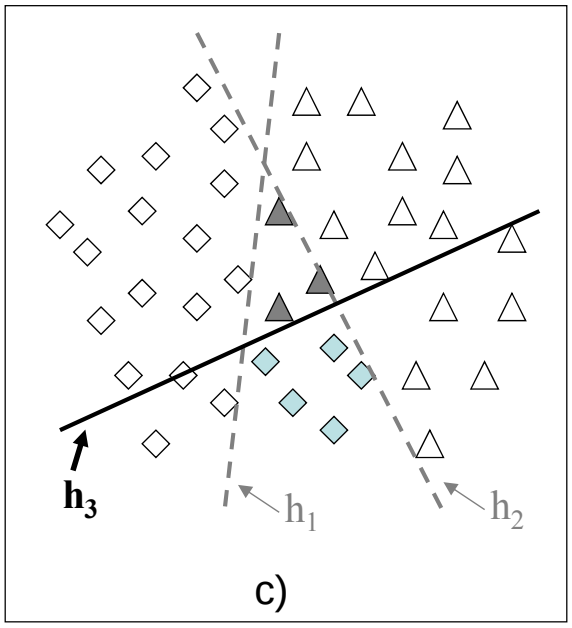

Figure 4c: step 3 – final separation of residual points by a third hyperplane (all points are now correctly classified)

We observe that the aspiration threshold T need not be a constant, but may be generated from a ratio of a desired size of $C_{k^*}(d)$ relative to the size of $I_{k^*}(d)$ or $G_{k^*}(d)$. Evidently, if T is made large enough, then the method becomes the same as an ordinary SS method. However, as the next section will make clear, there are advantages to choosing T relatively small, so that the method will perform perfect separation steps when possible.

3.2 Retrospective Enhancement – General Case

The key ideas underlying retrospective enhancement are as follows.

Treating SS Intervention. Although SS intervention is likely to be performed rarely (or not at all) during the execution of the General SPS Procedure, the handling of an SS intervention step for the purpose of retrospective enhancement is easy to specify. Each time such a step is carried out (by executing Step 5(b)), the optimization of F($G_1(d)$, $G_2(d)$) in Step 5 is immediately followed by a second optimization that seeks to separate the correctly classified points (constituting the sets $C_1(d)$ and $C_2(d)$) as fully as possible. Specifically, this is done by optimizing a function F′($C_1(d)$, $C_2(d)$) that is designed, as the formulation (1^o) of Section 1.5, to yield a first level retrospective enhancement.

The regions $R_1'(d)$ and $R_2'(d)$ obtained from this latter optimization will normally differ from the regions $R_1(d)$ and $R_2(d)$ obtained from the optimization of F($G_1(d)$, $G_2(d)$). Consequently, the solution that optimizes F′($C_1(d)$, $C_2(d)$) can yield a different pair of children $G_1(d+1)$ and $G_2(d+1)$ of $G(d)$ in Step 5(b) of the General SPS Procedure. For this reason the new solution obtained from this second optimization must be identified before adding these children to *List* in 5(b)(1) or recording the updates of *Correct* and *Incorrect* in 5(b)(2).

Treating Perfect Separation Sequences. The most effective form of retrospective enhancement results by re-processing a sequence of SPS steps that is not interrupted by SS intervention. This type of enhancement involves several layers of important considerations, and we devote the rest of the section to covering its details.

There are two major operations to be performed: creating a set of buffer regions and refining regions previously created (which can include discarding regions that become redundant or dominated by others).

As a foundation for both of these operations, we are concerned with subsequences of consecutively generated regions $R_k(d)$ and $R_{k*}(d)$, for $d = d_1, \ldots, d_2$, uninterrupted by an SS Intervention step, such that the index k of the primary group $G_k(d)$ remains unchanged for all d satisfying $d \in [d_1,d_2]$ (i.e., for all d such that $d_1 \le d \le d_2$), and such that the subsequence is maximal (hence, d_1 cannot be made smaller and d_2 cannot be made larger, subject to maintaining the primary group index unchanged). We observe that $G_k(d)$ itself remains invariant throughout this sequence, i.e., $G_k(d) = G_k(d_1) = G_k(d_2)$ for all $d \in [d_1,d_2]$, and we call such a sequence a *primary-invariant sequence*.

We also assume a first pass of the General SPS Procedure has already been performed, handling SS intervention steps as previously described (by optimizing the function F′($C_1(d)$, $C_2(d)$) each time such an intervention occurs, and adding the resulting children to *List*). Upon completion of this pass of the procedure by reaching Step 6 with *List* empty, we launch a new pass whose goal, roughly speaking, will be to create new regions $R_1(d)$ and $R_2(d)$, that separate G_1 and G_2 by the greatest possible amount, restricting

attention to those elements of these groups that were ultimately classified correctly on the first pass. Specifically, denote the sets G, $G_k(d)$, $R_k(d)$, $C_k(d)$, $I_k(d)$, *Correct* and *Incorrect* of the first pass of the General SPS procedure by G^o, $G_k^o(d)$, $R_k^o(d)$, $C_k^o(d)$, $I_k^o(d)$, *Correct*^{o} and *Incorrect*^{o}. Then we start the second pass by removing all points of the *Incorrect*^{o} set from each of the other sets, thus defining the new G that starts the second pass by $G = G^o - Incorrect^o$, or more simply $G = Correct^o$.

Evidently, if we generate exactly the same regions $R_k(d)$ and $R_{k*}(d)$ on a new pass of the General SPS Procedure as on the original pass ($R_k(d) = R_k^o(d)$ and $R_{k*}(d) = R_{k*}^o(d)$), then during a primary-invariant sequence the new $G_k(d)$ sets for $d \in [d_1,d_2]$ will consist of the original $G_k^o(d)$ sets reduced by the removal of the points of *Incorrect*^{o} (i.e., $G_k(d) = G_k^o(d) - Incorrect^o$), but the sets of correctly classified points, and in particular the sets $C_{k*}(d)$ that are added to the set *Correct* throughout the execution of a primary-invariant sequence, will be unchanged (i.e., they don't intersect *Incorrect*^{o}, and hence $C_{k*}(d) = C_{k*}^o(d)$).

3.2.1 Creating Buffer Regions

When the regions $R_k(d)$ and $R_{k*}(d)$ are used to classify new points, the SPS process dictates that a new point in the secondary region $R_{k*}(d)$ is assigned to Group k*, whereas any point in the primary region $R_k(d)$ may be assigned to either Group k or k* – a decision that is deferred until a secondary region is generated that contains the point. At the very end of the process, if the final step is an SS step, then a point is simply assigned to Group 1 if it lies in $R_1(d_o)$ and is assigned to Group 2 if it lies in $R_2(d_o)$. Thus the final step is treated as if it were a perfect separation of both groups, and by eliminating the elements of *Incorrect*^{o} from consideration, it does in fact create a perfect separation for the remaining points of these two groups in the original G of the second pass ($G = G^o - Incorrect^o$).

This gives rise to a special case for identifying the value d_2 of a primary-invariant sequence identified from the first pass. If a perfect separation of both sets did not occur when $d = d_2$ in such a sequence, and if $d_o = d_2 + 1$, then the first pass of the method performed a final SPS step that consisted of optimizing the SS function at Step 5. Under these circumstances, since we remove incorrectly classified elements from consideration on the second pass, we conclude as previously noted that the regions $R_1^o(d_o)$ and $R_2^o(d_o)$ generated on the final step of the first pass create a perfect separation of the two residual sets $G_1(d_o)$ (= $C_1^o(d_o)$) and $G_2(d_o)$ (= $C_2^o(d_o)$) available to the second pass. Consequently, when the final step follows on the heels on a primary-invariant sequence, we may treat the primary index k as remaining unchanged on this last step (regardless of the value of k when $d_2 = d_o - 1$). This makes it possible to re-define $d_2 := d_2 + 1 = d_o$ for the purpose of identifying the maximal primary-invariant sequence. This ability to make d_2 larger by 1 unit than it would otherwise be has useful consequences, as we will soon see.

In the second pass we begin with the same set of regions $R_1^o(d)$ and $R_2^o(d)$ produced at each step of the first pass, and hence each primary-invariant sequence starts with $R_k(d) = R_k^o(d)$ and $R_{k*}(d) = R_{k*}^o(d)$, for $d_1 \le d \le d_2$. As we undertake to modify the sets $R_k(d)$ and

$R_{k^*}(d)$ to create a better separation of correctly classified points, the union of the correctly classified secondary sets for the primary-invariant sequence during the first pass, which we denote by $C^* = \cup(C_{k^*}^{o}(d)\colon d_1 \le d \le d_2)$, plays an important role.

In general, there are two types of deficiencies of the first pass that we seek to remedy on the second pass. First, we don't want any part of the boundary of $R_{k^*}(d)$ to be too close to the set of points $G_k(d)$, because then it is quite possible that a new point that should belong to Group k but lies at the "edge" of a region containing Group k points will fall in $R_{k^*}(d)$, and consequently such a new point will mistakenly be assigned to Group k*.

The second type of deficiency is more subtle. During a primary-invariant sequence, consider particular point A_p in G_{k^*} that belongs to at least one of the regions $R_{k^*}(d)$, for $d \in [d_1,d_2]$ (hence $A_p \in C^*$) but A_p lies very close to the boundary of every such region that contains it. It would be advantageous to create an additional region $R_{k^*}(d)$ that acted as a "buffer" for A_p by containing A_p more deeply within it – and yet whose boundary was not too close to the set of points $G_k(d)$ (recalling that $G_k(d) = G_k(d_1)$ for all d of the primary-invariant sequence). Then a new point that lay within this additional $R_{k^*}(d)$ region would reasonably be classified as belonging to Group k*. But without the existence of this buffer region, the new point might lie very close to A_p and not so close to the points of Group k, so that it might ultimately be assigned to Group k.

One way to yield a better set of $R_k(d)$ and $R_{k^*}(d)$ regions, therefore, is to create buffer regions that "protect" points of C^* such as A_p that happen to lie very close to the boundary of every secondary region that contains them. (The protection is not really for points such as A_p, but for new points that lie close to these points but not within the current secondary regions, and that should be assigned to Group k* as well.)

The procedure we propose for doing this is the following.

