

To appear in *Expert Systems with Applications* (2012).

Integrated Classifier Hyperplane Placement and Feature Selection

John W. Chinneck
Systems and Computer Engineering
Carleton University
Ottawa, Ontario K1S 5B6
Canada
chinneck@sce.carleton.ca
May 18, 2011

Abstract

The process of placing a separating hyperplane for data classification is normally disconnected from the process of selecting the features to use. An approach for feature selection that is conceptually simple but computationally explosive is to simply apply the hyperplane placement process to all possible subsets of features, selecting the smallest set of features that provides reasonable classification accuracy. Two ways to speed this process are (i) use a faster filtering criterion instead of a complete hyperplane placement, and (ii) use a greedy forward or backwards sequential selection method. This paper introduces a new filtering criterion that is very fast: maximizing the drop in the sum of infeasibilities in a linear-programming transformation of the problem. It also shows how the linear programming transformation can be applied to reduce the number of features after a separating hyperplane has already been placed while maintaining the separation that was originally induced by the hyperplane. Finally, a new and highly effective integrated method that simultaneously selects features while placing the separating hyperplane is introduced.

1. Introduction

Classifier decision trees are constructed by a sequential process of placing separating surfaces, frequently hyperplanes. It is often advantageous to use as few features as possible when placing each separating surface, generally because there are costs associated with collecting the data for each feature (e.g. the cost of a medical test), but also because using fewer features sometimes results in better classification accuracy.

There is a sizable literature on feature selection for classification. Excellent summaries are provided by Dash and Liu [1997] and Liu and Yu [2005]. Following Liu and Yu [2005], there are three main categories of feature selection methods: (i) *filter methods* which use metrics based on the data set to select features, (ii) *wrapper methods* which select subsets of features and evaluate them by applying the data mining technique (e.g. separating hyperplane) while using only the selected subset of features, and (iii) *hybrid methods* that combine elements of both filter and wrapper methods.

A subcategory of all methods is the search strategy. *Complete search* evaluates all possible combinations of features; this is combinatorially explosive and impractical for more than a few features, though it does return the optimum solution (relative to the evaluation metric). For data sets having many features, alternative approaches are needed. *Random search* begins with a random subset of features and iterates towards an improved set, e.g. via simulated annealing. *Sequential search* proceeds through the set of features based on an ordering heuristic. The two most common sequential approaches are:

- *Sequential forward selection*. This begins with no features and selects the single feature that gives the best value of the evaluation metric. It then iteratively adds the next feature that provides the next best value of the evaluation metric. One-by-one addition of features continues until a stopping criterion is met, e.g. there is no improvement in the evaluation criterion by adding the next feature.
- *Sequential backward elimination*. This is the opposite of sequential forward selection: it begins with all features included and eliminates the feature whose removal gives the best value of the evaluation metric. It then iteratively eliminates the next feature whose removal gives the next best value of the evaluation criterion. One-by-one elimination of features continues until a stopping criterion is met, e.g. there is no improvement in the evaluation criterion by adding the next feature.

This paper develops a variety of sequential forward and backwards search methods using filtering based on a new feature selection evaluation metric. It also develops a new integrated method that uses this evaluation metric while alternating between feature addition and separating hyperplane placement. Finally it develops new methods for removing features in such a way that a given linear separation is maintained. In other words, an arbitrary technique can first be applied to find a desirable separating hyperplane, and then features can be removed while maintaining the same separation.

There has been relatively little work on integrated methods for simultaneous separating hyperplane placement and feature selection. Bredensteiner and Bennett [1997] solve a parametric bilinear program to do so while maintaining a specified level of total accuracy. The problem is solved by a variation of a Franke-Wolfe algorithm.

Bradley and Mangasarian [1998] introduce a parameter into the objective function to permit the relative weighting of two objectives: the original objective function that seeks to find a high accuracy separation and a second objective function that minimizes the number of features. The resulting optimization problem is nonlinear but convex and is solved by a successive linear approximation algorithm. Bradley, Mangasarian, and Street [1998] examine similar approaches. Guo and Dyer [2003] report good results when these techniques are applied in facial expression recognition.

Dunbar et al [2010] formulate the simultaneous hyperplane placement and feature selection problem as a nonlinear support vector machine problem, and then reformulate it as a quadratic minimization problem subject to nonnegativity constraints. This is an extension of a method originally proposed by Bradley and Mangasarian [1998]. Maldonado et al [2011] present

another support vector machine based method that results in a nonlinear problem that must be solved.

1.1 Hyperplane Placement and the Maximum Feasible Subset Problem

The problem of placing a separating hyperplane to minimize the number of misclassified binary data points is equivalent to the following problem: given an infeasible set of linear inequalities, find the maximum cardinality subset that constitutes a feasible set [Amaldi 1994, Parker 1995, Chinneck 2001, Chinneck 2009]. This second problem is known by a number of names: the Maximum Satisfiability Problem, the Maximum Feasible Subset Problem (MaxFS), the Minimum Unsatisfied Linear Relation Problem, or the Minimum Cardinality IIS Set Covering Problem; we will use MaxFS hereafter.

The conversion of the data misclassification minimization problem to the MaxFS problem is straightforward [Chinneck 2001]. Given a training set of I binary data points ($i=1\dots I$) in J dimensions ($j=1\dots J$), in which the value of attribute j for point i is denoted by d_{ij} , where the class of each point is known (either Type 0 or Type 1), construct one linear inequality constraint for each data point:

- for each point of Type 0: $\sum_j d_{ij} w_j \leq w_0 - \epsilon$
- for each point of Type 1: $\sum_j d_{ij} w_j \geq w_0 + \epsilon$

where ϵ is a small positive constant (often set at 1). The variables are the unrestricted w_j where $j=0\dots J$, while the d_{ij} are known constants.

This set of linear inequalities has a feasible solution (which is easily found by linear programming) if and only if the set of data points can be completely separated by a single hyperplane. Where the data cannot be completely separated by a single hyperplane, the set of linear inequalities is infeasible. In this case, a solution to the MaxFS problem identifies the maximum cardinality feasible subset of constraints, and at the same time identifies the smallest subset of excluded constraints. Any feasible point satisfying the largest feasible subset of inequalities provides values for the w variables, thereby identifying the parameters of the separating hyperplane $\sum_j d_{ij} w_j = w_0$. Such a feasible point is normally found by linear programming (LP). The constraints excluded by the MaxFS solution correspond to data points that are misclassified by the resulting hyperplane. Thus a solution for the MaxFS problem provides a hyperplane that misclassifies the minimum number of data points.

Unfortunately, the MaxFS problem is NP-hard [Sankaran 1993; Chakravarti 1994; Amaldi and Kann 1995]. A number of solution approaches have been developed for this problem; see Chinneck [2008, chapter 7] for a survey. Small problems can be formulated for exact solution, either as mixed-integer linear programs, or as a linear programs with equilibrium constraints (LPEC). Heuristic solutions for the LPEC formulation have been developed in the machine learning community [Mangasarian 1994, Bennett and Bredensteiner 1997]. Parker [1995] and Parker and Ryan [1996] described a method that gradually enumerates infeasible subsets of constraints from which at least one must be removed to create a feasible subset. Chinneck

[1996, 2001] developed a number of greedy heuristics that reduce a measure of the infeasibility of the current subset of constraints at each iteration.

Any of the algorithms described above can be applied to solve the MaxFS problem for the purpose of identifying a separating hyperplane that minimizes the number of misclassifications. The methods developed in this paper are based on the infeasibility-reducing algorithms by Chinneck [1996, 2001, 2009] for several reasons. First, these methods provide the best results over a variety of categories of MaxFS problems [Jokar and Pfetsch 2008]. Second, they are easily extended to pursue goals other than maximum overall accuracy [Chinneck 2009]. Third, and most crucially, the sequential nature of the methods allows us to make the trade-off between the accuracy of the hyperplane placement and the selection of features in an integrated manner, which is the subject of this paper.