For each $A_p \in C^*$, let $D(A_p)$ denote the maximum of the distances of A_p from the boundaries of those regions $R_{k^*}(d)$ in which A_p lies. (Thus $D(A_p)$ measures the amount of "protection" A_p has as a result of lying in C^*.) We refer again to the quantity $D_{min}(A_i)$ introduced in the discussion of pre-processing in Section 1.6, which for $A_p \in C^*$ (hence $A_p \in G_{k^*}(d_1)$) becomes

$$D_{min}(A_p) = \text{Min}(D(A_p,A_i)\colon A_i \in G_k(d_1))$$

noting that the foregoing definition remains the same if $G_k(d_1)$ is replaced by $G_k(d)$ for any $d \in [d_1,d_2]$ since the primary group remains unchanged for d in this interval.

If $D(A_p) < .5\, D_{min}(A_p)$, then it may be possible to create a new region $R_{k^*}(d')$ (e.g., for $d' = d_2 + 1$) that will better separate a point $A_p \in C^*$ from points of $G_k(d_1)$. This possibility arises from the fact that a region whose boundary lies half-way between A_p and the closest A_i in $G_k(d_1)$ would better protect A_p and yet also avoid the reverse risk of being too close to points of the primary group. (We can choose to make $d' > d_2$, rather than creating a separation involving the region $R_{k^*}(d')$ at an earlier point in the primary-

invariant sequence, because the type of protection provided by a buffer region can be provided at any step.) Consequently, we identify the set of these fertile points for buffering

$$FB = \{A_p \in C^*: D(A_p) < .5D_{min}(A_p)).$$

If *FB* is empty, then no buffering is attempted. Otherwise, we seek a new region $R_{k*}(d')$ to establish a better separation of the points of *FB* from those of $G_k(d_1)$.

It may not be possible to select a single new region $R_{k*}(d')$ that will provide a useful buffer for all points of *FB*, and hence we perform a simple variant of topological clustering (Glover, 2006b) to yield one or more subsets of *FB* that provide the foundation for generating new regions. For this we pre-order the distances $D(A_p,A_q)$ for $A_p,A_q \in FB$ in ascending order and construct a digraph DG whose nodes are the indexes p for the points $A_p \in FB$ and whose arcs (p,q) will be generated with the interpretation that A_p and A_q belong to the same cluster. By extension, then, A_p and A_q belong to the same cluster if and only if they lie within a connected component of the digraph. We identify these connected components by introducing a label $\rho(A_p)$ for each A_p that names the component A_p belongs to (where $\rho(A_p) = 0$ if A_p belongs to no connected component other than the trivial one consisting of the point A_p itself). Define the length of an arc (p,q) to be the distance $D(A_p,A_q)$. We make use of an upper limit UL on the largest length $D(A_p,A_q)$ that any arc (p,q) is allowed to have in DG. More particularly, we make use of an incremental limit Δ so that, when the lengths of arcs in a candidate set CA (defined below) are arranged in ascending order, we cease to examine arcs beyond the point where $D(p,q) > D(p',q'') + \Delta$, where $D(p,q)$ and $D(p',q'')$ are two successive distances in this ordering.

Cluster Method by Creating the Digraph DG

0. Create the set of candidate arcs $CA = \{(p,q): p < q, A_p, A_q \in FB, D(A_p,A_q) \leq UL\}$. Let $\rho_o = 0$ and $\rho(A_p) = 0$ for all $A_p \in FB$. The digraph DG begins with all nodes p for $A_p \in FB$ but without any arcs.
1. Let $(r,s) = \arg\min(D(A_p,A_q): (p,q) \in CA)$.
2. If $\rho_o > 0$ and $D(A_r,A_s) > Pre\_D + \Delta$, stop. Otherwise:
   (a) add arc (r,s) to the digraph DG and remove it from CA.
   (b) set $Pre\_D = D(A_r,A_s)$
   (c) If $\rho(A_r) = \rho(A_s) = 0$, set $\rho_o := \rho_o + 1$ and set $\rho(A_r) = \rho(A_s) = \rho_o$.
   (d) If $\rho(A_r) = 0$ and $\rho(A_s) > 0$ set $\rho(A_r) := \rho(A_s)$, and remove from CA all arcs (r,q) or (q,r) such that $\rho(A_q) = \rho(A_s)$.
   (e) If $\rho(A_s) = 0$ and $\rho(A_r) > 0$ set $\rho(A_s) := \rho(A_r)$, and remove from CA all arcs (s,q) or (q,s) such that $\rho(A_q) = \rho(A_r)$.
   (f) If $\rho(A_r) > 0$ and $\rho(A_s) > 0$, remove from CA all arcs (p,q) such that $\rho(A_p) = \rho(A_r)$ or $\rho(A_s)$ and such that $\rho(A_q) = \rho(A_r)$ or $\rho(A_s)$ (but $\rho(A_q) \neq \rho(A_p)$), and relabel all points A_p such that $\rho(A_p) = \rho(A_s)$ by setting $\rho(A_p) := \rho(A_r)$.
3. If $CA = \varnothing$, stop. Otherwise return to Step 1.

The preceding method can be undergo additional pre-processing to remove arcs of the initial CA that would create the termination $D(A_r,A_s) > Pre\_D + \Delta$ of Step 2. (The value UL can be reduced by taking account of this termination to remove such arcs, if UL is not already small enough.) Also, the removal of arcs from CA can be effected by reference to lists of arcs (p,q), for $p < q$, associated with each node p. (Such a list is sometimes called the forward star of p, or the list of neighbors of p.)

When the Cluster Method stops, each value $\rho = 1, \ldots, \rho_o$ such that $\rho(A_p) = \rho$ for at least one $A_p \in FB$ identifies a cluster (connected component of the Digraph DG) that consists of all A_p such that $\rho(A_p) = \rho$.

The method to create Buffer regions then operates as follows.

Buffer Method

0. Start with $FB_o = FB$, and set $d' = d_2 + 1$.
1. Select a cluster *CL*, not previously chosen, that has been created by the preceding Cluster Method or created in Step 2 below, such that $CL \cap FB_o \neq \varnothing$. If no such cluster exists, stop.
2. Optimize the function $F_o(G_k(d_1), CL)$ to create new regions $R_k(d')$, $R_{k*}(d')$ subject to $R_k(d') \supseteq G_k(d_1)$, designed (as the formulation (2) of Section 2.3) to yield a first level retrospective enhancement if a perfect separation of $G_k(d_1)$ and *CL* can be achieved. If such a separation does not result, remove from *CL* all of its points that lie in region $R_k(d')$, designating them to constitute a new cluster to be examined in Step 1. Then repeat Step 2 for the current reduced *CL*.
3. Remove from FB_o all points A_p that lie in $R_{k*}(d')$ and set $d' := d' + 1$. Then return to Step 1.

The reference to formulation (2) in Step 2 is motivated by the observations in Section 2.3 about the uses of formulation (2) in contexts where a perfect separation may likely be achieved. In the Buffer Method, the goal is primarily to maximize the minimum separation between the groups, and hence the coefficient k_o would be given a value that dominates the values of the coefficients k_i in this formulation.

The case where a perfect separation does not result in Step 2 of the Buffer Method entails only one repeat of this step, since the reduced *CL* is assured to yield such a separation. However, a better approach would be to identify a smaller value for Δ and repeat the Cluster Method to produce a more appropriate set of clusters. Such a Δ value can be implicitly identified by instead selecting a smaller UL value, which can be chosen to assure that the removal of arcs (p,q) having $D(A_p,A_q) > UL$ from the connected component defining the cluster *CL* of Step 2 will produce new connected components. In fact this step can be streamlined. Whenever the Cluster Method is re-run by reducing either Δ or UL (and without increasing either of these values), then we may restrict attention to a starting CA list that consists simply of the arcs of the final DG produced on the preceding pass. Since these arcs are already ordered by the first pass, they do not need to be ordered again.

The index d′ of the new regions $R_k(d')$ and $R_{k^*}(d')$ starts at $d_2 + 1$ for the reason previously specified and to avoid confusion with other regions previously generated. There is no concern that d' may exceed the limit d_o, because buffer regions improve robustness rather than diminish it. (The larger value of d' does not signify that the method is carried to a greater depth in the customary meaning of depth.)

We observe that the region $R_{k^*}(d')$ normally does not contain points outside of C^*, i.e., we do not expect to enlarge the set of correctly classified points in C^* by adding the region $R_{k^*}(d') \cap G_{k^*}(d_1)$ to C^*. Such an outcome may occur gratuitously, however, since on the second pass the group $G_k(d_1)$ that is required to be encompassed within the region $R_{k^*}(d')$ may be reduced as a result of removing the misclassified points of the first pass.

The new buffer regions have a second benefit, because they contribute to the ability to create a further improved and more robust separation by a process of modifying the original regions $R_{k^*}(d)$ for $d \in [d_1,d_2]$. We examine this process next.

3.2.2 Refining Regions Previously Created.

Upon applying the Buffer Method in the case where *FB* is not empty, we identify a possibly larger version C_o^* of C^* given by $C_o^* = C^* \cup (C_{k^*}(d') : d_2 < d' \leq d'')$, where d″ is the largest value of d′ values produced by the Buffer Method, and where the regions $C_{k^*}(d') = R_{k^*}(d') \cap G_{k^*}$ identify the points that are correctly classified by these new $R_{k^*}(d')$ (noting that all of these new regions may possibly lie in C^*).

If we remove any previous region $R_{k^*}(d)$ for $d \in [d_1,d_2]$ from the classification process, then the remaining regions correctly classify the set of points $C_o^*(d)$, which we define to be the union of all sets defining C_o^* except for the set $C_{k^*}^o(d)$, hence $C_o^*(d) = \cup(C_{k^*}^o(dd) : d_1 \leq dd \leq d_2, dd \neq d) \cup (C_{k^*}(d') : d_2 < d' \leq d'')$. It is possible that $C_o^*(d) = C_o^*$, so that the region $R_{k^*}(d)$ is redundant for the purpose to classifying all points of C_o^* correctly.