Chinneck's MaxFS solution algorithms first require that the linear inequalities be converted to elastic form [Brown and Graves 1975] by the addition of nonnegative elastic variables, e_i , one per inequality, as follows:

- Type 0 inequalities take the elastic form $\sum_j d_{ij} w_j - e_i \leq w_0 - \epsilon$
- Type 1 inequalities take the elastic form $\sum_j d_{ij} w_j + e_i \geq w_0 + \epsilon$

An *elastic program* is constructed, consisting of the elastic inequalities, nonnegativity bounds on the elastic variables, and an elastic objective function of the form *minimize* $SINF = \sum_{i=1}^I e_i$. Because of the minimization, an elastic variable will take on a positive value only when the original non-elastic version of the constraint is violated, hence *SINF* indicates the "sum of the infeasibilities". The main MaxFS heuristic is then applied to the elastic program, based on the following concepts:

- An LP solution of the elastic program minimizes *SINF*, which is a measure of the total infeasibility in the current set of constraints. *SINF* can be reduced by removing constraints that contribute to the infeasibility. When *SINF* reaches zero, the remaining constraints constitute a feasible subset.
- Constraints are removed one at a time, and a new lower value of *SINF* is found by re-solving the LP for the smaller set of constraints.
- A candidate list of constraints for removal at each iteration can be generated in various ways. The list can be as short as one member if a quick solution is desired.
- When there are several candidates for removal, the one that lowers *SINF* the most is chosen. This greedy heuristic is highly effective in practice.

A simplified statement of the most basic version of the *SINF*-reducing algorithm is shown in Algorithm 1 (some steps that improve efficiency are omitted for clarity of the main procedure). Numerous LPs are solved, but the method is highly efficient because each new LP is very similar to the previous one solved. This means that each new LP can be solved in just a few simplex iterations due to the advanced start routines in modern LP solvers.

There are several ways to construct the list of candidate constraints in Step 4 of Alg. 1. The method that yields the best results includes as candidates all constraints to which the elastic objective function is sensitive in the current LP solution, i.e. for which the reduced cost associated with e_i is nonzero: in this case constraint i will be either violated or tight. This list of candidates may be lengthy, so there are other ways to construct shorter lists, but possibly at

INPUT: an infeasible set of linear constraints.

1. Elasticize the constraints by adding appropriate elastic variables.
2. Solve the elastic LP.
3. If $SINF = 0$ then exit.
4. Construct the list of candidate constraints for removal.
5. For each candidate constraint:
 - 5.1. Temporarily remove the candidate constraint.
 - 5.2. Solve the reduced elastic LP and note the new value of $SINF$.
 - 5.3. Reinstate the candidate constraint.
6. Permanently remove the candidate constraint whose temporary removal gave the smallest value of $SINF$.
7. Go to Step 2.

OUTPUT: large cardinality feasible subset of constraints.

Algorithm 1: Finding a large cardinality feasible subset of constraints [Chinneck 2009].

the cost of finding a smaller feasible subset, and hence a less accurate separating hyperplane.

Where $\sigma(e_i)$ represents the reduced cost associated with the variable e_i , the product $e_i|\sigma(e_i)|$ is a good heuristic estimator of the relative magnitude of the reduction in $SINF$ experienced when constraint i is removed during Step 5.1 of Alg. 1 (provided that e_i is positive, i.e. the original non-elastic version of constraint i is violated) [Chinneck 2001]. When $e_i=0$, the size of $|\sigma(e_i)|$ alone is a good estimator of the relative magnitude of the reduction in $SINF$ when constraint i is removed during Step 5.1 of Alg. 1 [Chinneck 2001]. These two observations provide a way to construct shorter lists of candidate constraints in Step 4 of Alg. 1. Simply take the top k largest elements of the two lists: the constraints corresponding to the k largest values of $e_i|\sigma(e_i)|$ for constraints having $e_i>0$, and the constraints corresponding to the k largest values of $|\sigma(e_i)|$ for constraints having $e_i = 0$.

A good heuristic for selecting just a single candidate in Step 4 of Alg.1 (thereby reducing Step 5 to a single test), is to choose the constraint having the largest value of $e_i|\sigma(e_i)|$ from among constraints in which $e_i > 0$ [Chinneck 2001]. When $SINF > 0$ there is always at least one $e_i > 0$.

In the sequel, the *Orig* algorithm will be taken to mean the original version of the algorithm which includes as candidates all constraints to which the elastic objective function is sensitive in the current LP solution. A fast version of the original version of the algorithm in which exactly one candidate is used in Step 4 of Alg. 1 is also tested. The candidate chosen is the constraint having the largest value of $e_i|\sigma(e_i)|$ from among constraints having $e_i > 0$.

The reduced set of data points corresponding to the satisfied data point inequalities is completely linearly separated by the final hyperplane returned by the algorithm. However it may be advantageous to adjust the placement of the final hyperplane to obtain better generalization. There are variety of ways to do this, including minimizing the distance from the hyperplane to the misclassified points, maximizing the distance from the hyperplane to the correctly classified points, averaging these two approaches, etc. [Chinneck 2009]. It is also possible to maximize the margins by applying a support vector machine to the subset of correctly classified points [Cristianini and Shawe-Taylor 2000].

Note that while these methods are specifically for linear separating hyperplanes, they are easily extendable to nonlinear separating surfaces by including appropriate nonlinear data. For example, squaring the value of some feature x and including it as a new feature allows x^2 to included in the linear combination of terms that is returned by the separating hyperplane.

2. Integrating Feature Selection and Hyperplane Placement

The linear program used in Alg. 1 for hyperplane placement includes an inequality for each of the data points in the binary dataset, but the elements of \mathbf{w} , the variables whose solution values provide the coefficients of the features in the separating hyperplane, are completely unrestricted in value. However it is simple to introduce constraints that prevent certain features from being used in the resulting hyperplane. For example, to prevent feature j from being used, add the constraint $w_j=0$. When constraints of this form are added to the set of inequalities for the data points, the complete set of constraints integrates both hyperplane placement and feature selection. We have the option of sequential forward addition of features (by removing constraints of the form $w_j=0$), or of sequential backwards elimination of features (by adding constraints of the form $w_j=0$). We can also alternate between removing constraints corresponding to data points, and adding or removing constraints corresponding to features. Some variants of these options that have proved to be particularly useful are described below.

The algorithms that follow make use of 3 main elements:

- constraints derived from the data points,
- constraints to allow/remove features, and
- the value of the elastic objective function, $SINF$, defined over only the constraints derived from the data points.

Using the elastic objective function defined over only the data points reflects the goal of maximizing the overall accuracy subject to the features currently used in the separating hyperplane. Elastic variables are therefore not added to the feature constraints of the form $w_j=0$; constraints of this type are either removed or added in their entirety (this is simple to do in most LP solvers by specifying that the constraint is either of type "equality", which includes the constraint, or of type "free" which means that the constraint is not binding). As in Alg. 1, the best choice among options is normally indicated by the largest drop in the value of $SINF$; this is the new metric for the filtering methods of feature selection developed here. $SINF$ can be used

in both of the usual filtering modes: sequential forward selection and sequential backwards elimination. It can also be used in a new integrated manner described later.

These basic building blocks are very flexible. We examine below a variety of ways to use them to allow feature selection prior to placing a hyperplane, feature reduction after a separation has been found (is there a smaller set of features that gives an equivalent separation?), and integrated hyperplane placement and feature selection. In the same vein, while simultaneously considering feature selection, these basic ingredients can also be used to pursue other hyperplane placement goals besides maximizing total accuracy, such as balancing the population accuracies, balancing the accuracies on each side of the hyperplane, etc. [Chinneck 2009].

2.1 Feature Selection Prior to Hyperplane Placement

The *AddB* algorithm is a sequential forward feature selection algorithm that operates prior to placing the separating hyperplane, as shown in Alg. 2. *AddB* ("add features before hyperplane placement") begins with no features included at all (i.e. constraints $w_j=0, j=1\dots J$ are in place). Features are then added one by one in a greedy manner: in each round, the feature that most reduces *SINF* is added, provided that it reduces *SINF* below the value reached when the last feature was permanently added in the previous round. Feature selection terminates when adding the next feature does not reduce *SINF* any further. After a set of features is selected, the separating hyperplane is then found, beginning in Step 4.

INPUT: a set of inequality constraints representing the data points $i=1\dots I$, and a set of equality constraints representing the feature constraints $j=1\dots J$.

1. Elasticize the data point constraints by adding appropriate elastic variables.
 2. Solve the elastic LP. $SINF_{LastBest} \leftarrow SINF$.
 3. Do J times:
 - 3.1. $SINF_{Min} \leftarrow \infty$, $Feature \leftarrow \emptyset$.
 - 3.2. For each feature $j=1$ to J :
 - 3.2.1. If feature j is still excluded (i.e. constraint $w_j=0$ is still included) then:
 - 3.2.1.1. Temporarily add the feature by removing the feature constraint.
 - 3.2.1.2. Solve the elastic LP and note the value of $SINF$.
 - 3.2.1.3. If $SINF < SINF_{Min}$ then $SINF_{Min} \leftarrow SINF$ and $Feature \leftarrow j$.
 - 3.2.1.4. Remove feature j by re-introducing the constraint $w_j=0$.
 - 3.3. If $SINF_{Min} < SINF_{LastBest}$ then:
 - 3.3.1. Select feature $Feature$ by permanently removing the constraint $w_{Feature}=0$.
 - 3.3.2. $SINF_{LastBest} \leftarrow SINF_{Min}$.
 - 3.3.3. If $SINF_{LastBest} = 0$ then go to Step 4.
 - 3.4. Else go to Step 4.
 4. Solve the elastic LP.
 5. If $SINF = 0$ then exit.
 6. Construct the list of candidate data point constraints for removal.
 7. For each candidate constraint:
 - 7.1. Temporarily remove the candidate constraint.
 - 7.2. Solve the reduced elastic LP and note the new value of $SINF$.
 - 7.3. Reinstate the candidate constraint.
 8. Permanently remove the candidate constraint whose temporary removal gave the smallest value of $SINF$.
 9. Go to Step 4.
- OUTPUT: a set of features (those for which $w_j \neq 0$) and a separating hyperplane equation (given by $\sum_j w_j x_j = w_0$).

Algorithm 2: Adding features before hyperplane placement (**AddB**).

Of course, it is possible to proceed in the opposite way: begin with all features in place and iteratively remove them, i.e. use a sequential backwards elimination procedure. This generally increases $SINF$, so an exit condition is needed to prevent it from becoming too large. The *DelB1* algorithm ("delete features before, variant 1"), shown in Alg. 3, allows a small percent increase in $SINF$ relative to the last accepted value, denoted by $SINF_{LastBest}$. The parameter P represents the allowable percent increase in $SINF$ relative to $SINF_{LastBest}$. There is a single run through the features, testing and potentially removing each feature one by one.

The *DelB2* variant allows a small percent increase relative to the original $SINF$ with all features in place, hence it is identical to Alg. 3 except that Step 3.4 is omitted.

INPUT: a set of inequality constraints representing the data points $i=1\dots l$, and a parameter P .

1. Elasticize the data point constraints by adding appropriate elastic variables.
2. Solve the elastic LP. $SINF_{LastBest} \leftarrow SINF$.
3. For each feature $j=1$ to J :
 - 3.1. Delete feature j by adding the feature constraint $w_j=0$.
 - 3.2. Solve the elastic LP.
 - 3.3. If $SINF > SINF_{LastBest} \times (1+P/100)$ then reinstate feature j by removing the feature constraint $w_j=0$.
 - 3.4. Else $SINF_{LastBest} \leftarrow SINF$.
4. Solve the elastic LP.
5. If $SINF = 0$ then exit.
6. Construct the list of candidate data point constraints for removal.
7. For each candidate constraint:
 - 7.1. Temporarily remove the candidate constraint.
 - 7.2. Solve the reduced elastic LP and note the new value of $SINF$.
 - 7.3. Reinststate the candidate constraint.
8. Permanently remove the candidate constraint whose temporary removal gave the smallest value of $SINF$.
9. Go to Step 4.

OUTPUT: a set of features (those for which $w_j \neq 0$) and a separating hyperplane equation (given by $\sum_j w_j x_j = w_0$).

Algorithm 3: Deleting features before hyperplane placement, allowing a small increase in $SINF$ relative to last accepted value (*DelB1*).

2.2 Feature Selection After Hyperplane Placement

The question here is this: given a separating hyperplane found by an arbitrary method, is there a hyperplane that has the same sets of correctly and incorrectly classified points but which uses fewer features? The building blocks allow this question to be addressed in a number of ways.

Given a separating hyperplane, the points in the training set can be grouped into the correctly and incorrectly classified sets. Incorrectly classified points are removed, leaving only the correctly classified points. The *AddA* algorithm (*add* features *after* hyperplane placement) operates on the constraints derived from the correctly classified points in a sequential forward selection manner as shown in Alg. 4. All of the feature constraints are included at the outset (i.e. all features are excluded), so $SINF$ will initially be greater than zero. At least one feature must be added, and Steps 3 and 4 take care of identifying and adding the feature that most reduces $SINF$. Step 5 adds further features in a series of iterations. Each iteration adds the feature that most decreases $SINF$. The iterations cease when $SINF$ reaches zero .

INPUT: a set of correctly classified data points.

1. Construct and elasticize the data point constraints for the correctly classified points. Construct and add the feature constraints for $j=1\dots J$.
2. Solve the elastic LP. $SINF_{LastBest} \leftarrow SINF$. $SINF_{Min} \leftarrow \infty$.
3. For each feature $j=1$ to J :
 - 3.1. Temporarily include feature j by deleting the feature constraint $w_j=0$.
 - 3.2. Solve the elastic LP.
 - 3.3. If $SINF < SINF_{Min}$ then $SINF_{Min} \leftarrow SINF$ and $Feature \leftarrow j$.
 - 3.4. Remove feature j by adding the feature constraint $w_j=0$.
4. If $SINF_{Min} < SINF_{LastBest}$ then:
 - 4.1. $SINF_{LastBest} \leftarrow SINF_{Min}$.
 - 4.2. Permanently add feature $Feature$ by removing constraint $w_{Feature} = 0$.
5. For each feature $j=1$ to J :
 - 5.1. $SINF_{Min} \leftarrow \infty$.
 - 5.2. For each feature $k=1$ to J :
 - 5.2.1. If feature k is not included (i.e. constraint $w_k=0$ is included) and the elastic objective function is sensitive to it then:
 - 5.2.1.1. Temporarily include feature k by deleting the feature constraint $w_k=0$.
 - 5.2.1.2. Solve the elastic LP.
 - 5.2.1.3. If $SINF < SINF_{Min}$ then:
 - 5.2.1.3.1. $SINF_{Min} \leftarrow SINF$. $Feature \leftarrow k$.
 - 5.2.1.3.2. If $SINF=0$ then go to Step 5.3.
 - 5.2.1.4. Remove feature j by adding the feature constraint $w_j=0$.
 - 5.3. If $SINF_{Min} < SINF_{LastBest}$ then:
 - 5.3.1. $SINF_{LastBest} \leftarrow SINF_{Min}$.
 - 5.3.2. Permanently add feature $Feature$ by removing constraint $w_{Feature} = 0$.
 - 5.3.3. Solve the elastic LP.
 - 5.3.4. If $SINF_{LastBest} = 0$ then exit.
 - 5.4. Else exit.
6. Exit.
7. OUTPUT: a set of features (those for which $w_j \neq 0$) and a separating hyperplane equation (given by $\sum_j w_j x_j = w_0$).

Algorithm 4: Selecting features by adding them after a separation is available (**AddA**).

The *DeIA* algorithm takes the opposite tack of sequential backward elimination by initially including all of the features and gradually deleting them, as shown in Alg. 5. Note that only the constraints for points that were incorrectly classified by the previously found separating hyperplane are elasticized, and hence the elastic variables for only these constraints appear in the elastic objective function. All features are initially included. In each of up to J rounds, Step 2 identifies and removes the feature whose removal causes the smallest increase in $SINF$. Note that if too many features are removed then the LP may be infeasible; this is handled in Step

2.2.1.3. The algorithm terminates when the removal of each remaining feature causes infeasibility in the LP, hence no feature can be identified for removal.

INPUT: a set of correctly classified points, and a set of incorrectly classified points (possibly empty).