The possibility of improvement comes from the fact that the original determination of $R_{k^*}(d)$ $(= R_{k^*}^o(d))$ was based on trying to bring a significant portion of the not-yet-correctly-classified points of G_{k^*} into the correctly classified domain by encompassing them within $R_{k^*}(d)$. However, we now know that we only need to make sure that $R_{k^*}(d)$ encompasses the points of $C_o^* - C_o^*(d)$, which in general will be appreciably smaller than the group $G_{k^*}(d)$ that $R_{k^*}(d)$ originally sought to encompass. Moreover, we know that within $R_{k^*}^o(d)$ already contains all of $C_o^* - C_o^*(d)$, so there quite likely exists a new region $R_{k^*}(d)$ that can replace $R_{k^*}^o(d)$ and create a better separation between $G_k(d_1)$ and $C_o^* - C_o^*(d)$ than $R_{k^*}^o(d)$ does.

The following straightforward method generates such substitute (refined) regions.

Refinement Method.
0. Start with dSet = the set of integers $d \in [d_1,d_2]$.
1. If dSet is empty, stop. Otherwise, select and remove some d from dSet.

2. If $C_o^* - C_o^*(d) \neq \varnothing$, optimize a function $F'(G_k(d_1), C_o^* - C_o^*(d))$ that is designed, as the formulation (1^o) of Section 1.5, to yield a first level retrospective enhancement, and denote the regions produced by $R_k(d)$ and $R_{k^*}(d)$. Then redefine $R_{k^*}^o(d) := R_{k^*}(d)$ (which thus may change the composition of C_o^* and of $C_o^*(d)$ for other indexes d), and return to Step 1.
3. If $C_o^* - C_o^*(d) = \varnothing$, remove $R_{k^*}(d)$ (and its complement $R_k(d)$) from the set of regions used to classify points of G (thus likewise changing the composition of C_o^* and of $C_o^*(d)$ for other indexes d), and return to Step 1.

Typically, the choice of the element d to remove from dSet in Step 1 is to start with $d = d_2$ and work backward to d_1. In Step 3, rather than simply remove a region $R_{k^*}(d)$ from being used to classify elements of G, we may instead seek to "cover" its removal by executing a further iteration of the Buffer Method, to see if there is a better way to separate points now that this set is gone. In this case the Buffer Method can be simplified by choosing the limiting value UL of the Cluster Method to be small. As previously noted, when UL is reduced we can perform a new pass of the Cluster Method by starting with a CA list produced by the final DG of the preceding pass.

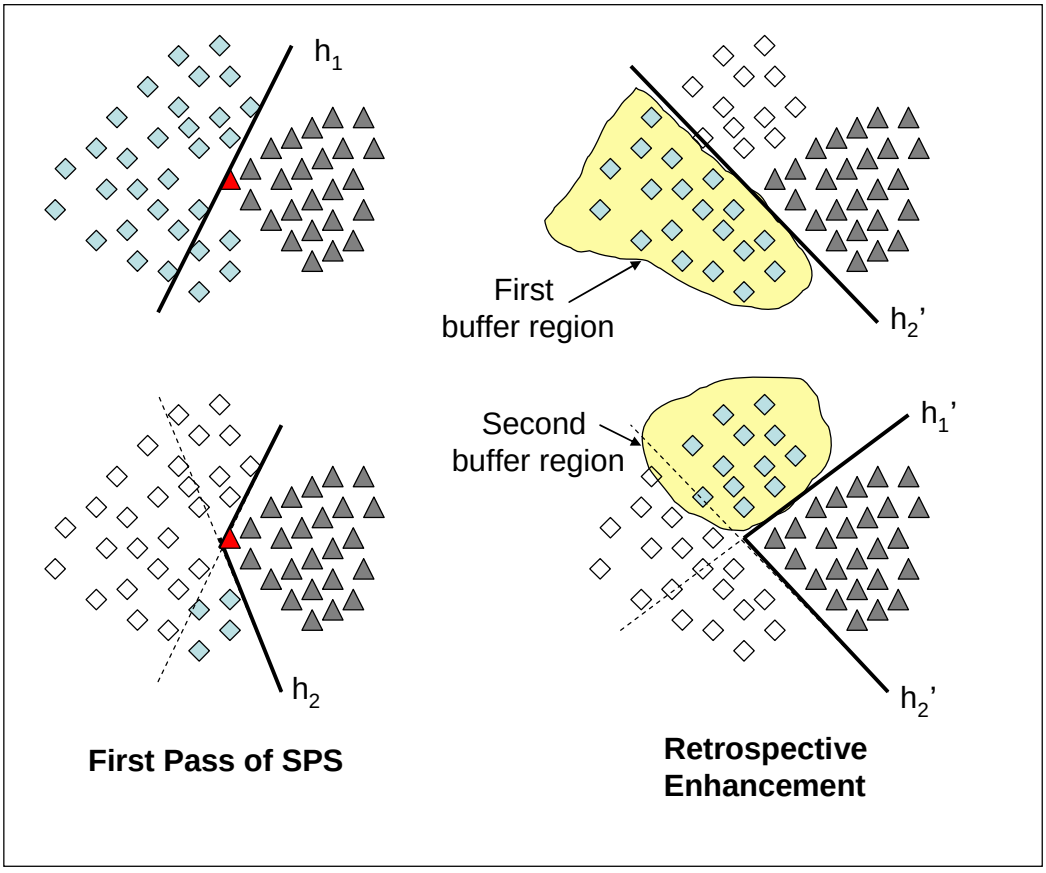

Figure 5: schematic representation of the retrospective enhancement procedure

Figure 5 shows a schematic representation of the retrospective enhancement procedure. Moving downward from the top left corner, the first graph shows hyperplane h_1 obtained

by a first SPS iteration; the second shows h_2 obtained by a second SPS iteration, resulting in a perfect separation of all the elements. However, this separation is not as robust as would be desired, given the precarious position of several elements that lie extremely close to the boundary – like the triangle highlighted near the intersection of h_1 and h_2, for instance. The graph of the top right corner shows a buffer region based on h_2 (ignoring h_1 for now) that allows for a more robust separation of the sets by a new hyperplane h_2' that replaces h_2. Finally, a second buffer region is used to obtain h_1', which replaces h_1. The result is a more robust boundary that better separates both groups. Later sections make it possible to take advantage of the buffering and refinement processes of retrospective enhancement by employing a mixed integer programming model to produce multiple hyperplanes to separate the groups at each stage.

4. Mixed Integer Programming Formulations

4.1 A Basic Mixed Integer Programming Model

As a first step toward introducing more advanced mixed integer models, we begin by examining a simple model to minimize the number of misclassified points by means of a single hyperplane.

Let z_i denote a 0-1 integer variable that takes the value 1 if the point A_i is misclassified and takes the value 0 otherwise. The following model (Glover, 1993) seeks to minimize the sum of the z_i variables, and hence to minimize the number of misclassified points:

$$\text{Minimize } \sum (z_i: i \in G) \tag{4.1}$$

subject to

$$A_i x - Mz_i + s_i = b, \quad i \in G_1 \tag{4.2}$$

$$A_i x + Mz_i - s_i = b, \quad i \in G_2 \tag{4.3}$$

$$x, b \text{ unrestricted} \tag{4.4}$$

$$z_i \in \{0,1\}, i \in G \tag{4.5}$$

$$(m_1 \sum (A_i: i \in G_2) - m_2 \sum (A_i: i \in G_1))x = 1 \tag{4.6a}$$

$$m_1 \sum (s_i - Mz_i: i \in G_2) + m_2 \sum (s_i - Mz_i: i \in G_1) = 1 \tag{4.6b}$$

Note that (4.2) and (4.3) express the inequalities $A_I x - Mz_i \leq b$ and $A_I x + Mz_i \geq b$, to which we have added slack variables s_i to convert the inequalities into equations. The constant M in this formulation takes a "large value" to assure that the associated inequality will be redundant whenever $z_i = 1$, and the quantities Mz_i take a role analogous to that of the v_i variables in the associated linear programming formulation of (1). (This connection is also evident upon comparing the normalization constraint (4.6b) to (1.6b).

Several related mixed integer formulations have also been proposed in the literature, including those of Stam and Ragsdale (1992), Abad and Banks (1993) and Glen (1999, 2003). However, the alternative formulations have various deficiencies, and generally require more variables. For example, one of the more effective formulations, due to Glen

(1999, 2003),[5] nevertheless requires doubling the number of x variables by splitting each x_j into a positive and negative part, and then producing a normalization constraint that sets the sum of these variables to 1. The classic approach of replacing unrestricted variables by the difference of two non-negative variables has merit in the context of mixed integer models for feature selection, as discussed in Section 4.3, because of the susceptibility of these latter models to a parametric solution approach. (When using such a replacement in the present context, we recommend an alternative normalization that weights the sum of non-negative variables in a given group by the cardinality of the opposite group, by analogy to the normalization indicated in (1.6a) and (1.6b).) An interesting study of credit scoring applications by Falangis (2006) supports Glen's demonstrations that his model achieves useful outcomes.

Several heuristics for dealing with these alternative mixed integer formulations are also proposed in the references cited. Special heuristics have likewise been proposed for solving formulation (4), making use of theorems in Glover (1993) that establish relationships between the mixed integer formulation and its linear programming relaxation.[6]

We now make the present model (4) more complete, by marrying it with additional considerations derived from the previous linear programming models. To do this we consider a means to weight the s_i variables as in formulation (1) by coefficients k_i and also to explicitly make use of the comprehensive satisfaction variable s_o, as introduced in formulation (1′). We do not include a v_o variable in this formulation, though we observe that this can be done without requiring that v_o be replaced by an associated integer variable.

To allow the formulation to work as intended, the variables s_i themselves cannot be directly weighted by coefficients k_i, but rather must be assigned these weights indirectly. The reason for this derives from the fact that assigning an integer variable z_i a value of 1 causes the slack variable s_i to receive an exceedingly large value, when in fact s_i should be treated as receiving a value of 0 in this case. This conclusion arises by observing that the constraints made redundant by $z_i = 1$ identify points A_i that fall on the wrong side of the hyperplane, and we must exclude these points from consideration in seeking to weight the separations for the correctly classified points.

We can produce the desired effect by introducing continuous variables t_i bounded so that t_i will be permitted to equal s_i only if the associated constraint is binding, i.e., only when $z_i = 0$. Otherwise, when $z_i = 1$, t_i will have an upper bound of 0, so that the value of s_i will not affect the objective when the point A_i is misclassified. Under the assumption that $z_i =$

[5] Glen re-phrases the problem as one of maximizing the number of correct classifications, which formulation (3) above handles automatically by complementing each of the 0-1 z_i variables, replacing it by the variable $y_i = 1 - z_i$.