1. Construct and elasticize the data point constraints for the incorrectly classified points. Construct and add the nonelastic data point constraints for the correctly classified points.
2. For each feature $j=1$ to J :
 - 2.1. $SINF_{Min} \leftarrow \infty$. $Feature \leftarrow 0$.
 - 2.2. For each feature $k=1$ to J :
 - 2.2.1. If feature k is still included (i.e. constraint $w_k=0$ is not included) then:
 - 2.2.1.1. Temporarily remove feature k by adding the feature constraint $w_k=0$.
 - 2.2.1.2. Solve the elastic LP.
 - 2.2.1.3. If elastic LP is infeasible then $SINF \leftarrow \infty$.
 - 2.2.1.4. If $SINF < SINF_{Min}$ then $SINF_{Min} \leftarrow SINF$ and $Feature \leftarrow k$.
 - 2.2.1.5. Reinstate feature k by removing the feature constraint $w_k=0$.
 - 2.3. If $Feature > 0$ then permanently remove feature $Feature$ by adding the feature constraint $w_{Feature}=0$.
3. Solve the elastic LP and exit.

OUTPUT: a set of features (those for which $w_j \neq 0$) and a separating hyperplane equation (given by $\sum_j w_j x_j = w_0$).

Algorithm 5: Selecting features by removing them after a linear separation is available (**DeIA**).

The rationale behind Alg. 5 is that the separating hyperplane should remain as close as possible to the misclassified points (as measured by $SINF$ over the elastic constraints for the misclassified points). Alg. 5 also works when the set of misclassified points is empty. However in this case, $SINF$ is always zero, so the method provides no guidance concerning the ordering of the features to test for potential removal: features are tested in their natural order.

2.3 Integrated Feature Selection and Hyperplane Placement

A major advantage of including the feature removal constraints along with the data point constraints is that both feature selection and hyperplane placement can be handled in an integrated manner in a single model. While feature selection can be done before or after hyperplane selection as in Sections 2.1 and 2.2, the two operations can also be alternated, as described in this section. Hyperplane placement and the selection of features proceed together.

The *Int* ("integrated") algorithm described in Alg. 6 is a straightforward adaptation of the MaxFS algorithm that elasticizes the feature constraints and simply includes them directly in Alg. 1 along with the data point constraints. However to initialize the process the algorithm must first identify at least one feature to include, as shown in Steps 3-5. Steps 3 and 4 first attempt to add the single feature that most reduces $SINF$ (by eliminating the associated feature

constraint). If that process fails, then Step 5 initiates the opposite process: it adds all features (eliminates all feature constraints) and then tries to remove as many as possible (by adding the feature constraints back in), choosing at each step the feature that least increases S/NF as long as S/NF does not increase beyond the original value. Once this is done, then a straightforward application of the MaxFS algorithm follows in Step 6 and beyond. Initial testing showed that Step 5 is needed in a third of the data sets (4 of 12), and does not impact the success of the method.

INPUT: a set of inequality constraints representing the data points $i=1\dots I$, and a set of equality constraints representing the feature constraints $j=1\dots J$.

1. Elasticize the data point constraints by adding appropriate elastic variables.
 2. Solve the elastic LP. $SINF_{LastBest} \leftarrow SINF$. $SINF_{Min} \leftarrow \infty$.
 3. For each feature $j=1\dots J$:
 - 3.1. Include feature j by removing the feature constraint $w_j=0$.
 - 3.2. Solve the elastic LP.
 - 3.3. If $SINF < SINF_{Min}$ then $SINF_{Min} \leftarrow SINF$ and $Feature \leftarrow j$.
 - 3.4. Remove feature j by adding the feature constraint $w_j=0$.
 4. If $SINF_{Min} < SINF_{LastBest}$ then:
 - 4.1. $SINF_{LastBest} \leftarrow SINF_{Min}$.
 - 4.2. Permanently add feature $Feature$ by removing constraint $w_{Feature}=0$.
 5. Else:
 - 5.1. $SINF_{Reference} \leftarrow SINF_{LastBest}$.
 - 5.2. Include all features by removing all feature constraints.
 - 5.3. Solve the elastic LP. $SINF_{LastBest} \leftarrow SINF$.
 - 5.4. For each feature $j=1\dots J$:
 - 5.4.1. $SINF_{Candidate} \leftarrow \infty$. $Feature \leftarrow 0$.
 - 5.4.2. For each feature $k=1\dots J$:
 - 5.4.2.1. If feature k is still included (i.e. constraint $w_k=0$ is not included):
 - 5.4.2.1.1. Remove feature k by removing constraint $w_k=0$.
 - 5.4.2.1.2. Solve elastic LP.
 - 5.4.2.1.3. If $SINF < SINF_{Candidate}$ then $SINF_{Candidate} \leftarrow SINF$ and $Feature \leftarrow k$.
 - 5.4.3. If $SINF_{Candidate} < SINF_{Reference}$ then permanently remove feature $Feature$ by adding constraint $w_{Feature}=0$.
 - 5.4.4. Else go to Step 6.
 6. Solve the elastic LP.
 7. If $SINF = 0$ then exit.
 8. Construct the list of candidate data point and feature constraints for removal.
 9. For each candidate constraint:
 - 9.1. Temporarily remove the candidate constraint.
 - 9.2. Solve the reduced elastic LP and note the new value of $SINF$.
 - 9.3. Reinstate the candidate constraint.
 10. Permanently remove the candidate constraint whose temporary removal gave the smallest value of $SINF$.
 11. Go to Step 4.
- OUTPUT: a set of features (those for which $w_j \neq 0$) and a separating hyperplane equation (given by $\sum_j w_j x_j = w_0$).

Algorithm 6: The integrated algorithm for simultaneous feature selection and hyperplane placement (*Int*).

3. Experiments

A series of experiments were run to test the algorithms described in Section 2. These were run on an Intel Core 2 Quad CPU Q9400 running at 2.67 GHZ with 8 Gbytes of RAM and a 64-bit Windows Vista operating system. All experiments were run in a single-threaded manner, though other operations on the machine resulted in minor variations in times for repeated experiments.

The software prototype is coded in Octave 3.2.3 [Octave 2011], an interpreted open-source Matlab clone. An essential element of the method is the LP solver, which is GLPK in Octave [GLPK 2011]. GLPK is a moderately capable LP solver according to a recent benchmark comparison [Mittelman 2011]. The software prototype code is not optimized for performance in any way. Many speed improvements are possible including storage of intermediate results to avoid re-solving LPs in numerous places, use of a better LP solver, and use of a compiled language. The parameter P used in the *De/B1* and *De/B2* algorithms is set at 5, a useful value in preliminary testing.

The reported statistics are based on the final separating hyperplane returned by the hyperplane placement algorithm. No attempt is made to adjust this final hyperplane prior to applying it to the testing set. It is possible that testing set results may be improved by such a concluding step, such as finding a support vector machine solution for the final separable set of data points, but preliminary experiments showed no particular advantage in doing this.

The algorithms developed in this paper are suitable only for data sets consisting solely of features having numerical values (real, binary, integer), and having a binary outcome. Twelve data sets meeting these criteria were chosen from the well-known UCI repository [Frank and Asuncion 2010]; basic statistics are summarized in Table 1. Note that incomplete instances were removed in all cases, so the number of instances shown may differ from the number shown in the UCI repository. Some multcategory data sets were converted to binary by choosing one of the categories as the positive outcome with all others deemed negative outcomes, as shown in the comments column in the table. The average number of features over the data sets is 30.00.

Some characteristics of the data sets are noteworthy:

- The number of instances in a data set affects the granularity of the results, especially in ten-fold cross-validation. The *iris* data set has just 150 points, which leaves just 15 points in a ten-fold test set. One additional misclassified point in this case amounts to a 6.7% reduction in overall classifier testing accuracy.
- Some data sets have many features and hence more potential for feature reduction. Six data sets have more than 10 features: *musk1*, *sonar*, *vehicle*, *wdbc*, *wine*, and *wdbc*.
- It is easy to find a 100% accurate separating hyperplane during training for these four data sets: *musk1*, *sonar*, *wdbc*, and *wine*.
- The fractions of the total instances in a data set that are of either type affects feature selection. For example, *wdbc* is 85.6% accurate with a null separating hyperplane

because positive instances constitute 14.4% of the population. Thus simply classifying *all* points as negative achieves an 85.6% accuracy.

Name	Data Set in Frank and Asuncion [2010]	Instances	Features	Comments
breast1	Breast Cancer Wisconsin (Original)	683	9	
bupa	Liver Disorders	345	6	
glass	Glass Identification	214	9	Type 2 vs. other types.
iris	Iris	150	4	Versicolour vs. other types.
newthyroid	Thyroid Disease, database from Stefan Aeberhard	215	5	Type 1 vs. other types.
pima	Pima Indians Diabetes	768	8	
sonar	Connectionist Bench (Sonar, Mines vs. Rocks)	208	60	
vehicle	Statlog (Vehicle Silhouettes)	850	18	Bus vs. other types.
wdbc	Breast Cancer Wisconsin (Diagnostic)	569	30	
wine	Wine	178	13	Type 3 vs. other types.
wdbc	Breast Cancer Wisconsin (Prognostic)	194	32	Outcome: recurrence within 24 months.
musk1	Musk (Version 1)	476	166	

TABLE 1: DATA SETS USED IN EXPERIMENTS.

The main results of the ten-fold cross-validation experiments are summarized in Table 2. Times are in seconds, accuracies are in percent. "Feature time" is the subset of the total training time that is devoted to selecting the features. There is no feature selection time for the *Orig* algorithm, and it is not meaningful to separate it out in the integrated *Int* algorithm. The highest test accuracy and the smallest number of features used for each model are shown in boldface.

data set	alg.	mean					variance				
		train time	feat time	train acc	test acc	train feat	train time	feat time	train acc	test acc	train feat
breast1	Orig	23.8		98.42	95.90	9.00	19.9		0.05	3.29	0.00
breast1	Int	34.2		98.00	96.34	4.50	21.8		0.09	4.43	0.50
breast1	AddB	26.4	2.7	98.42	95.90	9.00	19.5	0.0	0.05	3.29	0.00
breast1	DelB1	22.6	0.7	98.16	96.19	5.80	15.7	0.0	0.08	1.55	0.40
breast1	DelB2	23.1	2.0	98.21	95.60	6.30	18.0	0.0	0.06	5.77	0.23
breast1	AddA	26.0	2.6	98.42	95.60	8.40	18.8	0.0	0.05	3.39	0.71
breast1	DelA	24.1	0.4	98.42	95.75	8.10	19.5	0.0	0.05	4.54	0.54
bupa	Orig	136.1		75.91	69.29	6.00	30.2		0.40	17.88	0.00
bupa	Int	138.1		75.94	69.29	5.80	57.1		0.40	25.57	0.18
bupa	AddB	63.8	0.0	61.32	59.42	2.30	0.2	0.0	2.28	5.23	0.46

		mean					variance				
data set	alg.	train	feat	train	test	train	train	feat	train	test	train
		time	time	acc	acc	feat	time	time	acc	acc	feat
bupa	DelB1	107.7	0.1	68.96	67.82	2.20	18.6	0.0	5.95	59.16	0.18
bupa	DelB2	113.9	0.2	74.81	70.43	4.00	22.3	0.0	0.96	41.63	0.00
bupa	AddA	135.0	0.2	70.43	65.82	4.10	28.3	0.0	73.87	41.93	8.10
bupa	DeIA	135.3	0.1	75.91	69.29	5.80	30.1	0.0	0.40	17.88	0.18
glass	Orig	27.3		80.47	65.50	9.00	8.6		7.35	163.67	0.00
glass	Int	21.0		72.58	66.39	3.60	30.6		40.23	29.6	7.60
glass	AddB	17.3	0.1	66.36	62.64	1.20	15.3	0.0	6.28	11.9	2.62
glass	DelB1	18.3	0.1	67.97	65.87	1.90	9.3	0.0	40.32	10.7	6.77
glass	DelB2	23.2	0.2	79.70	71.08	6.60	8.6	0.0	8.01	41.7	0.27
glass	AddA	27.2	0.3	80.47	65.50	9.00	8.3	0.0	7.35	163.7	0.00
glass	DeIA	27.2	0.1	80.42	65.48	8.80	8.5	0.0	7.04	141.5	0.18
iris	Orig	4.3		83.26	80.00	4.00	0.0		0.39	39.5	0.00
iris	Int	4.2		76.52	70.67	2.10	0.1		3.66	120.5	0.54
iris	AddB	4.3	0.0	83.26	80.00	4.00	0.1	0.0	0.39	39.5	0.00
iris	DelB1	3.8	0.0	78.67	74.00	2.00	0.1	0.0	18.51	113.1	1.11
iris	DelB2	3.8	0.0	79.70	75.33	2.30	0.2	0.0	15.51	128.9	0.90
iris	AddA	4.3	0.0	83.26	80.00	4.00	0.1	0.0	0.39	39.5	0.00
iris	DeIA	4.3	0.0	83.26	80.00	4.00	0.0	0.0	0.39	39.5	0.00
musk1	Orig	3.2		100.00	67.63	166.00	0.0		0.00	34.2	0.00
musk1	Int	1051.0		94.02	81.28	24.70	7283.9		0.95	23.86	5.79
musk1	AddB	1015.8	1013.7	100.00	80.66	59.30	6296.2	6302.2	0.00	23.07	10.90
musk1	DelB1	102.4	99.7	100.00	77.30	90.40	7.0	6.9	0.00	25.7	8.71
musk1	DelB2	7746.2	7743.2	100.00	77.30	90.40	18078.2	18088.1	0.00	25.67	8.71
musk1	AddA	1024.9	1008.8	100.00	80.66	59.30	6183.1	6383.6	0.00	23.07	10.90
musk1	DeIA	1499.7	1494.1	100.00	77.51	90.40	1973.5	1973.5	0.00	22.14	8.71
newthyroid	Orig	2.8		95.04	91.19	5.00	0.0		0.13	6.4	0.00
newthyroid	Int	3.6		94.99	90.74	4.70	0.1		0.18	8.8	0.23
newthyroid	AddB	2.9	0.1	95.04	91.19	5.00	0.0	0.0	0.13	6.4	0.00
newthyroid	DelB1	2.7	0.0	94.94	91.19	4.60	0.0	0.0	0.17	6.4	0.27
newthyroid	DelB2	2.7	0.1	94.94	91.19	4.60	0.0	0.0	0.17	6.4	0.27
newthyroid	AddA	2.9	0.1	95.04	91.19	4.90	0.0	0.0	0.13	6.4	0.10
newthyroid	DeIA	2.9	0.1	95.04	91.65	4.90	0.0	0.0	0.13	8.2	0.10
pima	Orig	1776.2		80.67	78.12	8.00	2218.4		0.25	22.11	0.00
pima	Int	1560.2		80.44	77.34	6.20	6066.0		0.27	17.73	0.62
pima	AddB	1774.3	1.9	80.67	78.12	8.00	2102.0	0.0	0.25	22.11	0.00
pima	DelB1	1122.2	0.5	77.08	75.25	1.80	29059.6	0.0	1.57	13.17	0.18
pima	DelB2	1276.4	1.8	78.31	75.12	3.00	1040.2	0.0	0.17	23.21	0.00
pima	AddA	1760.1	1.9	80.67	78.25	7.90	2247.2	0.0	0.25	22.60	0.10
pima	DeIA	1770.2	0.3	80.67	78.25	7.90	2074.1	0.0	0.25	22.60	0.10