[6] The ability to take advantage of such relationships by means of a heuristic approach has not yet been explored in a computational study.

0 for at least one $i \in G$ it is not necessary to introduce a corresponding variable t_o associated with s_o.

Let U_i be an upper bound on the value of the associated variable s_i in the situation where the point A_i lies in the desired half-space ($z_i = 0$). We weight the z_i variables by a large value M_o, to be sure the objective of minimizing their sum remains the dominant goal of the model. Then the mixed integer model becomes

$$\text{Minimize } M_o\sum (z_i: i \in G) - \sum (k_i t_i: i \in G) - k_o s_o \qquad (4.1')$$

subject to

$$A_i x - Mz_i + s_i + s_o = b, \quad i \in G_1 \qquad (4.2')$$

$$A_i x + Mz_i - s_i - s_o = b, \quad i \in G_2 \qquad (4.3')$$

$$x, b \text{ unrestricted} \qquad (4.4')$$

$$z_i \in \{0,1\}\ i \in G \qquad (4.5')$$

$$(m_1\sum (A_i: i \in G_2) - m_2\sum (A_i: i \in G_1))x = 1 \qquad (4.6a')$$

$$m_1\sum (s_i - Mz_i: i \in G_2) + m_2\sum (s_i - Mz_i: i \in G_1) + ms_o = 1 \qquad (4.6b')$$

$$t_i \le s_i, \quad t_i \le U_i(1 - z_i), \quad i \in G \qquad (4.7')$$

$$s_o \ge 0, s_i \ge 0, t_i \ge 0, \ i \in G \qquad (4.8')$$

The inequality $t_i \ge 0$ in (4.8′) is redundant under the assumption that the k_i coefficients are positive and hence this inequality may be discarded from the model if desired.

Studies of the foregoing model by means of a sequential perfect separation approach in Better et al. (2006) have demonstrated its effectiveness in a cancer diagnosis application and a Japanese bank application.

4.2 Improved Mixed Integer Programming Formulation

To complete the foundation for creating an effective multi-hyperplane model, we first identify a variation of the previous model that is more suited to our needs. Our proposed variation has the same number of variables and constraints as the model (4′).

We produce the new formulation by combining the model of (4) with the original linear programming formulation of (1). This reintroduces the v_i variables back into the problem, and we accompany these variables with additional constraints to compel the 0-1 z_i variables to take appropriate values.

$$\text{Minimize } M_o\sum (z_i: i \in G) + \sum (h_i v_i - k_i s_i): i \in G) - k_o s_o \qquad (4.1'')$$

subject to

$$A_i x - v_i + s_i = b, \quad i \in G_1 \qquad (4.2'')$$

$$A_i x + v_i - s_i = b, \quad i \in G_2 \qquad (4.3'')$$

$$x, b \text{ unrestricted} \qquad (4.4'')$$

$$z_i \in \{0,1\}\ i \in G \qquad (4.5'')$$

$$(m_1\sum (A_i: i \in G_2) - m_2\sum (A_i: i \in G_1))x = 1 \qquad (4.6a'')$$

$$m_1\sum (s_i - v_i: i \in G_2) + m_2\sum (s_i - v_i: i \in G_1) = 1 \qquad (4.6b'')$$

$$v_i \le U_i z_i,\ \ s_o \le s_i + U_o z_i,\ \ i \in G \qquad (4.7'')$$
$$s_o \ge 0,\ s_i \ge 0,\ v_i \ge 0,\ \ i \in G \qquad (4.8'')$$

The constant U_o in (4.7″) is an upper bound on s_o, hence on the minimum (not maximum) value that can be received by the s_i variables that take on positive values. Because of the ability to make U_o relatively small, the model has advantages when subjected to a mixed integer programming solution procedure.

The two inequalities $v_i \le U_i z_i$ and $s_o \le s_o \le s_i + U_o z_i,\ \ i \in G$ of (4.7″) replace the associated inequalities $t_i \le s_i$ and $t_i \le U_i(1 - z_i),\ \ i \in G$, of (4.7′), though the inequalities of (4.7″) perform a different function than those of (4.7′). The t_i variables of the previous formulation are not needed in the present one. Effectively, formulation (4″) may be seen as a mixed integer extension of formulation (1″).

We can in fact reduce the number of inequalities in (4″) by stipulating that the inequalities of $s_o \le s_i + U_o z_i,\ \ i \in G$ of (4.7″) are only included for $i \in G_1$ or for $i \in G_2$, according to which of G_1 or G_2 has a smaller number of elements. The variable s_o, which serves the purpose of increasing the minimum separation of the satisfied points from the hyperplane boundary, now applies only to the satisfied points of one of the two groups. However, this use of s_o is equivalent to the previous one, since we can simply re-define b after solving the model where s_o is changed as indicated by setting $b := b + .5s_o$, and we see that s_o receives just twice the value it would receive in the formulation where s_o is left unchanged. (Similarly, it would be possible to include s_o only in one of the equations (4.2′) or (4.3′) in formulation (4′). However, this does not change the number of constraints in that formulation.) We will make additional use of this observation about associating s_o with only one of the two groups in our later development.

We suggest a variation of model (4″) that replaces the term $\sum(z_i: i \in G)$ in the objective (4.1″) by the term $m_2\sum(z_i: i \in G_1) + m_1\sum(z_i: i \in G_2)$. This variation seeks a "more balanced" solution that weights the sums of violations so that they will tend to be more nearly proportional to the numbers of elements in each of the two groups.

A particular advantage of model (4″) over model (4′) is that it remedies the possibility that the normalization constraint in (4′) can become unstable under solution by a branch and bound method. The potential for instability is subtly hidden, but can be recognized by noting that the effect of setting z_i values to 0 and 1 in (4′), as done at various points in applying a branch and bound method, eliminates the associated points A_i from having any effect on the model. This effectively removes these points from their associated sets G_1 or G_2, and thus causes the normalization constraint to become distorted because the values of m_1 and m_2 representing the numbers of elements in these sets remain constant. This situation where an integer programming model can become changed by the process of solving it is highly unusual! The ability to avoid it by formulation (4″) is accompanied by other advantages of this model when it is extended to create a multi-hyperplane formulation.

4.2.1 Exploiting Model (4″) by a Fixed Charge Interpretation.

Model (4″) offers an additional advantage due to the fact that it can be re-interpreted as a fixed charge model. In particular, we can remove the z_i variables altogether, and view the v_i variables as fixed charge variables, stipulating that the associated fixed charge has the value M_o for each of these variables (incurred for a given v_i if and only if it is positive). The merit of this interpretation derives from the fact that a highly effective heuristic is available for fixed charge problems based on the framework of *ghost image processes* (Glover, 1994). A study in the context of fixed charge generalized network problems (Glover, Amini and Kochenberger, 2005) discloses that this approach obtains optimal solutions for all problems capable of being solved optimally by the state-of-the-art CPLEX MIP software, and produces solutions substantially superior to those obtained by this software for problems too large or too difficult for CPLEX to solve to optimality when allowed to run more than 100 times longer than the ghost image procedure.[7] The fixed charge implementation of the study is not specific to the network setting, and hence can readily be adapted to the present context, affording an opportunity to solve problems of much greater size than would normally be possible by the mixed integer formulation.

4.3 Feature Selection

A formulation closely related to (4″) can be used to model the problem of feature selection, where the goal is to identify a limited number of attributes (features) that can yield an effective differentiation between elements of G_1 and G_2. The importance of the feature selection problem stems from the need to reduce the number of features considered when dealing with large data bases, and also from the fact that restricting the number of attributes often proves valuable for combating the "overfitting" problem, and thereby yields superior outcomes when seeking to classify elements of hold-out samples.

4.3.1 Elements of the Feature Selection Formulation

A common formulation of the feature selection problem consists of imposing an upper bound on the number of attributes permitted to be used, and then seeking the best differentiation between the groups subject to this bound. In our setting, the objective can be viewed as one of finding a best separating hyperplane subject to bounding the number of components of x that are allowed to be non-zero.

To provide a mixed integer programming model for this problem, we employ the device of introducing a binary variable w_j where $w_j = 1$ if x_j is allowed to be non-zero and $w_j = 0$ otherwise. Let $N = \{1, \ldots, n\}$ and let $U(w)$ represent the chosen upper limit on the number of x_j for $j \in N$ that are permitted to be non-zero. Then the bound can be enforced by means of the inequality

$$\sum (w_j : j \in N) \leq U(w).$$

[7] After obtaining solutions superior to those found by CPLEX on all instances of this testbed, no attempt was made to run CPLEX on remaining problems of corresponding size and difficulty.

The MIP formulation that captures the connection between the w_j variables and the x_j variables can be completed in two ways. The first identifies upper and lower bounds U_j and L_j for each x_j. (Note that L_j is generally negative, and the bound U_j is of course not to be confused with the bound U_i for v_i in sections 4.1 and 4.2.) The condition that compels x_j to be 0 when w_j is 0, and otherwise allows x_j to be bounded by U_j and L_j, can then be written as

$$U_j w_j \geq x_j \geq L_j w_j, \ \ j \in N.$$

It is possible to avoid using the inequalities $x_j \geq L_j w_j$ by replacing each x_j for $j \in N$ with the variable $x_j' \equiv x_j - L_j w_j$, which is assured to be non-negative. In particular, substituting for $x_j \equiv x_j' + L_j w_j$ in the inequality $U_j w_j \geq x_j \geq L_j w_j$ yields $U_j' w_j \geq x_j' \geq 0$, where $U_j' \equiv U_j - L_j$. (This effectively reduces the number of inequalities of the problem, since the non-negativity of the x_j' variables is handled implicitly by standard LP and MIP solvers.) Then the original x_j values are recovered from $x_j \equiv x_j' + L_j w_j$ in the final solution.

The second way to create a suitable MIP formulation makes use of the familiar device of replacing x_j by the difference of two non-negative variables (see, e.g., Charnes and Cooper, 1961 and Dantzig, 1963). Denoting these variables by x_j^+ and x_j^-, we write $x_j = x_j^+ - x_j^-$. Then by the definitions of U_j and L_j we can compel these variables to be 0 when w_j is 0 by introducing the constraints

$$x_j^+ \leq U_j w_j \text{ and } x_j^- \leq -L_j w_j.$$

Alternatively, by defining $U_j^o = \text{Max}(U_j, -L_j)$,[8] we may introduce the single constraint

$$x_j^+ + x_j^- \leq U_j^o w_j.$$

Finally, to assure at most one of x_j^+ and x_j^- will be positive, we let c_j and d_j be small positive coefficients, and augment the minimization objective of the MIP formulation to include the term

$$\sum (c_j x_j^+ + d_j x_j^- : j \in N).$$

We allow c_j and d_j to be different from each other as the basis for solving the feature selection MIP formulation by a parametric penalty approach, as described in Appendix 1. It may be noted that the formulation devices underlying the present feature selection model can also be used in feature selection for L1 regression models, and the resulting optimization problem can similarly be solved using the approach sketched in the accompanying appendix.

[8] Of course, U_j^o may simply be taken to be a "large" value without first trying to estimate the values of U_j and L_j.

5. Mixed Integer Programming Formulations for Multi-Hyperplane Separation

Our multi-hyperplane separation approach expands model (4″) by introducing a collection of half spaces denoted by

$$A_i x^p \leq b^p \text{ and } A_i x^p \geq b^p \text{ for } p \in P = \{1, 2, \ldots, p_o\}.$$

In practice, the value of p_o (hence the size of P) can be kept small, e.g., at most 3 or 4, because of the ability to use our mixed integer model within the framework of successive separation, and more particularly within the framework of successive perfect separation. By means of the SPS framework, and by taking advantage of retrospective enhancement, the multi-hyperplane model offers useful advantages even for p_o as small as 2.

It is possible to generate complex collections of subspaces composed of various intersections and unions of the half-spaces $A_i x^p \leq b^p$ and $A_i x^p \geq b^p$, and to produce a mixed integer formulation that assigns the points of Group 1 and Group 2 to specified instances of these subspaces. A general design for creating such formulations is given in Appendix 2. However, by designing our present muli-hyperplane approach to be used in conjunction with the SPS strategy, we focus on two simple types of separation that exhibit useful structures and that encompass a variety of more complex alternatives by the compounding effect of the tree-based derivations. As a means of solving these models, we recommend the parametric approach of Glover (2006a).

5.1 All Union versus All Intersection Separating Conditions

The first type of separation we consider requires all points of the primary SPS group to lie in the union of the half-spaces $A_i x^p \leq b^p$, $p \in P$, while all points of the secondary SPS group lie in the complementary region consisting of the intersection of the half-spaces $A_i x^p \geq b^p$, $p \in P$. Strictly speaking, for these two regions to be complementary, the latter one should be formed from the intersection of the open half-spaces $Ax^p > b^p$, $p \in P$. We undertake to assure this, and simultaneously to induce points of the secondary group to lie as far inside this space as possible, by making use of the internal deviation variable s_o and by following up with the retrospective enhancement procedure. Glen (1999) was the first to propose a mixed integer model for this initial case we examine, although using a somewhat different framework and without accounting for internal deviations.

In the general case of multi-hyperplane separations discussed in the Appendix 2, it is not possible to restrict Group 1 points to be associated only with inequalities of the form $A_i x^p \leq b^p$ and to restrict Group 2 points to be associated only with inequalities of the form $A_i x^p \geq b^p$. However, this beginning special case we are concerned with constitutes an exception.

For convenience, we suppose that we represent the primary group as Group 1 and the secondary group as Group 2. To switch these groups, we correspondingly switch the labels of Group 1 and Group 2.

Then we formulate our goals as follows.

(a) For each $i \in G_1$: compel $A_ix^1 \le b^1$ or $A_ix^2 \le b^2$ or … or $A_ix^{po} \le b^{po}$
(b) For each $i \in G_2$: induce $A_ix^1 \ge b^1$ and $A_ix^2 \ge b^2$ and … and $A_ix^{po} \ge b^{po}$

The terminology of "inducing" the condition expressed in (b) means that we seek to minimize the number of violations of this condition, subject to enforcing condition expressed in (a).

To achieve these goals, we transform the half-space inequalities into equations in the usual manner by writing

$$A_ix^p - v_i^p + s_i^p = b^p, \quad i \in G_1, \quad p \in P$$
$$A_ix^p + v_i^p - s_i^p = b^p, \quad i \in G_2, \quad p \in P$$

Similarly, we introduce 0-1 variables z_i^p, $p = 1, \ldots, p_o$, where $z_i^p = 1$ if $v_i^p > 0$ and $z_i^p = 0$ if $v_i^p = 0$. Then (a) and (b) are respectively equivalent to the following conditions.

(a1) For each $i \in G_1$: compel $z_i^p = 0$ for at least one element $p \in P$; or equivalently compel $z_i^p = 1$ for at most $p_o - 1$ elements $p \in P$.
(b1) For each $i \in G_2$: induce $z_i^p = 0$ for all elements $p \in P$; or equivalently induce $z_i^p = 1$ for no elements $p \in P$.

These conditions are met by introducing non-negative continuous variables y_i, $i \in G_2$, and minimizing the sum of these variables in the objective function, subject to

(a2) For each $i \in G_1$: $\sum(z_i^p: p \in P) \le p_o - 1$.
(b2) For each $i \in G_2$: $y_i^p \ge z_i^p$ for each $p \in P$.

Condition (a2) can alternately be achieved by introducing non-negative continuous variables y_i for $i \in G_1$ that are given pre-emptively large weights for minimizing their sum in the objective function, subject to the inequalities $y_i^p \ge \sum(z_i^p: p \in P) - (p_o - 1)$, and condition (b2) can be achieved by instead imposing the constraint $p_o y_i^p \ge \sum(z_i^p: p \in P)$, although the latter yields a weaker LP relaxation of the mixed integer model.

Combining these observations with the considerations already discussed in relation to formulation (4″), we obtain the following mixed integer programming model:

$$\text{Minimize } M_o\sum(y_i: i \in G_2) + \sum(h_iv_i^p - k_is_i^p): i \in G_2, p \in P) - k_os_o \qquad (5.1)$$

subject to

$$A_ix^p - v_i^p + s_i^p = b^p, \quad i \in G_1, \quad p \in P \qquad (5.2)$$

$$A_i x^p + v_i^p - s_i^p = b^p, \quad i \in G_2, \ p \in P \tag{5.3}$$
$$x^p, b^p \text{ unrestricted}, \ p \in P \tag{5.4}$$
$$z_i^p \in \{0,1\} \ i \in G, \ p \in P \tag{5.5}$$
$$(m_1 \sum (A_i: i \in G_2) - m_2 \sum (A_i: i \in G_1)) x^p = 1, \ p \in P \tag{5.6a}$$
$$m_1 \sum (s_i^p - v_i^p: i \in G_2) + m_2 \sum (s_i^p - v_i^p: i \in G_1) = 1, \ p \in P \tag{5.6b}$$
$$v_i^p \le U_i z_i^p, \ i \in G, \ p \in P \tag{5.7}$$
$$s_i^p \ge 0, \ v_i^p \ge 0, \ i \in G, \ p \in P \tag{5.8}$$
$$s_o \ge 0, \ y_i \ge 0, \ s_o \le s_i + U_o y_i, \ i \in G_2 \tag{5.9}$$
$$\sum (z_i^p: p \in P) \le p_o - 1, \ i \in G_1 \tag{5.10}$$
$$y_i \ge z_i^p, \ i \in G_2, \ p \in P \tag{5.11}$$

Thus the formulation (5) contains roughly $p_o = |P|$ times the number of variables and constraints of formulation (4″). The advantage of keeping p_o small by exploiting the SPS framework is significant.

We have in fact incorporated more variables and constraints than necessary into the preceding formulation, in order to demonstrate a connection with more general formulations discussed in Appendix 2. In the present case, we can replace the p_o integer variables z_i^p, $p \in P$ for each $i \in G_2$, by a single integer variable y_i, for each $i \in G_2$ (i.e., upon removing the indicated zero-one variables z_i^p, the variable y_i is changed from being continuous to being a zero-one variable). Then (5.5) and (5.7) are reduced by eliminating reference to variables z_i^p and associated inequalities over $i \in G_2$, to become

$$z_i^p \in \{0,1\}, \ i \in G_1, \ p \in P \tag{5.5a}$$
$$v_i^p \le U_i z_i^p, \ i \in G, \ p \in P \tag{5.7a}$$

Accompanying this, the non-negativity condition for y_i in (5.9) is replaced by the zero-one condition, and (5.11) shrinks to directly relate the y_i variables to the continuous violations v_i^p, replacing p_o inequalities for each $i \in G_2$ by a single inequality to produce

$$s_o \ge 0, \ s_o \le s_i + U_o y_i, \ y_i \in \{0,1\}, \ i \in G_2 \tag{5.9a}$$
$$U y_i \ge \sum (v_i^p: p \in P), \ i \in G_2 \tag{5.11a}$$

where U is an upper bound on the sum of the violations on the right of (5.11a). It is clear that this more economical formulation achieves the same result as the formulation without these replacements.

Either with or without these replacements, the goal of minimizing the sum of the y_i for $i \in G_2$ is given a large weight M_o in the objective (5.1) to allow it to dominate the other parts of the objective. The term $\sum (h_i v_i^p - k_i s_i^p): i \in G_2, p \in P)$ is included for generality but may be omitted. The inclusion of $- k_o s_o$ in the objective is relevant for reasons previously discussed. The size of M_o may be decreased (though it must be maintained greater than k_o) to allow a trade-off where a small number of additional points of Group 2 may be allowed to fall outside of the region targeted for this group, in exchange for allowing satisfied points to be separated more cleanly from unsatisfied points.

An alternative approach to incorporating such trade-offs is to solve formulation (5) first in the form shown, and then to impose a bound on $\sum (y_i: i \in G_2)$ so as not to overshoot the optimum value of this sum by more than a stipulated amount. Then the term $\sum (y_i: i \in G_2)$ can be removed from the objective function on a follow-up solution pass that seeks an optimal solution for the residual objective.

The application of formulation (5) in the sequential perfect separation approach occurs by employing this formulation in place of formulation (2) in section 2.3, and otherwise following the prescriptions of the SPS method. Similarly, the general form of retrospective enhancement can take direct advantage of formulation (5).