		mean					variance				
data set	alg.	train	feat	train	test	train	train	feat	train	test	train
		time	time	acc	acc	feat	time	time	acc	acc	feat
sonar	Orig	0.4		100.00	72.10	60.00	0.0		0.00	71.0	0.00
sonar	Int	34.6		92.47	74.43	10.90	8.2		1.42	57.1	2.54
sonar	AddB	30.8	30.5	100.00	74.43	34.70	20.0	20.1	0.00	163.0	19.12
sonar	DelB1	3.5	3.1	100.00	78.81	41.40	0.0	0.0	0.00	53.2	10.27
sonar	DelB2	57.1	56.7	100.00	78.81	41.40	35.4	35.5	0.00	53.2	10.27
sonar	AddA	30.8	30.3	100.00	74.43	34.70	19.6	19.8	0.00	163.0	19.12
sonar	DelA	20.4	20.1	100.00	78.81	41.40	4.9	5.6	0.00	53.2	10.27
vehicle	Orig	61.9		98.90	94.94	18.00	183.2		0.04	12.9	0.00
vehicle	Int	234.1		98.52	95.88	11.20	1216.6		0.04	8.69	1.51
vehicle	AddB	653.5	8.2	84.24	81.65	7.20	251201.6	92.6	162.63	92.95	86.40
vehicle	DelB1	65.0	3.0	98.51	95.88	12.10	162.2	0.0	0.05	5.3	0.32
vehicle	DelB2	71.5	14.6	98.71	95.29	13.40	93.5	6.5	0.03	8.9	1.82
vehicle	AddA	77.0	14.4	93.96	91.06	13.30	283.1	47.6	106.54	92.0	50.90
vehicle	DelA	64.5	1.9	98.90	94.94	16.40	188.6	0.4	0.04	13.5	2.49
wdbc	Orig	0.6		100.00	93.67	30.00	0.0		0.00	12.5	0.00
wdbc	Int	27.9		98.67	97.53	4.60	6.3		0.05	2.2	0.27
wdbc	AddB	27.6	27.3	100.00	95.43	21.10	33.4	32.6	0.00	9.8	21.21
wdbc	DelB1	4.3	3.7	100.00	95.08	21.40	0.2	0.1	0.00	11.6	18.93
wdbc	DelB2	30.2	29.7	100.00	95.08	21.40	93.5	93.1	0.00	11.6	18.93
wdbc	AddA	28.4	27.2	100.00	95.43	21.10	33.2	33.1	0.00	9.8	21.21
wdbc	DelA	7.6	6.6	100.00	94.90	21.40	2.6	2.8	0.00	11.3	18.93
wine	Orig	0.1		100.00	93.27	13.00	0.0		0.00	19.9	0.00
wine	Int	0.3		99.56	96.60	2.10	0.0		0.09	15.4	0.10
wine	AddB	0.2	0.2	100.00	96.60	2.80	0.0	0.0	0.00	8.6	0.18
wine	DelB1	0.2	0.1	100.00	96.60	3.30	0.0	0.0	0.00	15.4	1.12
wine	DelB2	0.9	0.9	100.00	96.60	3.30	0.0	0.0	0.00	15.4	1.12
wine	AddA	0.3	0.2	100.00	96.60	2.80	0.0	0.0	0.00	8.6	0.18
wine	DelA	0.4	0.3	100.00	96.60	3.30	0.0	0.0	0.00	15.4	1.12
wdbc	Orig	6.5		97.48	75.71	32.00	0.9		0.09	20.6	0.00
wdbc	Int	14.6		89.00	83.45	4.30	47.2		12.40	24.8	18.01
wdbc	AddB	7.6	0.3	85.68	85.05	0.20	0.6	0.0	0.22	8.7	0.40
wdbc	DelB1	4.7	0.6	85.85	84.08	0.60	1.5	0.0	0.48	17.7	1.60
wdbc	DelB2	14.6	8.8	95.88	75.66	19.70	0.6	0.3	0.21	66.3	1.34
wdbc	AddA	7.0	0.4	85.57	85.58	0.00	0.9	0.0	0.05	4.4	0.00
wdbc	DelA	9.3	2.8	97.48	76.26	26.00	0.7	1.1	0.09	20.3	10.67

TABLE 2: TEN-FOLD CROSS-VALIDATION DATA

3.1 Feature Selection Prior to Hyperplane Placement

Three new algorithms select features using the *SINF*-based filter prior to placing the separating hyperplane. *AddB* is a sequential forward selection algorithm, and *DelB1* and *DelB2* are two variants of sequential backward elimination algorithms. Table 3 compares all three to the original algorithm which does not use feature selection. $\Delta\% \text{acc}$ is the difference in ten-fold average total accuracy on the testing set between the method and the *Orig* algorithm; Δfeat is the difference in the ten-fold average number of features between the method and the *Orig* algorithm.

data set	AddB		DelB1		DelB2	
	$\Delta\% \text{acc}$	Δfeat	$\Delta\% \text{acc}$	Δfeat	$\Delta\% \text{acc}$	Δfeat
breast1	0.00	0.00	0.29	-3.20	-0.29	-2.70
bupa	-9.87	-3.70	-1.47	-3.80	1.14	-2.00
glass	-2.86	-7.80	0.37	-7.10	5.58	-2.40
iris	0.00	0.00	-6.00	-2.00	-4.67	-1.70
musk1	13.02	-106.70	9.67	-75.60	9.67	-75.60
newthyroid	0.00	0.00	0.00	-0.40	0.00	-0.40
pima	0.00	0.00	-2.87	-6.20	-3.00	-5.00
sonar	2.33	-25.30	6.71	-18.60	6.71	-18.60
vehicle	-13.29	-10.80	0.94	-5.90	0.35	-4.60
wdbc	1.75	-8.90	1.40	-8.60	1.40	-8.60
wine	3.33	-10.20	3.33	-9.70	3.33	-9.70
wdbc	9.34	-31.80	8.37	-31.40	-0.05	-12.30
<i>average</i>	<i>0.31</i>	<i>-17.10</i>	<i>1.73</i>	<i>-14.38</i>	<i>1.68</i>	<i>-11.97</i>

TABLE 3: SELECTING FEATURES BEFORE PLACING HYPERPLANE. DIFFERENCES RELATIVE TO ORIG ALGORITHM.

On average, all three methods reduce the number of features, as expected, sometimes by a significant number (e.g. by 106.7 features on average for *AddB* on the *musk1* data set). However it is surprising to see that all three methods also increase the % total accuracy on average as well. While *DelB1* and *DelB2* both reduce the number of features on average in every data set, *AddB* fails to do so for 4 data sets. Overall, the best of the methods is *DelB1* which has the highest average increase in ten-fold testing accuracy and removes the second largest number of features on average. *DelB1* removes 14.38 features of the 30.00 average features in the data sets. The time required for the feature selection using *DelB1* is very small, ranging from 0.01 seconds to 99.74 seconds over the data sets, with a geometric mean of 0.5 seconds. The geometric mean times for the other two methods are also small, though slightly larger.

However, a closer analysis shows that *AddB* is able to remove many more features for the four data sets for which it is easy to achieve a 100% linear separation in the training set, while increasing the ten-fold training accuracy about as much as the other methods. *AddB* performs well in this case because it is able to terminate the process of adding features as soon as *SINF* reaches zero, i.e. as soon as enough features are added that a complete separation is found.

DelB1 remains the best method for data sets that are not easily separated in training.

So *DelB1* with $P=5$ is recommended when there is no prior information about the ability of the *Orig* algorithm to place a 100% accurate separating hyperplane. If it is known beforehand that such a hyperplane exists, then *AddB* is the method of choice.

3.2 Feature Selection After Hyperplane Placement

Two new algorithms attempt to reduce the number of features after a hyperplane has already been placed, while making sure that all of the points correctly classified by the original placement continue to be correctly classified as the number of features is reduced. The two methods are *AddA* (sequential forward selection) and *DelA* (sequential backward selection).

Results are summarized in Table 4, with the change in 10-fold cross-validation test set accuracy and number of features shown relative to the 10-fold test set results for the *Orig* algorithm. In a few cases the change in 10-fold accuracy is negative, indicating that the reduced-features hyperplane has lower accuracy compared to the *Orig* algorithm, even though the reduced-features hyperplane successfully classifies all of the training set points that were successfully classified when the *Orig* algorithm used all of the features. Overall the results are not dissimilar to those shown in Table 3 for selecting features before placing the hyperplane.

data set	AddA		DelA	
	Δ %acc	Δ feat	Δ %acc	Δ feat
breast1	-0.29	-0.60	-0.14	-0.90
bupa	-3.46	-1.90	0.00	-0.20
glass	0.00	0.00	-0.02	-0.20
iris	0.00	0.00	0.00	0.00
musk1	13.02	-106.70	9.88	-75.60
newthyroid	0.00	-0.10	0.45	-0.10
pima	0.13	-0.10	0.13	-0.10
sonar	2.33	-25.30	6.71	-18.60
vehicle	-3.88	-4.70	0.00	-1.60
wdbc	1.75	-8.90	1.23	-8.60
wine	3.33	-10.20	3.33	-9.70
wdbc	9.87	-32.00	0.55	-6.00
<i>average</i>	<i>1.90</i>	<i>-15.88</i>	<i>1.84</i>	<i>-10.13</i>

TABLE 4: SELECTING FEATURES AFTER PLACING HYPERPLANE. DIFFERENCES RELATIVE TO ORIG ALGORITHM.