One subtlety introduced by formulation (5) is that it implicitly gives rise to four different ways to create a distinction between the primary and secondary sets, rather than only two ways as in the case where the regions of interest consist simply of half-spaces. Because the regions generated by (5) are not symmetric (as half-spaces are), we can create a different model by reversing which conditions are compelled and which are induced in (a) and (b), thus producing

(a1) For each $i \in G_1$: induce $A_ix^1 \le b^1$ or $A_ix^2 \le b^2$ or … or $A_ix^{po} \le b^{po}$
(b2) For each $i \in G_2$: compel $A_ix^1 \ge b^1$ and $A_ix^2 \ge b^2$ and … and $A_ix^{po} \ge b^p$

The change in formulation (5) to handle (a1) and (a2) is apparent.

5.2 Additional Structure for the Case of $p_o = 3$.

Because of the relevance of focusing on cases where p_o is small, we examine next a special case for $p_o = 3$ that is a natural accompaniment of the structure examined in Section 5.1. In particular we seek the handle the situation where the primary and secondary sets are generated as shown in (a) and (b), following:

(a) For each $i \in G_1$: compel ($A_ix^1 \le b^1$ and $A_ix^2 \le b^2$) or ($A_ix^3 \le b^3$)
(b) For each $i \in G_2$: induce ($A_ix^1 \ge b^1$ and $A_ix^3 \ge b^3$) or ($A_ix^2 \ge b^2$ and $A_ix^3 \ge b^3$)

We describe the logic for treating this case in a more general form than used to identify the formulation in Section 5.1, and thus provide a means for generating additional cases. (Further details concerning this form of analysis are given in Appendix 2.)

Starting with (a) to define the conditions applicable to the primary SPS group, which we have expressed in disjunctive normal form, the conditions (b) applicable to the secondary SPS group are generated by complementing the primary group definition, and then re-expressing the result likewise in disjunctive normal form. To qualify as a disjunctive normal form expression, each condition is expressed as the union of a collection of intersections (as a disjunction of a series of conjunctions). We use the notation where the symbol p represents $A_ix^p \le b^p$ and the symbol * denotes complementation, hence p*

represents $A_i x^p > b^p$ (where we take the liberty of referring to this latter inequality in the relaxed form $A_i x^p \geq b^p$). Then we derive (b) from (a) as follows.

First, the primary condition identified in (a) corresponds to $(1 \cap 2) \cup 3$ (which is already expressed in disjunctive normal form to start). The complement of this expression is $(1 \cap 2)^* \cap 3^*$ and we put it likewise in disjunction normal form by the series of steps $(1 \cap 2)^* \cap 3^* = (1^* \cup 2^*) \cap 3^* = (1^* \cap 3^*) \cup (2^* \cap 3^*)$. The latter expression corresponds to (b) above.

The following rule models an expression in disjunctive normal form by creating an inequality that compels a variable y_i to be 1 if the expression is true and to be 0 if the expression is false. We have chosen the index i of y_i to associate this variable with a specific point A_i, following the notation used in the formulation of Section 4.1. Let $(g_1 \cap \ldots \cap g_r)$ be any component of the disjunctive normal form expression associated with the point A_i, and let z_i^p be the 0-1 variable such that $z_i^p = 1$ if and only if g_p is true. Then we write $y_i \geq z_i^1 + \ldots + z_i^r - (r - 1)$ for each of these components. (If r = 1 the resulting inequality is just $y_i \geq z_i^1$.) Evidently, $y_i = 1$ if $z_i^p = 1$ for all p = 1, …, r, and otherwise y_i may permissibly receive the value 0. Moreover y_i will automatically be 0 if the condition $y_i = 1$ is not compelled, because in the present context the variable y_i receives a positive weight in the objective function. Finally, in the case where we seek to compel a disjunctive normal form expression not to be true, we simply set $y_i = 0$ and generate the inequality $r - 1 \geq z_i^1 + \ldots + z_i^r$.

We illustrate this analysis by applying it to (a) and (b) above. First, to satisfy (a) we must prohibit its complement (b) from holding (i.e., we must prohibit any point A_i for $i \in G_1$ from satisfying (b)). By reference to the complement (b) of (a), expressed in disjunctive normal form, this gives rise to the two inequalities

$$y_i \geq z_i^1 + z_i^3 - 1 \text{ and } y_i \geq z_i^2 + z_i^3 - 1, \; i \in G_1.$$

where, for $i \in G_1$, z_i^p is the 0-1 variable that equals 1 if and only if the inequality $A_i x^p \leq b^p$ is violated, and hence $A_i x^p > b^p$ is satisfied. Since we wish to compel (b) not to hold, we set $y_i = 0$ in these two inequalities to yield

$$1 \geq z_i^1 + z_i^3 \quad \text{and} \quad 1 \geq z_i^2 + z_i^3, \; i \in G_1.$$

Next, starting with (b), which we want to induce by penalizing its violation, we seek to avoid satisfying the complement expressed in (a), and hence we obtain the inequalities

$$y_i \geq z_i^1 + z_i^2 - 1 \text{ and } y_i \geq z_i^3, \; i \in G_2$$

where, for $i \in G_2$, z_i^p is the 0-1 variable that equals 1 if and only if the inequality $A_i x^p \geq b^p$ is violated, and hence $A_i x^p < b^p$ is satisfied. In this case we do not compel $y_i = 0$, but retain the two inequalities as shown and rely on penalizing y_i in the objective function

to yield $y_i = 0$ (achieving this outcome if it will result in minimizing the sum of the y_i variables over $i \in G_2$).

By means of this analysis we obtain the following formulation, for the specific conditions embodied in (a) and (b) preceding (and for $P = \{1,2,3\}$).

$$\text{Minimize } M_o\sum (y_i: i \in G_2) + \sum (h_i v_i^p - k_i s_i^p): i \in G_2, p \in P) - k_o s_o \qquad (5.1')$$

subject to

$$A_i x^p - v_i^p + s_i^p = b^p, \quad i \in G_1, \ p \in P \qquad (5.2')$$

$$A_i x^p + v_i^p - s_i^p = b^p, \quad i \in G_2, \ p \in P \qquad (5.3')$$

$$x^p, b^p \text{ unrestricted}, \ p \in P \qquad (5.4')$$

$$z_i^p \in \{0,1\} \ i \in G, \ p \in P \qquad (5.5')$$

$$(m_1\sum (A_i: i \in G_2) - m_2\sum (A_i: i \in G_1))x^p = 1, \ p \in P \qquad (5.6a')$$

$$m_1\sum(s_i^p - v_i^p: i \in G_2) + m_2\sum(s_i^p - v_i^p: i \in G_1) = 1, \ p \in P \qquad (5.6b')$$

$$v_i^p \le U_i z_i^p, \ i \in G, \ p \in P \qquad (5.7')$$

$$s_i^p \ge 0, \ v_i^p \ge 0, \ i \in G, \ p \in P \qquad (5.8')$$

$$s_o \ge 0, \ y_i \ge 0, \ s_o \le s_i + U_o y_i, \ i \in G_2 \qquad (5.9')$$

$$1 \ge z_i^1 + z_i^3 \text{ and } 1 \ge z_i^2 + z_i^3, \ i \in G_1 \qquad (5.10')$$

$$y_i \ge z_i^1 + z_i^2 - 1 \text{ and } y_i \ge z_i^3, \ i \in G_2 \qquad (5.11')$$

As can be seen, this formulation is identical to that of (5), except for the inequalities of (5.10′) and (5.11′) which we have derived from the analysis of this section.

We examine one more case for $p_o = 3$ to conclude our analysis. In this instance we begin with the requirement for the primary set given by

(a′) For each $i \in G_1$: compel ($A_i x^1 \le b^1$ or $A_i x^2 \le b^2$) and ($A_i x^3 \le b^3$)
(b′) For each $i \in G_2$: induce the complement of (a) to hold

We have not yet stipulated the precise condition that expresses (b′) for the secondary set, but derive it from the knowledge of (a′). To begin, (a′) is not stated in disjunctive normal form, so we proceed to put it in that form. By the notation used to derive formulation (5′) we write (a′) as $(1\cup2) \cap 3$, and transform it into disjunctive normal form by the sequence $(1\cup2) \cap 3 = (1\cap3) \cup (2\cap3)$ (which in this instance is completed by a single step). We then proceed to generate the complement of (a′) and to put it in disjunctive normal form. Complementing $(1\cup2) \cap 3$ yields $(1\cup2)^* \cup 3^* = (1^*\cap2)^* \cup3^*$, and the process is complete.

As it turns out, it is unnecessary to generate a new model from this outcome, as may be seen by comparing the conditions (a′) and (b′) with the previous conditions (a) and (b).

(a) For each $i \in G_1$: compel $(1\cap2) \cup3$
(b) For each $i \in G_2$: induce $(1^*\cap3^*) \cup (2^*\cap3^*)$

(a′) For each $i \in G_1$: compel $(1\cap3) \cup (2\cap3)$

(b′) For each i ∈ G_2: induce (1*∩2)* ∪3*

In particular, these two sets of conditions can be seen to be the opposite of each other, if we swap the role of the sets G_1 and G_2 and the direction of the inequalities associated with these sets. That is, it is only an arbitrary choice to associate G_1 with "less than or equal to" inequalities of the form $A_i x^p \leq b^p$ and to associate G_2 with "greater than or equal to" inequalities" of the form $A_i x^p \geq b^p$, since the mixed integer model can reverse these inequalities simply by reversing the sign of x and b. With this in mind, we can as readily write (a′) and (b′) in the form

(a′) For each i ∈ G_1: compel (1*∩3*) ∪ (2*∩3*)
(b′) For each i ∈ G_2: induce (1∩2) ∪3

Finally, by interchanging the roles of G_1 and G_2 to make G_2 the primary set and G_1 the secondary set, (a′) and (b′) give rise to the conditions

(a″) For each i ∈ G_2: compel (1∩2) ∪3
(b″) For each i ∈ G_1: induce (1*∩3*) ∪ (2*∩3*)

Thus (a″) and (b″) result simply by changing the choice of which set is primary and which is secondary in (a) and (b), and we don't need a new formulation to capture the conditions of (a′) and (b′). It would have been possible to have reached this conclusion even before generating (b′) as the complement of (a′), simply by noting that (a′) is the same as (b) with the symbol p replaced by the symbol p*.

References

Abad P.L and W. J. Banks (1993) “New LP based heuristics for the classification problem,” *European Journal of Operational Research*, Vol. 67, pp. 88-100.

Barr, R. and F. Glover (1993) "Adding Optimization to Recursive Partitioning: Machine Discovery of Quality Improvement Strategies," ORSA/TIMS National Convention, Chicago, May, 1993.

Barr, R. and F. Glover (1995) "LP-based Recursive Partitioning for High-Speed Machine Learning and Pattern Recognition," TIMS XXXIII, Singapore, June, 1995.

Better, M., F. Liang and M. Samorani (2006) "Linear and Integer Programming Classification Analysis with Decision Trees,” paper presented at the *DIMACS Workshop on Data Mining, Systems Analysis and Optimization in Neuroscience*, Gainesville, Florida.

Charnes, A. and W.W. Cooper (1961) *Management Models and Industrial Applications of Linear Programming*, Vol. 1, Wiley, New York.

Christiani, N. and J. Shawe-Taylor (2000) *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, Cambridge University Press, Cambridge, UK.

Dantzig, G. (1963) *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ.

Dai, H. (2004) *Advances in Knowledge Discovery and Data Mining*, Springer Publishing.

Falangis, K. (2006) "Testing the accuracy of the MP models in the credit environment,” *Proceedings of the 4th International Conference on Business, Economics, Management and Marketing – 2006*, Athens, Greece.

Freed, N. and F. Glover (1981) "A Linear Programming Approach to the Discriminant Problem," *Decision Sciences*, Vol. 12, No. 1, pp. 68-79.

Glen, J.J. (1999) “Integer programming methods for normalization and variable selection in mathematical programming discriminant analysis models,” *Journal of the Operational Research Society*, Vol. 50, pp. 1043-1053.

Glen, J.J. (2003) “An iterative mixed integer programming method for classification accuracy maximizing discriminant analysis,” *Computers and Operations Research,* 30/181-198.

Glover, F. (1990) "Improved Linear Programming Models for Discriminant Analysis," *Decision Sciences*, Vol, 21, No. 4, pp. 771-785.

Glover, F. (1993) "Improved Linear and Integer Programming Models for Discriminant Analysis," *Creative and Innovative Approaches to the Science of Management*, RGK Foundation Press, pp. 187-215.

Glover, F. (1994) "Optimization by Ghost Image Processes in Neural Networks," *Computers and Operations Research*, Vol. 21, No. 8, pp. 801-822.

Glover, F. (2006a) "Parametric Tabu Search for Mixed Integer Programs," *Computers and Operations Research*, Volume 33, Issue 9, pp. 2449-2494.

Glover, F. (2006b) "Topological Methods for Classification and Clustering," Leeds School of Business, University of Colorado, Boulder.

Glover, F., M.Amini and G. Kochenberger (2005) "Parametric Ghost Image Processes for Fixed-Charge Problems: A Study of Transportation Networks," *Journal of Heuristics*, Volume 11, Number 4, pp. 307-336.

Glover, F., S. Keene and R. Duea (1988) "A New Class of Models for the Discriminant Problem," *Decision Sciences*, Vol. 19, 269-280.

Ma, Y. and V. Cherkassky (2005) "Multiple Model Estimation for Nonlinear Classification," in *Support Vector Machines: Theory and Applications*, L. Wang, ed., pp. 49-76, Springer-Verlag, New York

Mangasarian, O.L. (1965) "Linear and Nonlinear Separation of Patterns by Linear Programming," *Operations Research* 13, pp. 444-452.

Mangasarian, O.L. and R. R. Meyer (1979) "Nonlinear Perturbation of Linear Programs," *SIAM Journal on Control and Optimization* 17(6), pp. 745-752.

Schlkopf, B. and A. J. Smola (2002) *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond (Adaptive Computation and Machine Learning)*, Cambridge, MA: MIT Press.

Stam, A. and Ragsdale, C.T. (1992) "On the classification gap in mathematical programming-based approaches to the discriminant problem." *Naval Research Logistics*, 39/545-559.

Wang, L. (2005) *Support Vector Machines: Theory and Applications*, Springer-Verlag, New York.

Appendix 1. A Parametric Process for Solving the MIP Formulation for the Feature Selection Problem.

The MIP feature selection formulation described in Section 4.3 may be superimposed on either a basic LP separating hyperplane formulation (e.g., the formulation (1″) or the SPS formulation (2)) or an MIP formulation such as (4″). In each case, the feature selection formulation is completed simply by augmenting the basic formulation to include the 0-1 variables w_j together with the constraints and objective function components previously indicated.

We denote the resulting feature selection formulation incorporating the w_j variables as P*(w), and denote the basic (LP or MIP) separating hyperplane formulation that is augmented to produce P*(w) by P*. Accompanying this, we let LP* denote the LP problem corresponding to P*; where LP* is the same as P* if P* is an LP formulation, and LP* is the linear programming relaxation of P* if P* is an MIP formulation.

A parametric penalty approach for the feature selection problem can be described by viewing the problem as that of shrinking N to yield a smaller set of variables indexed by N^o. We first provide some background observations before sketching how such an approach may operate.

One way to carry out the shrinking of N is to first solve the problem LP*, and then choose the reduced set N^o for the formulation P*(w) to consist only of those attributes j whose associated weights x_j receive non-zero values in this solution. The set N^o thus chosen may be smaller than is preferable, since a potentially better solution may be obtained by solving P*(w) over a superset of N^o (given there is no assurance that the best subset of U(w) variables permitted to be non-zero will be identified by N^o). Such a superset of N^o can be generated by post-optimal pivoting in LP*, selecting pivots to generate multiple solutions having the same or nearly the same objective value as the linear programming optimum. (Typically, dual degeneracy of LP* assures multiple optima exist.)

In the case where the LP* by itself is too large to solve readily, then smaller versions of it may be formed by a sequence of solution passes each of which operates with a different selected subset of N. The collection of these subsets may appropriately contain overlapping elements, as exemplified by the following constructive procedure: choose each subset to contain 1/3 of the elements of N. After selecting the first subset, choose each additional subset so that half of its elements comes from the part of N that has not yet been allocated, and the other half comes from a portion of the preceding subset that has not been shared with any other subset. (The last subset takes its elements from the unshared portions of the next-to-last subset and the first subset.) The illustrated construction thereby produces 6 subsets, and its general form that assigns n′/d elements to each subset produces 2d.different subsets when d is a positive integer.

Upon thus selecting subsets of N and solving LP problems over these subsets, N^o can be specified to consist of the union of the variables x_j that receive non-zero values in these

smaller problem instances. N^o can be made smaller by giving priority to variables that have received non-zero values in a larger number of sub-problem solutions. (In the 6 subset example, each variable will receive a non-zero value in at most two solutions.) Such a strategy can also be useful as a means of identifying particular variables that are likely to be important for generating robust solutions.

With these preliminaries, we briefly sketch how the MIP problem P*(w) itself can be solved by a parametric form of tabu search. (The following discussion assumes a basic understanding of the tabu search framework; see, e.g., Glover and Laguna, 1997.) The approach works strictly with the LP problem LP*, but written in the form that replaces the x_j variables by the x_j^+ and x_j^- variables, and that includes these variables in the objective function with coefficients c_j and d_j are previously indicated. This may be viewed as working with the form of P*(w) that replaces the x_j variables by the x_j^+ and x_j^- variables, but that discards all reference to the w_j variables and hence is simply a linear programming problem.

We call the resulting LP problem LP(c,d), since the coefficients c_j and d_j in this problem are the ones to be manipulated by the parametric TS solution procedure. For the initial form of LP(c,d) we let all c_j and d_j equal 0 (or, for implementation, a small positive value). Evidently, if c_j and d_j are chosen large enough, then x_j^+ and x_j^-, and hence x_j, will be driven to 0. The goal is to identify the "right" variables to receive non-zero (large) penalty values in LP(c,d), in order to drive some desired number of variables to 0 and yet obtain a good solution to the residual problem over the variables that are not penalized.

To initiate the tabu search method, the first variables to penalize by assigning them large c_j and d_j values can be those that receive the smallest positive values in the solution to LP*. Assigning penalties to any particular set of variables can cause new variables x_j^+ or x_j^- that are not yet penalized to receive positive values in turn.

An elementary TS recency memory can be used to manage the parameters c_j and d_j foridding penalties previously introduced from being removed (set to 0) for a certain number of re-optimizations and also forbidding penalties removed from being re-introduced for a certain span of time, as established by customary rules for assigning tabu tenure. The rules for determining which penalized variables should be freed from their penalties can make use of LP reduced costs in the solution to LP(c,d). These reduced costs provide an evaluation that discloses the amount by which the associated variables resist their penalties.

Let RC_j^+ and RC_j^- denote the reduced costs for x_j^+ and x_j^- in the current LP solution. At optimality all reduced costs are non-negative and those for basic variables are 0. The true reduced costs TRC_j^+ and TRC_j^- for x_j^+ and x_j^-, representing the reduced costs there variables would have in the current basis if no penalties were assigned to the variables (before the added step of driving the reduced costs for basic variables again to zero) are given by $TRC_j^+ = RC_j^+ - c_j$ and $TRC_j^- = RC_j^- - d_j$. The smaller (or "more negative") this value is, the more attractive the variable is to be released from its penalty. The measure can be supplemented by taking account of the adjusted reduced cost values that result by

zeroing out the TRC_j^+ and TRC_j^- values for basic variables in the current objective function representation. Basic variables can additionally be evaluated by reference to their current values in the LP solution, where those with larger values are more attractive to be released.

By means of this evaluation, instead of automatically releasing a tabu variable that receives a penalty from its penalized status when its current tabu tenure expires, the method can consider all such tabu variables whose residual tenures fall below a specified value and release one or more of these that are evaluated as most attractive according to the reduced cost measure. Frequency-based memory can supplement this recency-based memory in the customary manner, as by modifying choice rules to discourage or encourage the assignment of penalties to particular variables according to how often (or for what cumulative duration) they have been penalized previously, or to increase or decrease the values of penalties assigned. Frequencies likewise can be used to amend the choice of variables to be released from tabu status by a rule that is independent of the reduced cost values, as by creating aspiration criteria for overriding tabu status that are a function of such frequencies.

Additional more advanced considerations, including associated intensification and diversification strategies, ways to exploit cutting planes within the parametric design, and detailed specifications of the mechanisms for handling tabu status in relation to penalties, can be found in the parametric tabu search method described in Glover (2006a).