The best results are again reached for those data sets in which the *Orig* algorithm is able to obtain a 100% separation (*musk1*, *sonar*, *wdbc*, *wine*). This allows the *AddA* method to stop adding features as soon as the complete separation is achieved; and it also stops the *DelA* method from deleting further features. Note that these good results are achieved despite the lack of inherent ordering of the features when a complete separation is available.

The time required for feature selection after hyperplane placement is usually quite small. The geometric mean of the ten-fold averages is 1.75 seconds for *AddA*, and 1.01 seconds for *DeIA*. The single outlier in both cases is the 166-feature *musk1* data set.

3.3 Integrated Feature Selection

The *Int* algorithm integrates hyperplane placement and feature selection, as shown in Alg. 6. The 10-fold cross-validation test set accuracies and numbers of features produced by this algorithm relative to the *Orig* algorithm are summarized in Table 5. Accuracy is increased on average an amount similar to the best of the methods for selecting features before and after hyperplane placement, but the average reduction in number of features is significantly larger as compared to all other methods. The average number of features is reduced for every data set, but most remarkably for *musk1*, *sonar*, and *wdbc*, where the number of features is dramatically smaller than found by any other method.

data set	Int	
	Δ %acc	Δ feat
breast1	0.44	-4.50
bupa	0.00	-0.20
glass	0.89	-5.40
iris	-9.33	-1.90
musk1	13.65	-141.30
newthyroid	-0.45	-0.30
pima	-0.77	-1.80
sonar	2.33	-49.10
vehicle	0.94	-6.80
wdbc	3.86	-25.40
wine	3.33	-10.90
wdbc	7.74	-27.70
<i>average</i>	<i>1.89</i>	<i>-22.94</i>

TABLE 5: INTEGRATED HYPERPLANE PLACEMENT AND FEATURE SELECTION. DIFFERENCES RELATIVE TO ORIG ALGORITHM.

We also examined the effect of allowing the introduction of a feature to cause a small worsening in *SINF*, as is done in the *DeIB1* and *DeIB2* algorithms. This did not provide consistent improvements.

3.4 Discussion

All six of the new methods for feature selection improve over the original algorithm, increasing accuracy on average and reducing the number of features on average. It is unrealistic to expect both an increase in accuracy and a reduction in the number of features in every case, and is in fact frequently useful to accept a small decrease in accuracy in trade for a significant reduction in the number of features, thus the two factors must be considered simultaneously. Figure 1 summarizes the trade-offs between the ten-fold average number of features and the ten-fold testing set average accuracy, relative to the best result for each data set. The data is analyzed

as follows. The highest ten-fold average accuracy returned by any method and the smallest ten-fold average number of features returned by any method are separately identified for each data set, and the differences to those two best values are calculated for all methods. The average ten-fold differences from the best results over all of the models are plotted for each method in Figure 1.

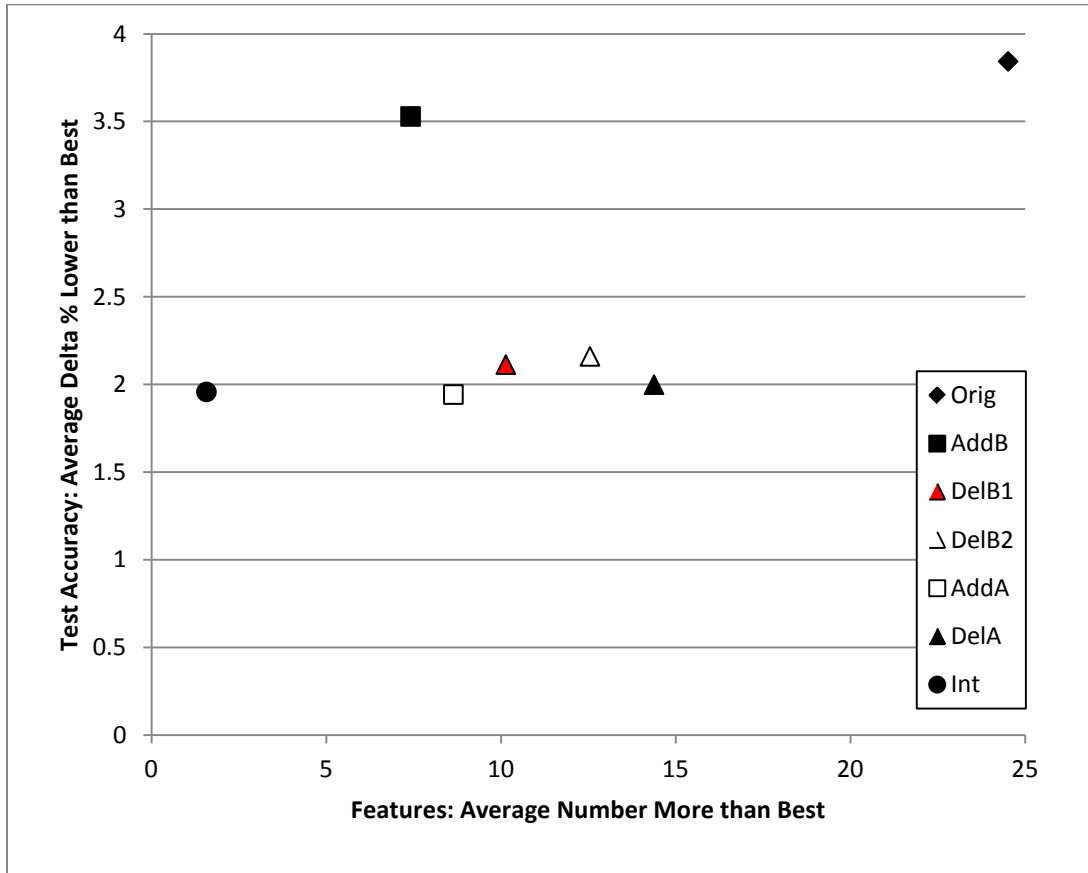


FIGURE 1: AVERAGE PERFORMANCE RELATIVE TO BEST

A perfect method would always have the smallest number of features and the highest accuracy, and hence would be at (0,0) in Figure 1. The closer a method is to (0,0), the better it is. The figure shows that all six of the new methods improve on the *Orig* method, having both better ten-fold testing accuracy on average and fewer ten-fold average features. Other patterns are visible. *AddB* is able to reduce the number of features significantly, but does not improve accuracy much on average. All of the other methods produce roughly similar improvements in ten-fold average accuracy, but at a range of numbers of features. While *AddA*, *DelB1*, *DelB2* and *DelA* cluster between 8.6 and 14.4 features more than best ten-fold average, *Int* is significantly better, averaging just 1.6 features more than best ten-fold average, while averaging just 1.96 percentage points worse than best ten-fold average accuracy.

Comparisons to other methods in the literature are difficult primarily due to differing selection of data sets. However there are a few data sets in common with work by Bradley and Mangasarian (1998) and Fung and Mangasarian (2004). Results for the *Int* method tend to be

slightly better than those reported by Bradley and Mangasarian (2004) and comparable to the linear methods in Fung and Mangasarian (2004). Results for *Int* are much better than those reported by Dunbar et al [2010] for the *wdbc* data set, and slightly worse for the *pima* data set.

The time required for feature selection is generally a small fraction of the total time (feature selection plus hyperplane placement) for most data sets. This is not the case for the *musk1*, *sonar*, *wdbc*, and *wine* data sets. These are the data sets which are completely classified by a single hyperplane. This has the effect of reducing the time for hyperplane placement to a very small value, hence feature selection constitutes a large fraction of the total time.