Appendix 2: Creating General Subspaces for Multi-hyperplane MIP formulations.

As in Section 5.2, we use the symbol $\cup$ to represent set union (the *logical or* operator) and the symbol $\cap$ to identify set intersection (the *logical and* operator). Similarly we let the symbol * represent complementation (the *logical not* operator).

To create the subspaces used for classification, we generate logical strings such as $(5^*\cap 4)\cup((6\cap 1^*)\cup 2)$ to identify regions designed to include the points of one group but not the other.[9] Our approach manipulates these strings by Boolean analysis to produce associated mixed integer programming models, and we may accordingly call the resulting subspaces *Boolean subspaces*. A Boolean subspace will be called *variable* if its form is represented as in with b^p and x^p variable, and will be called *fixed* if we have assigned specific constant values to b^p and the components of the vector x^p.

Our multi-hyperplane discrimination approach takes the following form:

1. Generate a series of logic strings to identify a collection of variable Boolean subspaces denoted by B_h, $h \in H$.
2. Create a mixed integer programming formulation based on B_h and its variable complement B_h^* to determine fixed instances B_h and B_h^* of these subspaces, for the objective of including as many points as possible of G_1 and G_2 in B_h and B_h^*, respectively.

For the purpose of comparing logic strings, and of creating a mixed integer programming model from them, we convert them into disjunctive normal form; i.e., we express the strings as the disjunction (union) of a collection of conjunctions (intersections), as represented by

$$V_1 \cup V_2 \cup V_3 \cup \ldots \cup V_r$$

where each V_k, $k = 1, \ldots, r$ has the form $V_k = (V_{k1} \cap V_{k2} \cap V_{k3} \ldots)$ and the terms V_{kj} are primitives, in the present case consisting of terms p and p*, for various values of $p \in P = \{1,\ldots, p_o\}$. The value of r is variable and depends on a given logical string. We suppose the primitives of each V_k have been logically reduced so that, within any given V_k, no primitive p or p* appears twice and at most one of the pair p and p* appears. (Multiple occurrences of a given primitive can be replaced by a single instance, and the appearance of both p and p*, which by convention we assume to have an empty intersection, causes V_k to drop out of the disjunctive normal form.) Likewise, we assume that the sets V_k have been reduced so that any set that is a proper subset of another is dropped, and all but one occurrence of multiple identical sets are dropped. (Note, under

[9] For the purpose of identifying implicit parentheses in subsequent strings, the $\cap$ operator takes a stronger position than the $\cup$ operator, analogous to the way the multiplication operator is used relative to the addition operator in ordinary arithmetic representations.

intersection, a set V_k is a subset of a set V_h if all of the primitives of V_h appear in V_k, hence V_k contains more rather than fewer primitives in its representation.)

In addition, if we wish to enumerate the possible disjunctive normal forms of interest, we may require the primitives of each V_k to be arranged in ascending order by size, where by convention we extend the definition of "<" so that $p < p^*$ and $p^* < q$ if $p < q$. With the primitives of the sets V_k thus ordered, we also use the symbol "<" to denote a lexicographical relationship between these sets, and define a set V_k to be *lexicographically smaller* than a set V_h, writing $V_k < V_h$, if V_k contains fewer primitives than V_h or if the two sets contain the same number of primitives, and the first primitive in which they differ is smaller in V_k than in V_h. (Thus, for example, the set $(3 \cap 5 \cap 7^*)$ is lexicographically smaller than both $(2 \cap 3 \cap 6^* \cap 7)$ and $(3 \cap 5^* \cap 7)$.)

Finally, we stipulate that the disjunctive normal form representation is ordered by lexicography so that $V_1 < V_2 < V_3 < .... < V_r$. We call this the *canonical representation* of the disjunctive normal form. It is easy to see that the canonical representation is unique, given our assumptions on reducing the sets V_k and dropping those that are superfluous. We also compare two canonical forms having the same number of component sets on the basis of their lexicographic size, saying that a canonical form identified by $V_1' < V_2' < V_3' < \ldots < V_r'$ is lexicographically smaller than one identified by $V_1 < V_2 < V_3 < .... < V_r$. if $V_h' < V_h$, for the first index h such that $V_h' \neq V_h$.

The relevance of comparing canonical forms in this way is that we consider two different canonical forms to be equivalent, for the purpose of generating subspaces, if we can rename the primitives in one of them so that their representations become identical. (We interpret renaming to include the possibility of mapping a primitive p^* into a primitive q, and vice versa.) Thus, to avoid the need to consider many different possible representations that are equivalent in this sense, we seek to restrict attention to canonical forms that are *minimal* by which we mean they cannot be made lexicographically smaller by renaming. Then we only generate subspaces from these minimal canonical forms.

We identify a heuristic renaming rule designed to achieve such a minimal representation, given that the representation is already canonical, and then will illustrate the rule to provide an understanding of these minimal forms.

Renaming Process

Stage 1.

1. Beginning with a canonical representation, rename the elements of set V_1 so that they are the primitives, 1, 2, 3, …,p in sequence. (This can change the names of elements in other sets, and also implicitly determines the names of primitives that are complements q or q^* of any of the primitives renamed.) If $r = 1$ or $p = p_o$ proceed to Stage 2. Otherwise, let $k = 2$ to examine set V_k for $k = 2$.

2. Restricting attention to primitive in V_k that have not yet been explicitly or implicitly renamed, assign these primitives the names p+1, p+2, …, p′. Let q or q^* denote the largest of these renamed primitives in the set. ($q = p'$ if any primitive has changed its name, and otherwise $q^* = p^*$.)

3. If k = r or q = p_o proceed to Stage 2. Otherwise, redefine p: = q, set k: = k+1, and return to Step 2 of Stage 1.

Stage 2.

1. If no change occurred in applying Stage 1, proceed to Stage 3.

2. Otherwise, put the primitives back in ascending order in each of the sets, and arrange the sets again in lexicographically increasing order to achieve a new canonical representation. If this causes no changes, likewise proceed to Stage 3.

3. Otherwise, return to repeat the process of Stage 1.

Stage 3.

1. If there is an element p < q where p or p* and q or q* lie in a given set V_k and where swapping the names p $\leftrightarrow$ q will decrease the lexicographic size of the resulting canonical form, then rename p and q as indicated. Repeat this process until no more name exchanges of this type remain.

The approach can be accelerated by restarting Stages 1 and 2 from the point where changes have occurred since they were last visited. We have expressed Stage 3 without care to make it efficient. Our goal is simply to demonstrate a process for seeking minimal canonical forms to clarify their nature. Subsequently we will employ a constructive process that creates such forms automatically without the need for a heuristic or exact procedure to revise a canonical form that lacks this property.

For the purpose of illustration, we start with an initial disjunctive normal form representation given by

$$(3\cap 2^*\cap 1\cap 2^*) \cup (1\cap 4^*\cap 4) \cup (4\cap 5) \cup (2\cap 5\cap 3\cap 5) \cup (5\cap 4\cap 2).$$

To put this in canonical form, we first arrange the primitives of each set in ascending order, to give

$$(1\cap 2^*\cap 2^*\cap 3) \cup (1\cap 4\cap 4^*) \cup (4\cap 5) \cup (2\cap 3\cap 5\cap 5) \cup (2\cap 4\cap 5).$$

Now, we remove an extraneous 2* from the first set and an extraneous 5 from the fourth set, and drop the second set because it contains both 4 and 4*. Similarly we drop the fifth set because it is a subset of the third ($(2\cap 4\cap 5) \subset (4\cap 5)$). Upon putting the remaining sets in lexicographically increasing order we have the canonical representation

$$(4\cap 5) \cup (1\cap 2^*\cap 3) \cup (2\cap 3\cap 5).$$

Next, we apply Stage 1 of the renaming process yields new names by the mapping $4 \rightarrow 1$, $5 \rightarrow 2$, $1 \rightarrow 3$, $2^* \rightarrow 4$, $2 \rightarrow 4^*$, $3 \rightarrow 5$, renaming all primitives before reaching the last set, yields

$$(1\cap 2) \cup (3\cap 4\cap 5) \cup (4^*\cap 5\cap 2)$$

or on re-ordering primitives in the last set, which causes the second and third sets to switch positions, we obtain

$$(1\cap 2) \cup (2\cap 4^*\cap 5) \cup (3\cap 4\cap 5)$$

Now we again apply Stage 1, yielding $4^* \rightarrow 3$ and $4 \rightarrow 3^*$, $5 \rightarrow 4$, $3 \rightarrow 5$ to give

$$(1\cap 2) \cup (2\cap 3\cap 4) \cup (5\cap 3^*\cap 4)$$

or on rearranging

$$(1\cap 2) \cup (2\cap 3\cap 4) \cup (3^*\cap 4\cap 5)$$

Upon once again applying Stage 1 nothing changes and we proceed to Stage 3. We see that we can swap the names 1 and 2 in the first set to decrease the lexicographic size of the second, thus yielding

$(1\cap 2) \cup (1\cap 3\cap 4) \cup (3^*\cap 4\cap 5)$.

No similar swaps remain and we are done. It is easy to show that this approach is not sufficient to guarantee a minimal form will be found. For example, if we start with

$(1\cap 2) \cup (4\cap 5) \cup (3\cap 4\cap 6)$

we first obtain (after re-ordering the last set)

$(1\cap 2) \cup (3\cap 4) \cup (3\cap 5\cap 6)$

This is a locally optimal solution. However, if we rename the elements of the second set by $3 \rightarrow 1$ and $4 \rightarrow 2$, followed by renaming those of the first set by $1 \rightarrow 3$ and $2 \rightarrow 4$, then we obtain

$(1\cap 2) \cup (3\cap 4) \cup (1\cap 5\cap 6)$

Moreover, we obtain this same result by renaming $1 \rightarrow 4$ and $2 \rightarrow 3$ in the first set.

For any such representation for a primary set, we can generate the complementary representation for the secondary set, and then exploit the outcome by the process described in Section 5.2.

Elementary set theory relationships useful for generating canonical representations and their complements:

$A\cup(B\cap C) = (A\cup B)\cap(B\cup C)$
$A\cap (B\cup C) = (A\cap B)\cap(A\cup C)$
$(A\cap B)^* = A^*\cup B^*$
$(A\cup B)^* = A^*\cap B^*$
$(A^*)^* = A$