The geometric means of the ten-fold cross-validation total training times (feature selection plus hyperplane placement) and the feature selection times alone are shown in Table 6. The feature selection time is not relevant for the *Orig* method, and cannot be separated out for *Int*. As can be seen by comparison with the time for the *Orig* algorithm, feature selection does add some time to the process. The feature selecting methods have similar training times, except for *DelB1*, which is significantly faster than the others. *DelB1* is also one of the recommended methods if features are to be selected prior to hyperplane placement. The feature selection times are very small, with geometric means in the range of 0.50 to 2.86 seconds, but the reduced number of features often increases the training time because the time to place the separating hyperplane by the *Orig* method increases, usually because the number of LPs that must be solved increases. All times could be reduced significantly by an efficient implementation of the algorithm and the use of a compiled language.

method	geo. mean of training time (s)	geo. mean of feature selection time (s)
Orig	7.67	-
Int	34.32	-
AddB	30.15	1.26
DelB1	13.73	0.50
DelB2	38.45	2.86
AddA	28.41	1.75
DelA	26.38	1.01

TABLE 6: GEOMETRIC MEANS OF TOTAL TRAINING TIME AND FEATURE SELECTION TIME

Note that the small feature selection times in Table 6 are independent of the hyperplane placement method. In other words, the feature selection times will be the same even if some other method of hyperplane placement (support vector machine etc.) is used. These feature selection methods are very fast because they solve a sequence of linear programs in which each LP model is very similar to the previous one, so advanced start techniques are effective.

Finally, the ten-fold cross-validation experiments were re-run using the fast version of all algorithms which selects a single constraint representing a data point for the list of candidates to consider for removal while placing the hyperplane. The fast version chooses the constraint having the largest value of $e_i|\sigma(e_i)|$ from among constraints having $e_i > 0$. This had minimal impact on the testing accuracy (an average drop of 0.39 percentage points across all methods)

and number of features selected (an average reduction of 0.03 features). However training time was reduced by about 60% on average, because the number of training LPs solved was reduced by the same fraction.

4. Conclusions

This paper makes a number of contributions:

- A new and effective filtering criterion for use in feature selection is introduced: reduction in the sum of infeasibilities (*SINF*). Feature selection methods based on this criterion are very quick. Used before a separating hyperplane is placed, this criterion improves the average accuracy and reduces the number of features as compared to an existing hyperplane placement method.
- New methods for selecting features after a separating hyperplane has already been found are introduced. The *AddA* and *DelA* methods preserve the training set separation found by the original hyperplane. These methods also improve the average accuracy and reduce the number of features as compared to an existing hyperplane placement method.
- A new integrated method that selects features at the same time as placing a separating hyperplane is introduced. The *Int* method is the best of all methods tested, reducing the number of features more than all other methods, on average, and increasing the testing accuracy as compared to an existing hyperplane placement method.

While a particular *SINF*-based method for placing separating hyperplanes has been used throughout this paper, it is important to note that the feature selection methods (with the exception of *Int*) can be used with *any* hyperplane placement method. The times for feature selection alone are small, so the additional overhead will be minimal. Overall, the best results are returned by the integrated *Int* method, which requires the use of the *SINF*-based method for hyperplane placement. However if feature selection is to be performed before or after hyperplane placement, then any hyperplane placement method can be used.

Recommendations in this case are: (i) for feature selection beforehand use *AddB* if the data set is known to be completely classified by a single hyperplane, and use *DelB1* otherwise, and (ii) for feature selection afterwards, use the *AddA* method.

Finally, the speed of these algorithms can be improved dramatically by several techniques. First, the prototype algorithms implemented for this paper omit several steps that have a large impact on efficiency, e.g. recording the list of constraints to which the objective function is sensitive when a new minimum *SINF* is identified, which means that the LP does not need to be re-solved at the end of the loop. Such efficiency-enhancing steps should be included. Second, the algorithms should be re-implemented in a compiled rather than interpreted language, and third, a more efficient LP solver than the one provided in Octave should be used. These are tasks for future research.

References

- Amaldi E (1994). **From Finding Maximum Feasible Subsystems Of Linear Systems To Feedforward Neural Network Design**. Ph.D. thesis no. 1282, Département de Mathématiques, École Polytechnique Fédérale de Lausanne, Switzerland.
- Amaldi E, Kann V (1995). *The Complexity And Approximability Of Finding Maximum Feasible Subsystems Of Linear Relations*, Theoretical Computer Science 147:181-210.
- Bennett KP, Bredensteiner E (1997). *A Parametric Optimization Method for Machine Learning*, INFORMS J. on Computing 9:311-318.
- Bradley PS, Mangasarian OL (1998). *Feature Selection Via Concave Minimization And Support Vector Machines*, in J. Shavlik, editor, Proceedings of the International Conference on Machine Learning, pages 82–90, San Francisco, California, 1998. Morgan Kaufmann Publishers.
- Bradley PS, Mangasarian OL, Street WN (1998). *Feature Selection via Mathematical Programming*, INFORMS Journal on Computing 10:209-217.
- Bredensteiner EJ, Bennett KP (1997). *Feature Minimization Within Decision Trees*. Computational Optimization and Applications, 10:110–126.
- Brown G, Graves G (1975). *Elastic Programming: A New Approach To Large-Scale Mixed Integer Optimisation*, ORSA/TIMS conference, Las Vegas.
- Chakravarti N (1994). *Some Results Concerning Post-Infeasibility Analysis*, European Journal of Operations Research 73:139-143.
- Chinneck JW (1996). *An Effective Polynomial-Time Heuristic for the Minimum-Cardinality IIS Set-Covering Problem*, Annals of Mathematics and Artificial Intelligence 17:127-144.
- Chinneck JW (2001). *Fast Heuristics for the Maximum Feasible Subsystem Problem*, INFORMS Journal on Computing 13:210-223.
- Chinneck JW (2008). **Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods**, Vol. 118, International Series in Operations Research and Management Sciences, Springer Science+Media LLC.
- Chinneck JW (2009). *Tailoring Classifier Hyperplanes to General Metrics*, in J.W. Chinneck, B. Kristjansson, M. Saltzman (eds.) **Operations Research and Cyber-Infrastructure**, Springer Science+Media LLC.
- Cristianini N, Shawe-Taylor J (2000). **An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods**, Cambridge University Press, Cambridge, UK.
- Dash M, Liu H (1997). *Feature Selection for Classification*, Intelligent Data Analysis 1:131-156.
- Dunbar M, Murray JM, Cysique LA, Brew BJ, Vaithilingam J (2010). *Simultaneous Classification and Feature Selection Via Convex Quadratic Programming With Application to HIV-Associated Neurocognitive Disorder Assessment*, European Journal of Operational Research 206: 470–478.
- Frank A, Asuncion A. (2010). *UCI Machine Learning Repository* [<http://archive.ics.uci.edu/ml>]. University of California, School of Information and Computer Science, Irvine, CA.

Fung GM, Mangasarian OL (2004). *A Feature Selection Newton Method for Support Vector Machine Classification*, Computational Optimization and Applications 28:185-202.

GLPK (2011). Homepage for the Gnu Linear Programming Toolkit:
<http://www.gnu.org/software/glpk/glpk.html>

Guo G, Dyer CR (2003). *Simultaneous Feature Selection and Classifier Training via Linear Programming: A Case Study for Face Expression Recognition*, Proceedings of the 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'03).

Jokar S, Pfetsch ME (2008). *Exact and Approximate Sparse Solutions of Underdetermined Linear Equations*, Siam Journal on Scientific Computation 31:23-44.

Liu H, Yu L (2005). *Toward Integrating Feature Selection Algorithms for Classification and Clustering*, IEEE Transactions on Knowledge and Data Engineering 17:491-502.

Maldonado S, Weber R, Basak J (2011). *Simultaneous Feature Selection and Classification Using Kernel-Penalized Support Vector Machines*, Information Sciences 181:115-128.

Mangasarian OL (1994). *Misclassification Minimization*, Journal of Global Optimization 5:349-360.

Mittelmann H (2011). Benchmark of Serial LP solvers, <http://plato.asu.edu/ftp/lpfree.html>, accessed April 15, 2011.

Octave (2011). Octave home page: <http://www.gnu.org/software/octave/>.

Parker MR (1995). **A set covering approach to infeasibility analysis of linear programming problems and related issues**. Ph.D. thesis, Dept. of Mathematics, University of Colorado at Denver, Denver, Colorado.

Parker MR, Ryan J (1996). *Finding the Minimum Weight IIS Cover of an Infeasible System of Linear Inequalities*, Annals of Mathematics and Artificial Intelligence 17:107-126.

Sankaran JK (1993). *A Note On Resolving Infeasibility In Linear Programs By Constraint Relaxation*, Operations Research Letters 13:19-20.